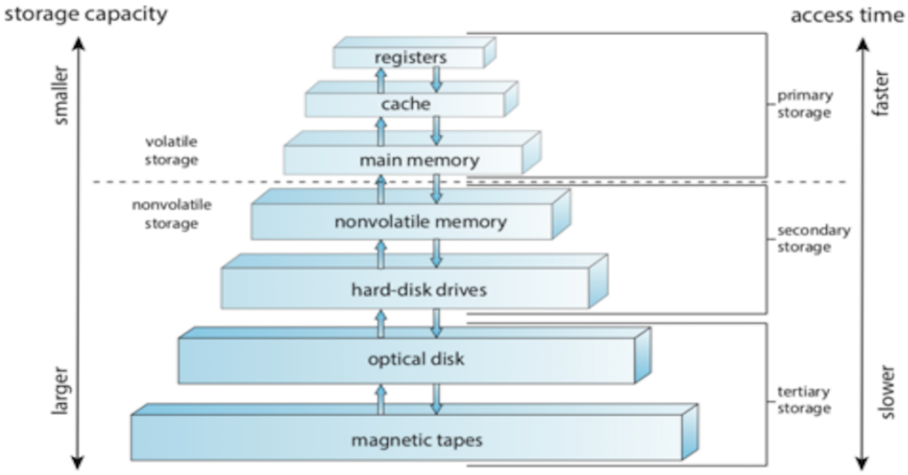
Chapter 1. Distributed Systems

Moor's Law

- {1965} => number of transistors per square inch on integrated circuits had doubled every year.
- {1975} => the space slowed down a bit, but data density had doubled approximately every 2/yr.

Hierarchy of successful Fast storage

- top is faster and smaller size.
- down is slower and large size.



Distributed System

Definitions:

- A collection of independent computers that appears to its users as a single coherent system.
- A collection of (perhaps) heterogeneous nodes connected by one or more interconnection networks which provides access to system-wide shared resources and services.
- A system in which hardware or software components located at networked computers communicate and coordinate their actions only by "message passing'.
- A system that consists of a collection of two or more independent computers which coordinate their processing through the exchange of synchronous or asynchronous message passing.
- A collection of autonomous computers linked by a network with software designed to produce an integrated computing facility.

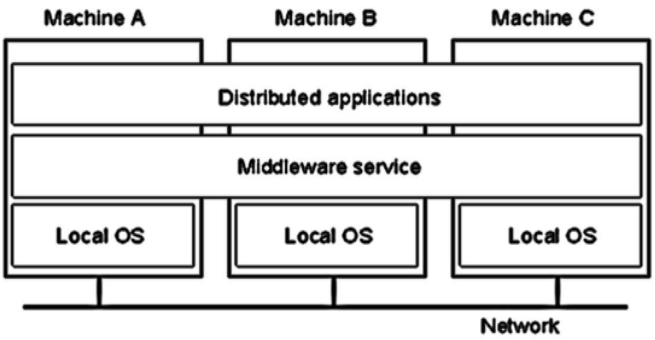
Construction of Distributed Systems:

- **Multiple Computers**
  - More than one physical computer, each consisting of CPUs, local memory, and possibly stable storage, and 1/O p a t h s to connect it with the environment.
- **Interconnections**
  - Mechanisms for communicating with other nodes via a network.
- **Shared State:**
  - a subset of nodes cooperates to provide a service, a shared state is maintained by these nodes. The shared state is distributed or replicated among the participants.

An Abstract View Distributed Systems

A distributed system organized a s middleware.

Note that the middleware layer extends over multiple machines.



Centralized and Decentralized Systems

Centralized Systems	Decentralized Systems
System shared by users all the time	Multiple autonomous components
All resources accessible	Some resources may not be accessible
Software runs in a single process	Components shared by users
Single physical location	Multiple physical locations
Single point of control	Multiple points of control
Single point of failure	Multiple points of failure
	No Global time & No Shared memory
	Software can run in concurrent processes on different processors
Example: Airplane booked, Banks	Example: Gird, Cloud

Distributed System Reasons

- **Functional distribution:**
  - Computers have different functional Capabilities (Client / Server, Host / Terminal).
- **Sharing of resources:** with specific functionalities (P2P).
- **Inherent distribution:**
  - stemming from the application domain, e.g.
    - Cash register and inventory systems for supermarket chains
    - Computer supported collaborative work
- **Load distribution / balancing:**
  - assign tasks to processors such that the overall system performance is optimized (HPC).
- **Replication of processing power:**
  - independent processors working on the same task
  - Distributed systems consisting of collections of microcomputers may have processing powers that no supercomputer will ever Achieve.
- **Physical Separation:** Systems that rely on the fact that computers are physically separated.
- **Economics:**
  - Collections of microprocessors offer a better price/performance ration than large mainframes.
    - mainframes: 10 times faster, 1000 times as expensive

Distributed System Applications:

1. Computing Dominated Problems (Distributed Computing): [HPC]
   - Mathematical Computations, Environmental and Biological model Modeling, Economic and Financial modeling, Graphics rendering, Network Simulations.
2. Storage Dominated Problems (Distributed Data):
   - Data Mining, Image Processing.
3. Communications Dominated Problems (Network Computing):
   - Transaction processing, Video on Demand, E-com, E-banking.

Common Distributed Computing Examples

- Network File System, Network Printer, ATM, Distributed databases, Network Computing, GPS, Retail point-of-sale terminals (POS), Air-traffic control, Enterprise Computing, Web.

Definitions:

- **Service:** manage a collection of related resources and present their functionalities to users and applications. the requesting process called *Client.*
- **Server:** a process on networked computer that accepts requests from Client processes on other computers to perform a service and responds appropriately.
- **Remote invocation:** A complete interaction between client and server, from the point when the *client* sends its request to when it receives the *server's* response.

Goals of Distributed System

1. **Resource Sharing:** Easy for users to access remote resources.
    - **Resource Types:** Hardware, Software and data.
2. **Transparency:** Hide that processes and resources are physically distributed across multiple computers.
    - Transparency has different aspects:

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 be moved to another location while in use
Replication	Hide that a resource may be redundant and shared by several competitive users
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

    - ***Access Transparency:*** Using identical operations to access local and remote resources, e.g. a Graphical User Interface (GUI) with folders.
    - ***Location Transparency:*** Resources to be accessed without knowledge of their location, e.g. URL (Support Availability).
    - ***Concurrency Transparency:*** Several processes operate concurrently using shared resources without interference with between them.
    - ***Replication Transparency:*** Multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.
    - ***Failure Transparency:*** Users and applications complete their tasks despite the failure of hardware and software components (Allow fail and recovery) (reliability) (e.g. email).
    - ***Mobility Transparency:*** Movement of resources and clients within a system without affecting the operation of users and programs (Migration and Relocation), e.g., mobile.
    - ***Performance Transparency:*** Allows the system to be reconfigured to improve performance as loads vary.
    - ***Scaling Transparency:*** Allows the system and applications to expand in scale without change to the system structure or the application algorithms

3. **Openness**:
    - is concerned with extensions and improvement of distributed systems.
    - Detailed interfaces of components need to be published.
    - New components have to be integrated with existing components.
    - It is determined by the degree to which new resource can be added and made available for using by a variety of client programs.
    - Differences in data representation of interface types on different processors (of different vendors) have to be resolved. (Heterogeneity)
4. **Scalability:** easy to expand and manage.
    - A system is described as scalable if will remain effective when there is a significant increase in the number of resources and the number of users.
    - The challenge is to build distributed systems that scale with the increase in the number of CPUs, users, and processes, larger databases, etc.
    - Scalability along several dimensions:
        - **Size Scalability:** Number of users and/or processes
        - **Geography Scalability:** Maximum distance between nodes.
        - **Administrative Scalability:** Number of administrative domains.

Distributed Computer System Metrics:

- **Latency:** network delay before any data is sent.
- **Bandwidth:** maximum channel capacity (analogue communication, digital communication).
- **Granularity:** relative size of units of processing required.
    - DS operate best with coarse grain granularity because of the slow communication compared to processing speed in general.
- **Processor speed:** MIPS, FLOPS.
- **Reliability:** ability to continue operating correctly for a given time.
- **Fault tolerance:** flexibility to partial system failure.
- **Security:** policy to deal with threats to the communication or processing of data in a system.
- **Administrative/management domains:** issues concerning the ownership and access to distributed systems components.
- **Performance:**
    - Response Time:
    - Throughput:
    - System utilization:
    - Network capacity utilization:

Distributed Programming Paradigms

- Client/server model
- Remote procedure calls
- Distributed File Systems
- Group communication and multicasts
- Distributed transactions
- Distributed shared memory
- Distributed object-based systems
- Publish-subscribe model
- Peer-to-peer model
- The Web

Distributed Systems Challenges

1. **Absence of a global clock:**

    - Due to *asynchronous message passing* there are limits on the precision with which processes in a distributed system can synchronize their clocks.
2. **Absence of a global state:**
    - In the general case, there is no single process in the distributed system that would havea knowledge of the current global state of the system
3. **Specific failure modes:**
    - Processes run autonomously, in isolation.
        - Failures of individual processes may remain undetected.
        - Individual processes may be unaware of failures in the system context.
4. **Heterogeneity:**
    - Distributed systems developed to many different kinds of software and hardware.
    - Heterogeneity at many levels:
        - **Network:** different kinds of software and hardware.
        - **Operating system:** different APIs to internet.
        - **Programming languages:** many different Programming Languages.
        - **Data:** different representations of data (Big Indian, Small Indian).
        - **Hardware:** Different clock cycles and memory capacity.
        - **Data Structures:** Implementations by different developer.

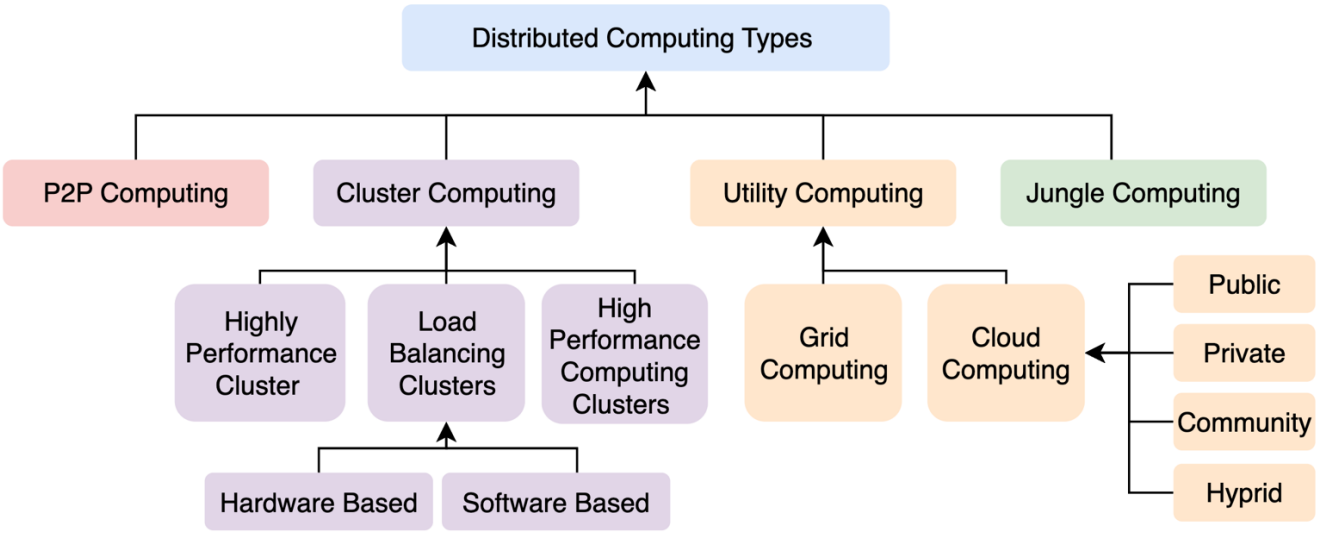
- **Heterogeneity (Solution):**
    - Middleware layer: (Common Object Request Broker Architecture (CORBA)): integrates many computing devices to act as a *coordinated computational* resource and *hide different topologies, communication networks and computing devices.*
    - Services that can be regarded as middleware:
        - **MOM:** Message Oriented Middleware.
        - **ORBs:** Object Request Brokers.
        - **ESB:** Enterprise Service Bus.
        - **Unifrom High Level API.**

5. **Openness:** Ensure extensibility and maintainability of systems.
6. **Security:** Privacy, Authentication, Availability, Trusting, Authorization
7. **Concurrency:**
    - Consistent scheduling of concurrent processes so that dependencies are preserved.
    - Allow several processes to operate concurrently using shared resources in a consistent fashion.
    - Avoidance of *dead lock* and *life lock* problems.
    -

Modeling a Distributed System

- **Asynchronous System**
    - No bound-on time to deliver a message. & No bound-on time to complete.
        - internet essentially asynchronous.
- **Synchronous System**
    - Known bound-on time to deliver a message. & Known bound-on time to complete.
        - LAN/cluster essentially synchronous.
- **Partially Synchronous System**
    - Initially system is asynchronous
    - Eventually the system becomes synchronous
        - Communication Protocols

Distributed Computing Types:



1. **Peer to Peer Computing:**
   - Large number of distributed resources connect by a network.
   - Every node acts as both a client and a server.
     - master-slave relationship exists among the peers.
   - no peer machine has a global information of the entire P2P system.
   - *No central coordination or no central database.*

P2P: is the sharing of computer resources and services by direct exchange between systems.
These resources and services include:
- Information Exchange, Processing Cycles, Cache Storage, & Disk Storage for Files.

What is driving P2P?
- Reduced The Load On Servers
- Inexpensive Computing Power
- Bandwidth and Storage

2. **Cluster Computing:**
   - Grouping multiple standalone computers in a cluster by a network.
   - Components of a cluster are connected to each other through fast local area networks.
   - Many types of computers can be refer to cluster computers, starting from a poor-man's supercomputer to *COWs (Clusters Of Workstations)*, and *NOWs (Networks Of Workstations)*

High Performance Computing (HPC) Cluster:
   - HPC clusters used where:
     - Time to solution a problem is important.
     - A problem is too big and can't fit on one single computer.
   - Ideal to run many similar jobs with different parameters or data sets (SPMD).
     - Hundreds of jobs submit and allow the cluster to manage workflow.
     - Depending on resources, all jobs run simultaneously, or some may wait in queue while other jobs finish.
     - is type of computing is local to a cluster node, the node doesn't communicate with other nodes, but may need high speed file system access *(Farm Computing Model).*

3. Utility Computing:

The design of Utility Computing based on a service providing (business) model

- when the users (consumers) need computing resources, they pay providers for using it.
- It's classified as: *Grid Computing* and *Cloud Computing*.

Grid Computing

- Using computers communicating over the *Internet* to work on a given problem or given application.
- Grid computing enables coordinated *resource sharing and problem solving in dynamic, multi-institutional virtual organizations.*
  - Using P2P as infrastructure of Grid.
- Enterprises or organizations present grids as integrated computing resources. They viewed as *virtual platforms* to support *virtual organizations.*
- The grids types: Knowledge, Data, Computational, Application Service Provisioning, Interaction or Utility.
- Most grid computing applications need middleware software to manage network resources.

Cloud Computing: Chapter 2.

Comparison	Grid Computing	Cloud Computing
Means of utilization	Allocation of multiple servers onto a single task or job	Virtualization of servers; one server to compute several tasks concurrently
Typical usage pattern	Typically used for job execution, i.e., the execution of a program for a limited time	More frequently used to support long-running services
Level of abstraction	Expose high level of detail	Provide higher-level abstractions

4. Jungle Computing

refers to the use of diverse, distributed and highly non-uniform performance computer systems to achieve peak performance.

- **Example:** Ibis high-performance distributed programming system.

- Combination of heterogeneous, hierarchical, and distributed computing resources.
- In many realistic scientific research areas, domain experts are being forced into concurrent use of multiple clusters, grids, clouds, desktop grids, independent computers.