

# Chapter 1: Introduction

# What is a Distributed System?

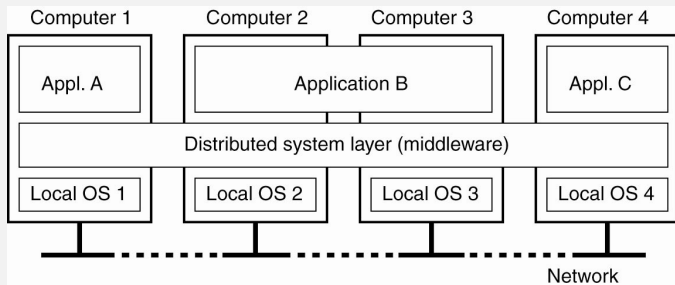
- ▶ A *distributed system* is a collection of independent computers that appears to its users as a single coherent system.

# Characteristics of a Distributed System (1)

- ▶ Collection of autonomous computing elements
  - ▶ Computing elements (or nodes) can be hardware and/or software
  - ▶ No **global clock**
  - ▶ Group membership is difficult: **Open groups** versus **Closed groups**
  - ▶ Often organized as an **overlay network**: structured, unstructured, peer-to-peer

## Characteristics of a Distributed System (2)

- ▶ Single coherent system
  - ▶ Requires **transparency**: process execution and data storage
  - ▶ Often implemented using **middleware**: a separate layer of software that is logically placed on top of the operating systems of the computers that are part of the distributed system.



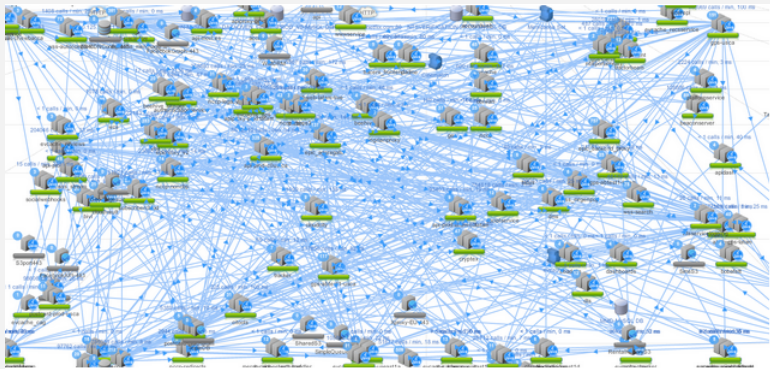
- ▶ It handles resource management, interapplication communication, masking of and recovery from failures, security and accounting services.

## Examples (1)

The image shows the Netflix logo, which consists of the word "NETFLIX" in a bold, white, sans-serif font. The letters are set against a solid red rectangular background. Each letter has a subtle 3D effect, with a dark shadow cast to its right and slightly downwards.

*The Internet is just a world passing around notes in a classroom. –Jon Stewart*

# Examples (1)



*The Internet is just a world passing around notes in a classroom. —Jon Stewart*

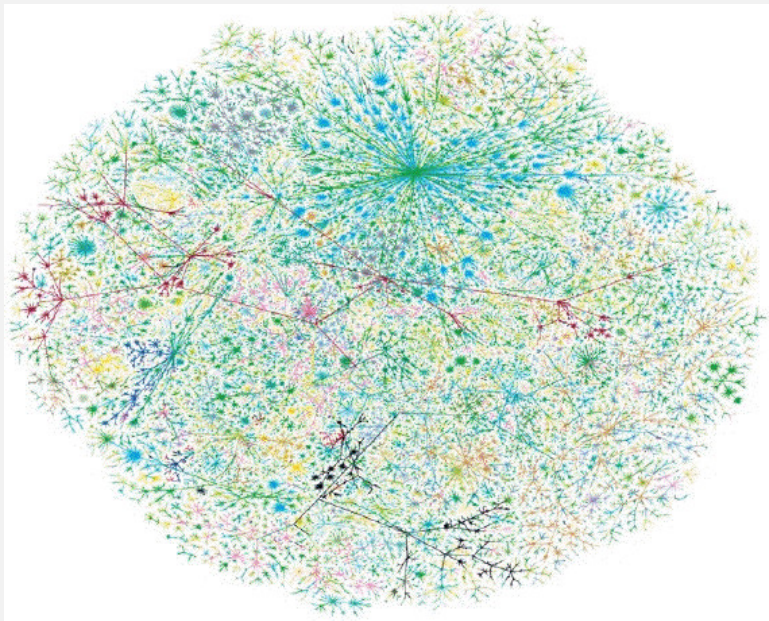
## Examples (2)



Google Search

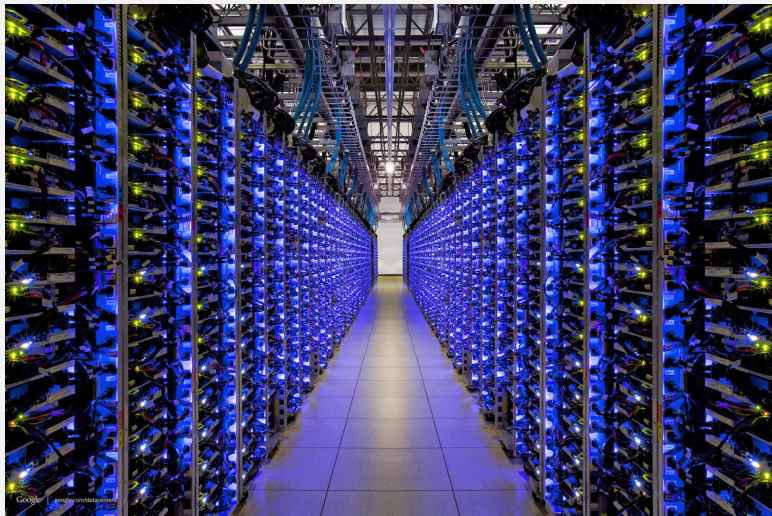
I'm Feeling Lucky

## Examples (2)





## Examples (2)



# Examples (3)

- ▶ Google search, GMail, Google Docs, et al
- ▶ Amazon
- ▶ Cloud storage, such as Dropbox et al
- ▶ Facebook, Twitter, EBay
- ▶ LinkedIn, WhatsApp, Instagram, Snapchat, TikTok
- ▶ Uber, Lyft
- ▶ Hadoop, Hive, Zookeeper, et al
- ▶ No-SQL Databases like HBase, Cassandra, MongoDB, et al
- ▶ **Distributed file systems**: NFS (Network File System), Google File System (GFS), Hadoop Distributed File System (HDFS), Amazon S3, Lustre, GlusterFS, Parallel Virtual File System (PVFS)
- ▶ **P2P: Peer to Peer**: Bit Torrent, Gnutella
- ▶ Domain Name System (DNS)
- ▶ Git: distributed version control system.
- ▶ SETI: a distributed computing project in which volunteers donate idle computer power to analyze radio signals for signs of extraterrestrial intelligence.
- ▶ Bitcoin: decentralized digital currency (and other blockchain applications)
- ▶ Virtually every substantial website!

# Goals of a Distributed System?

- ▶ Makes resources accessible
- ▶ Reasonably hides the fact that resources are distributed across a network (*Transparency*)
- ▶ Open
- ▶ Scalable

# Making Resources Accessible

- ▶ Benefits
  - ▶ Better economics by sharing expensive resources
  - ▶ Easier to collaborate and exchange information
  - ▶ Create virtual organizations where geographically dispersed people can work together using groupware
  - ▶ Enables electronic commerce
- ▶ Problems
  - ▶ Eavesdropping or intrusion on communication
  - ▶ Tracking of communication to build a profile

# Transparency

Type	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may move to another location while in use
Replication	Hide that a resource has multiple copies
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

# Degree of Transparency

Completely hiding the distribution aspects from users is not always a good idea in a distributed system.

- ▶ Attempting to mask a server failure before trying another one may slow down the system
- ▶ Expecting several replicas to be always consistent could degrade performance unacceptably
- ▶ For mobile and embedded devices, it may be better to expose distribution rather than trying to hide it
- ▶ Signal transmission is limited by the speed of light as well as the speed of intermediate switches

# Openness

An **open** distributed system offers services according to standard rules that describe the syntax and semantics of those services.

- ▶ Use of standard protocols.
- ▶ Services are described via **interfaces**, which are often describe via an **Interface Definition Language (IDL)**. Interfaces only specify syntax so semantics is left to the ambiguities of natural language.
- ▶ Interoperability, Portability, Extensibility.
- ▶ Separating policy from mechanism. For example: *caching* in a web browser.

# Scalability

Scalability can be measured against three dimensions.

- ▶ **Size**: be able to easily add more users and resources to a system
- ▶ **Geographical**: be able to handle users and resources that are far apart
- ▶ **Administrative**: be able to manage even if it spans independent administrative organizations

Centralized versus distributed implementations.



# Centralized Solutions with Scalability Problems

- ▶ Centralized services.
- ▶ Centralized data.
- ▶ Centralized algorithms.

# Distributed Approach

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.

**In-class exercise.** Simulate a centralized and a distributed algorithm for the same problem in class. The entire class will participate together in this exercise!



# Scaling Techniques

- ▶ **Hiding communication latencies**: Examples would be asynchronous communication as well as pushing code down to clients (E.g. Javascript)
- ▶ **Distribution**: Taking a component, splitting into smaller parts, and subsequently spreading them across the system. (E.g. Domain Name System)
- ▶ **Replication**: Replicating components increases availability, helps balance the load leading to better performance, helps hide latencies for geographically distributed systems. **Caching** is a special form of replication.

# Distributed System Development Pitfalls

False assumptions made by first time developer (*formulated by Peter Deutsch*).

- ▶ 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

“A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.”  
–Leslie Lamport

# In-class Exercise

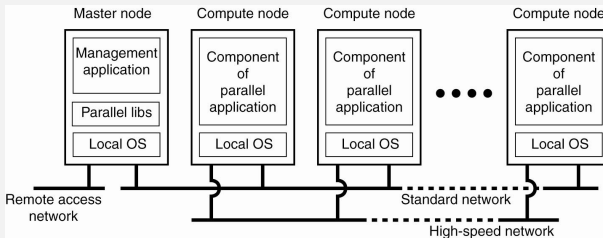
- ▶ Walk through architecture of various distributed systems ranging from: single server + multiple clients, multiple server + multiple clients, point to point and others.

# Types of Distributed Systems

- ▶ *Distributed Computing Systems*
  - ▶ Cluster Computing Systems
  - ▶ Grid Computing Systems
  - ▶ Cloud Computing
- ▶ *Distributed Information Systems*
  - ▶ Transaction Processing Systems
  - ▶ Enterprise Application Integration
- ▶ *Distributed Pervasive Systems*
  - ▶ Ubiquitous Computing Systems
  - ▶ Mobile Computing Systems
  - ▶ Sensor Networks

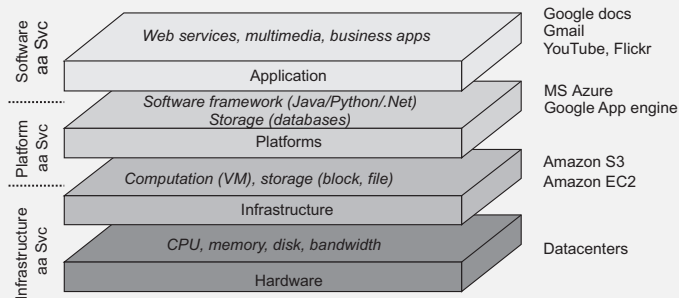
# Cluster Computing

- ▶ A **computer cluster** is primarily used to run a single program in parallel on multiple machines. The program relies on parallel libraries and frameworks such as MPI (Message Passing Interface), MapReduce and others.
- ▶ A *computer cluster* consists of a collection of compute and I/O nodes that are controlled and accessed from a single master node. The master allocates nodes to a parallel program, maintains a queue of submitted jobs and provides an interface to manage the cluster.
- ▶ A cluster usually has a dedicated high-speed interconnection network and an optional administrative network.



# Cloud Computing

- **Cloud Computing** provides the facilities to dynamically construct an infrastructure and compose what is needed from available services.





# Layers of Cloud Computing

- ▶ **Hardware:** Processors, routers, power and cooling systems. Customers normally never get to see these.
- ▶ **Infrastructure:** Deploys virtualization techniques. Evolves around allocating and managing virtual storage devices and virtual servers.
- ▶ **Platform:** Provides higher-level abstractions for storage and such. Example: Amazon S3 storage system offers an API for (locally created) files to be organized and stored in so-called buckets.
- ▶ **Application:** Actual applications, whether distributed or not. Includes apps such as office suites (text processors, spreadsheet applications, presentation applications). Comparable to the suite of apps shipped with traditional OSes.

# Integrating Applications

- ▶ **Situation:** Organizations confronted with many **networked applications**, but achieving interoperability was painful.
- ▶ **Basic approach:** A networked application is one that runs on a **server** making its services available to remote **clients**. Simple integration: clients combine requests for (different) applications; send that off; collect responses, and present a coherent result to the user.
- ▶ **Next step:** Allow direct application-to-application communication, leading to **Enterprise Application Integration (EAI)**.

# Example EAI: Transaction Processing Systems (1)

Transaction primitives:

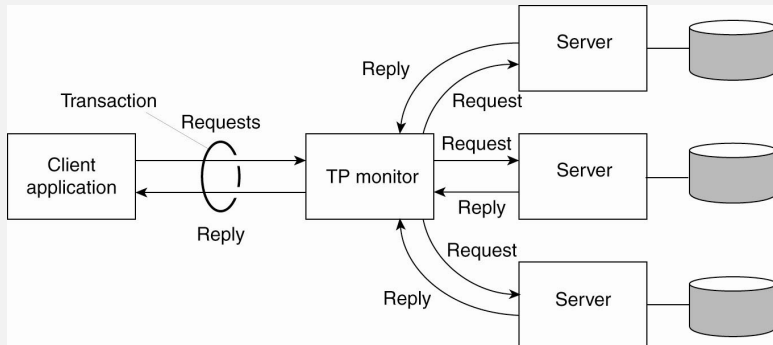
- ▶ `BEGIN_TRANSACTION`
- ▶ `END_TRANSACTION`
- ▶ `ABORT_TRANSACTION`
- ▶ `READ, WRITE`

**ACID** characteristic properties of transactions:

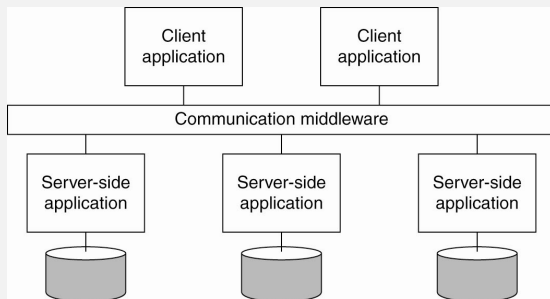
- ▶ **Atomic**: To the outside world, the transaction happens indivisibly
- ▶ **Consistent**: The transaction does not violate system invariants
- ▶ **Isolated**: Concurrent transactions do not interfere with each other
- ▶ **Durable**: Once a transaction commits, the changes are permanent

Transactions can be **nested**. Durability applies to top-level transactions only. For example: an airline and a hotel database. This led to the design of Transaction Processing Monitors (TP Monitors) to coordinate the commitment of multiple subtransactions.

## Example EAI: Transaction Processing Systems (2)



# Enterprise Application Integration



Middleware as a communication facilitator for enterprise application integration. This can be done in multiple ways. For example:

- ▶ **Remote Procedure Call (RPC)**: Requests are sent through local procedure call, packaged as message, processed, responded through message, and result returned as return from call.
- ▶ **Message Oriented Middleware (MOM)**: Messages are sent to logical contact point (published), and forwarded to subscribed applications.

# Distributed Pervasive Systems

Characteristics of pervasive systems (aka Internet Of Things):

- ▶ Blends into the environment
- ▶ Encourage ad hoc composition
- ▶ Naturally distributed
- ▶ Nodes are often small and battery powered
- ▶ Wireless/mobile communication is the norm

Examples: Ubiquitous Computing Systems, Mobile Computing Systems, Sensor (and actuator) networks

# In-class Exercise



Classify the distributed system examples we have seen so far into the three categories: Distributed Computing Systems, Distributed Information Systems, Distributed Pervasive Systems.

# Chapter 1: Recommended Exercises

- ▶ **Problem 1.** What is the role of middleware in a distributed system?
- ▶ **Problem 2.** Explain what is meant by transparency, and give examples of different types of transparency.
- ▶ **Problem 3.** Scalability can be achieved by applying different techniques. What are these techniques?
- ▶ **Problem 4.** Give further examples of distributed pervasive systems.