Distributed Systems

DAT520 - Spring 2024

Introduction

Prof. Hein Meling

University of Stavanger

Distributed Systems
A Whirlwind Tour

* Computing

DISTRIBUTED COMPUTING

DECENTRALIZED COMPUTING

CENTRALIZED COMPUTING

AUTONOMOUS COMPUTING

PARALLEL COMPUTING

VOLUNTEER COMPUTING

CLUSTER COMPUTING

TRUSTWORTHY COMPUTING

GRID COMPUTING

SERVICE-ORIENTED COMPUTING

UTILITY COMPUTING

CLOUD COMPUTING

PERVASIVE COMPUTING

EDGE COMPUTING

UBIQUITOUS COMPUTING

FOG COMPUTING

SERVERLESS COMPUTING

What is this?

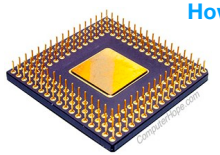
What is this?

Cray-1 supercomputer (1975)
160 MFLOPS

Cori @ Berkeley Lab

Paradigms in Computing

How do we increase speed?



4 GHz processor performs
4,000,000,000 clock cycles per second

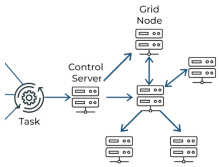
High Performance Computing (HPC)

Parallel Computing	Massively parallel processors are made by chaining together hundreds or thousands of inexpensive commercial microprocessors.
1970s	

Electric Power Grid ~ Grid / Utility Computing

Paradigms in Computing

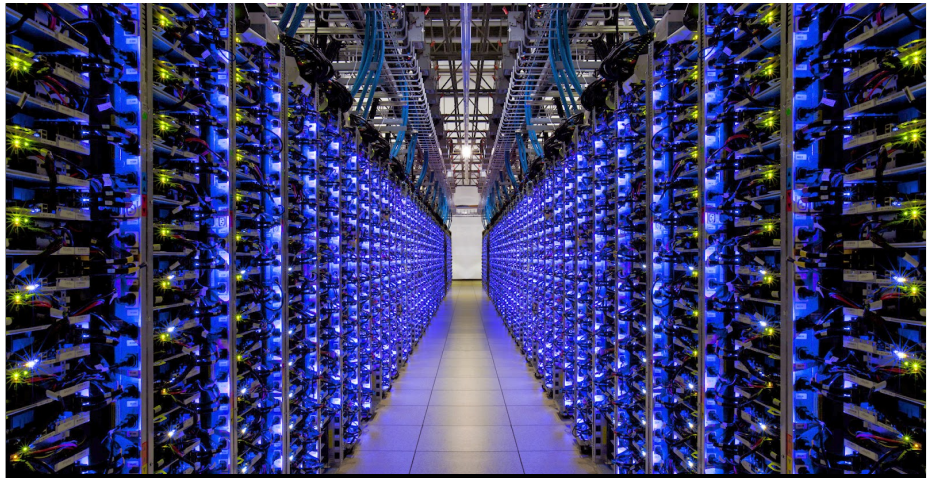
How do we reduce cost of computing, increase reliability, and increase flexibility?



- Open Science Grid (US Funded)
- European Grid Infrastructure

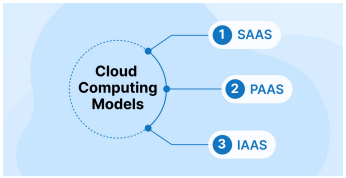
High Performance Computing (HPC)

Grid Computing **1990s**	The grid combines various computing resources, databases, and measuring devices that comprise a pool of resources for coordinated, integrated and flexible shared use.



Cloud Computing

Paradigms in Computing



How do we scale?

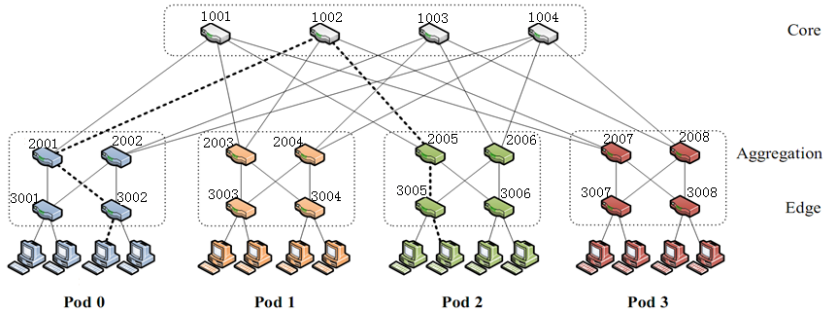
Infrastructure as a Service (IaaS)	Amazon EC2 / S3
Platform as a Service (PaaS)	Google App Engine
Software as a Service (SaaS)	Salesforce

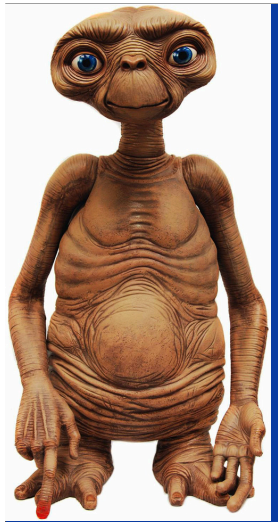
*** as a Service**

Cloud Computing **(today)**	Economies of scale: a pool of abstracted, virtualized, dynamically-scalable, computing power, storage, platforms, and services are delivered on demand over the Internet.

How can we connect tens of thousands of machines in a data center together?

Small Fat Tree Example

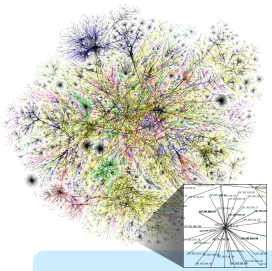




SETI HOME

SETI@home was a scientific experiment, based at UC Berkeley, that used Internet-connected computers in the Search for Extraterrestrial Intelligence (SETI). You could participate by running software that downloads and analyzes radio telescope data.

Paradigms in Computing



How do we scale?

Millions of *unreliable* and *heterogeneous* machines

High Performance Computing (HPC)

Volunteer Computing

Internet-connected computers, volunteered by their owners, as a source of computing power and storage.

Distributed Systems

Computer Networks: the autonomous computers are explicitly visible and must be specifically addressed

Distributed systems: the multiple autonomous computers and components are transparent to the user

Parallel computers: single system view without physical separation

Many problems in common:
- Scheduling
- Load balancing
- Resource and data sharing/distribution

Networks are in some sense also distributed systems (e.g. name services) and every distributed system relies on services provided by a network

Distributed systems may serve the same purpose as parallel computers: high performance

Distributed Systems Examples

Distributed Systems - Examples

Web Search

Search engines need to index the entire content of the web.

Google's index contains **hundreds of billions of webpages**.

- physical infrastructure of networked computers across data centers
- distributed file system that supports very large files
- distributed storage system with very large datasets
- distributed locking and agreement
- a programming model for very large parallel and distributed computation

Google YAHOO! Yandex Aol. Ask.com Baidu 百度 Bing DuckDuckGo

Distributed Systems - Examples

Generative Model Learning and Inference

AI models also need to encode the entire content of the web.

- massive parallel processing capabilities: CPUs, GPUs, Tensor PUs, Neural Engine
- efficient data pipelines: for feeding data into training and inference processes
- scalable network infrastructure: to handle high volume of data transfer between nodes during training and inference
- scalable model serving infrastructure: to handle thousands of simultaneous inference requests

Vicuna
Open source

Distributed Systems Definitions

"A system in which hardware or software components located at networked computers **communicate and coordinate their actions** only by message passing." [Coulouris]

"A distributed system is a collection of independent computers that **appear to the users of the system as a single computer**." [van Steen/Tanenbaum]

Distributed Systems - Definitions

What they have in common?

- Distributed hardware (computers)
- Distributed data
- Distributed control
- Connected through some network

"A distributed system is one in which the **failure of a machine you have never heard of can cause your own machine to become unusable.**" [Lamport]

Why Distributed Systems?

Inherent Distribution

Distribution as an Artifact

Distributed Systems: Why?

A distributed system should

- make resources easily accessible;
- hide the fact that resources are distributed across a network;
- be open;
- be scalable.

Distributed Systems: Why?

Make resources easily accessible

Example resources: compute and storage facilities, data, files, services, and networks.

There are many reasons for wanting to share resources.

One obvious reason is that of economics.

Distributed Systems: Why?

Hide that resources are distributed across a network

Transparency	Description
Access	Hide differences in data representation and how an object is accessed
Location	Hide where an object is located **(Naming)**
Relocation	Hide that an object may be moved to another location while in use
Migration	Hide that an object may move to another location
Replication	Hide that an object is replicated
Concurrency	Hide that an object may be shared by several independent users
Failure	Hide the failure and recovery of an object

Distributed Systems: Why?

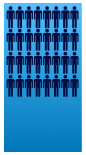
Be open

Interoperability, composability, and extensibility

An **open distributed system** is a system that offers components that can **easily be used** by, or **integrated into** other systems.

Often **itself composed** of components that **originate from elsewhere**.

Distributed Systems: Why?



Be scalable

Three scalability dimensions

- Size scalability
- Geographical scalability
- Administrative scalability

Distributed Systems: Why?

Be scalable

Three scalability dimensions

Size scalability

- more users
- more resources

Potential bottlenecks:

computation, storage, and network capacity

Distributed Systems: Why?

Be scalable

Three scalability dimensions

Geographical scalability

Users and resources may lie far apart, but the fact that communication delays may be significant is hardly noticed.

Portability to wide area networks: synchronous communications, reliability, bandwidth, multipoint communication.

Distributed Systems: Why?

Be scalable

Three scalability dimensions

Administrative scalability

Can still be easily managed even if it spans many independent administrative organizations.

Scalability (continued)

Scalability Problems

Concept	Example
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

Scaling Techniques

Hiding communication latencies
- geographical scalability — Move stuff closer to consumer (e.g., use a CDN)
- asynchronous communication — Don't wait for responses; do something else while waiting

Partition and distribution — Partition data and computation across many computers

Replication — Make copies of data (and computation) available at many machines
- availability
- load balance
- latency

Scalability Problems

Inconsistencies & Global synchronization

- **Observation 1**: Applying scaling techniques is easy, except:
  - Having multiple copies (cached or replicated) leads to **inconsistencies**; modifying one copy makes it different from other copies
  - Always keeping copies consistent requires **global synchronization** of every modification
  - Global synchronization does not scale

Scalability Problems

Inconsistencies & Global synchronization

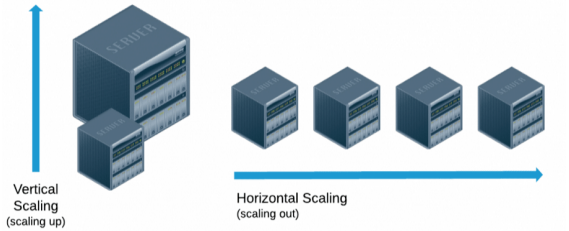
- **Observation 2**: Applying scaling techniques is easy, except:
  - We may sometimes **tolerate inconsistencies**; hence reduce the need for global synchronization
  - Tolerating inconsistencies is **application dependent**

Scaling Techniques

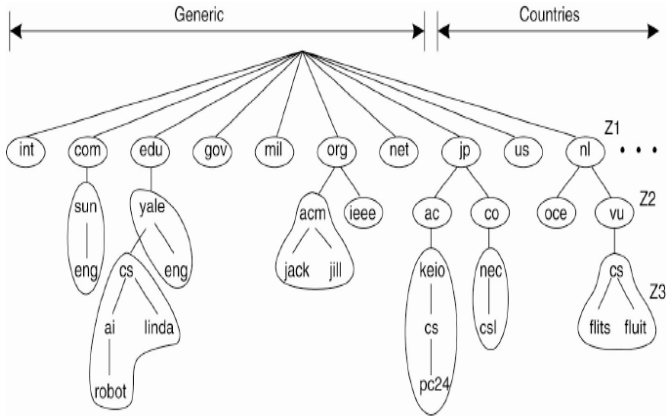
Performance problems due to limited (server & network) capacity

Scaling up (vertical)
improve capacity

Scaling out (horizontal)
add more machines



Scaling Techniques: Example



Failure Handling & Concurrency

Failure Handling

Challenge: Consistency, Consistency…

- Failure detection
  - Checksum, timeout
- Failure masking (also called Fault tolerance)
  - Message retransmission
  - Failover to another server (requires replicated servers)
- Failure recovery
  - Rollback a transaction

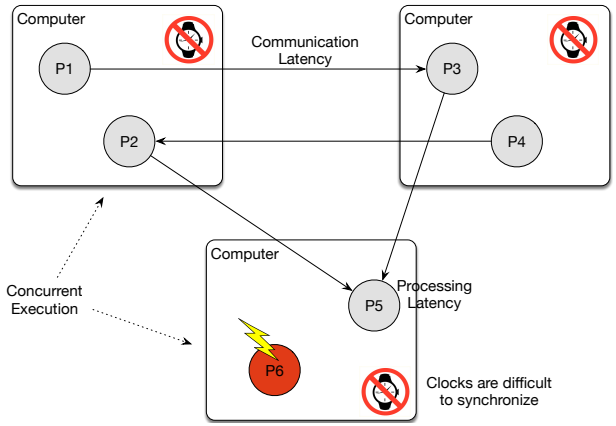
Concurrency

Challenge: Consistency, Consistency…

- Execute programs in parallel on different machines
  - Increase efficiency
- Need to synchronize access to shared resources
  - Across the network
  - To avoid conflicting updates (consistency!)

Characteristics of a Distributed System

Characteristics of a Distributed System



Characteristics of a Distributed System

Summary of characteristic challenges

- Concurrent execution of programs
- Communication latency
- Processing latency

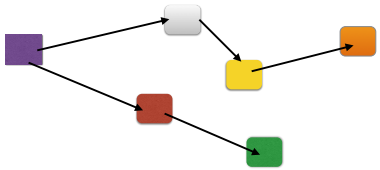
Cannot distinguish!

- Lack of a global clock
- Components can fail
  - Can lead to partial system failures

Characteristics of a Distributed System

Dependencies cause failures

- Dependencies between distributed components may exacerbate the impact of failures
- Distributed systems **complicates** the life of **developers**

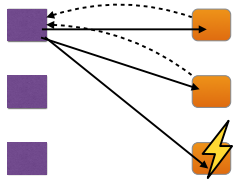


Dependency graph

Characteristics of a Distributed System

Independencies of failures

- Replication: multiple independent processes **enables** fault tolerance
- Distributed systems **should simplify** the life of **users**



Replication can hide failures!

Characteristics of a Distributed System

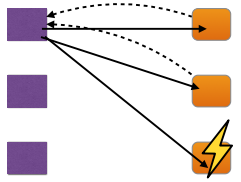
No globally synchronized clock

- Cannot use timestamps
  - To **accurately** synchronize and coordinate actions of different components

Characteristics of a Distributed System

Partial failures

- A subset of processes (components) can
  - fail or
  - become disconnected.
- How should remaining processes behave?
  - ★ nothing bad happens
  - ★ something useful (eventually) happens



Replication can hide failures!

Correctness

- **Safety** requires that nothing bad happens during execution
- **Liveness** requires that something good eventually happens during execution
  - ✦ the program's ability to make progress
  - ✦ termination is not necessarily a requirement for liveness

Safety Property

Property	The bad thing
Mutual exclusion	two processes executing in critical sections simultaneously
Deadlock freedom	deadlock
Livelock freedom	livelock
Partial correctness	terminating in a state that does not satisfy the postcondition after having been started in a state that satisfy the precondition
First come first serve	servicing a request that was made after one not yet serviced

Liveness Property

Property	The good thing
Starvation freedom	a process make progress infinitely often
Termination freedom	a program does not run forever
Guaranteed service	every request for service is satisfied eventually

Observation: Full distribution transparency may be too much

- **Completely hiding failures** is impossible
  - Cannot distinguish a slow computer from a failed one
  - Can never be sure that a server actually performed an operation before a crash
- Full transparency will **cost performance**, exposing distribution of the system
  - Keeping caches exactly up-to-date with master copy
  - Immediately writing to disk for fault tolerance

Distributed System Architectures

Distributed Communication Network

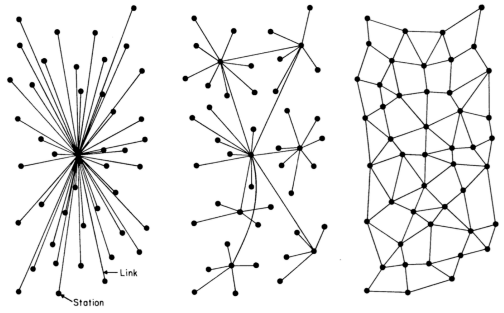


FIG. 1 — Centralized, Decentralized and Distributed Networks

Distributed System Architectures

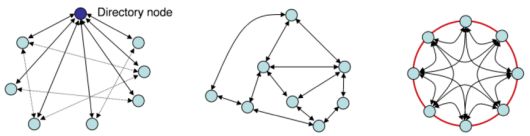
Centralized architecture

- client-server

Decentralized architecture

- Peer-to-peer (p2p)

Overlay Networks



Decentralized Architecture

Characteristics of Decentralized Algorithms

- No machine has complete information about the system state
- Machines make decisions based only on local information
- Failure of one machine does not ruin the algorithm
- There is no implicit assumption that a global clock exists

Pitfalls of Distributed System

Pitfalls when Developing Distributed Systems

Observation: Needless complexity due to false assumptions made by developers

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwidth is infinite
- Transport cost is zero
- There is one administrator

Questions?