DISTRIBUTED SYSTEMS

Introduction

A distributed system is a software system in which components located on networked computers communicate and coordinate their actions by passing messages. The components interact with each other in order to achieve a common goal.

Distributed systems Principles

A distributed system consists of a collection of autonomous computers, connected through a network and distribution middleware, which enables computers to coordinate their activities and to share the resources of the system, so that users perceive the system as a single, integrated computing facility.

Centralised System Characteristics

- One component with non-autonomous parts
- Component shared by users all the time
- All resources accessible
- Software runs in a single process
- Single Point of control
- Single Point of failure

Distributed System Characteristics

- Multiple autonomous components
- Components are not shared by all users
- Resources may not be accessible
- Software runs in concurrent processes on different processors
- Multiple Points of control
- Multiple Points of failure

Examples of distributed systems and applications of distributed computing include the following:

- telecommunication networks:
- telephone networks and cellular networks,
- computer networks such as the Internet,
- wireless sensor networks,
- routing algorithms;

- network applications:
  - World wide web and peer-to-peer networks,
  - massively multiplayer online games and virtual reality communities,
  - distributed databases and distributed database management systems,
  - network file systems,
  - distributed information processing systems such as banking systems and airline reservation systems;
- real-time process control:
  - aircraft control systems,
  - industrial control systems;
- parallel computation:
  - scientific computing, including cluster computing and grid computing and various volunteer computing projects (see the list of distributed computing projects),
  - distributed rendering in computer graphics.

Common Characteristics

Certain common characteristics can be used to assess distributed systems

- Resource Sharing
- Openness
- Concurrency
- Scalability
- Fault Tolerance
- Transparency

Resource Sharing

- Ability to use any hardware, software or data anywhere in the system.
- Resource manager controls access, provides naming scheme and controls concurrency.
- Resource sharing model (e.g. client/server or object-based) describing how
  - resources are provided,
  - they are used and
  - provider and user interact with each other.

Openness

- Openness is concerned with extensions and improvements of distributed systems.
- Detailed interfaces of components need to be published.
- New components have to be integrated with existing components.
- Differences in data representation of interface types on different processors (of different vendors) have to be resolved.

Concurrency

Components in distributed systems are executed in concurrent processes.

- Components access and update shared resources (e.g. variables, databases, device drivers).
- Integrity of the system may be violated if concurrent updates are not coordinated.
  - Lost updates

  - Inconsistent analysis

Scalability

- Adaption of distributed systems to
  - accomodate more users
  - respond faster (this is the hard one)
- Usually done by adding more and/or faster processors.
- Components should not need to be changed when scale of a system increases.
- Design components to be scalable

Fault Tolerance

Hardware, software and networks fail!

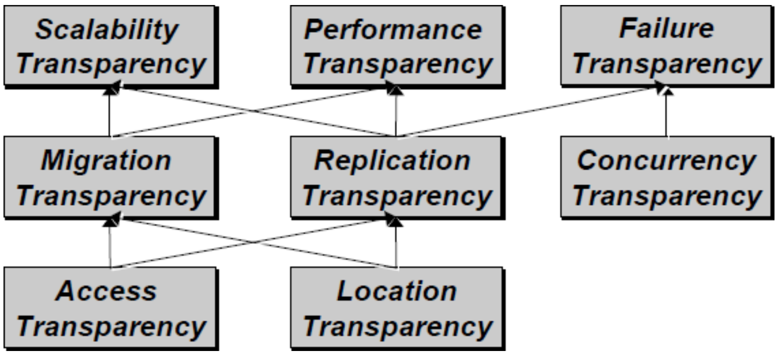
- Distributed systems must maintain availability even at low levels of hardware/software/network reliability.
- Fault tolerance is achieved by

  - recovery
  - redundancy

Transparency

Distributed systems should be perceived by users and application programmers as a whole rather than as a collection of cooperating components.

- Transparency has different dimensions that were identified by ANSA.
- These represent various properties that distributed systems should have.



Access Transparency

Enables local and remote information objects to be accessed using identical operations.

- Example: File system operations in NFS.
- Example: Navigation in the Web.
- Example: SQL Queries

Location Transparency

Enables information objects to be accessed without knowledge of their location.

- Example: File system operations in NFS
- Example: Pages in the Web
- Example: Tables in distributed databases

Concurrency Transparency

Enables several processes to operate concurrently using shared information objects without interference between them.

- Example: NFS
- Example: Automatic teller machine network
- Example: Database management system

Replication Transparency

Enables multiple instances of information objects to be used to increase reliability and performance without knowledge of the replicas by users or application programs

- Example: Distributed DBMS
- Example: Mirroring Web Pages.

Failure Transparency

- Enables the concealment of faults
- Allows users and applications to complete their tasks despite the failure of other components.
- Example: Database Management System

Migration Transparency

Allows the movement of information objects within a system without affecting the operations of users or application programs

- Example: NFS
- Example: Web Pages

Performance Transparency

Allows the system to be reconfigured to improve performance as loads vary.

- Example: Distributed make.

Scaling Transparency

Allows the system and applications to expand in scale without change to the system structure or the application algortithms.

- Example: World-Wide-Web
- Example: Distributed Database

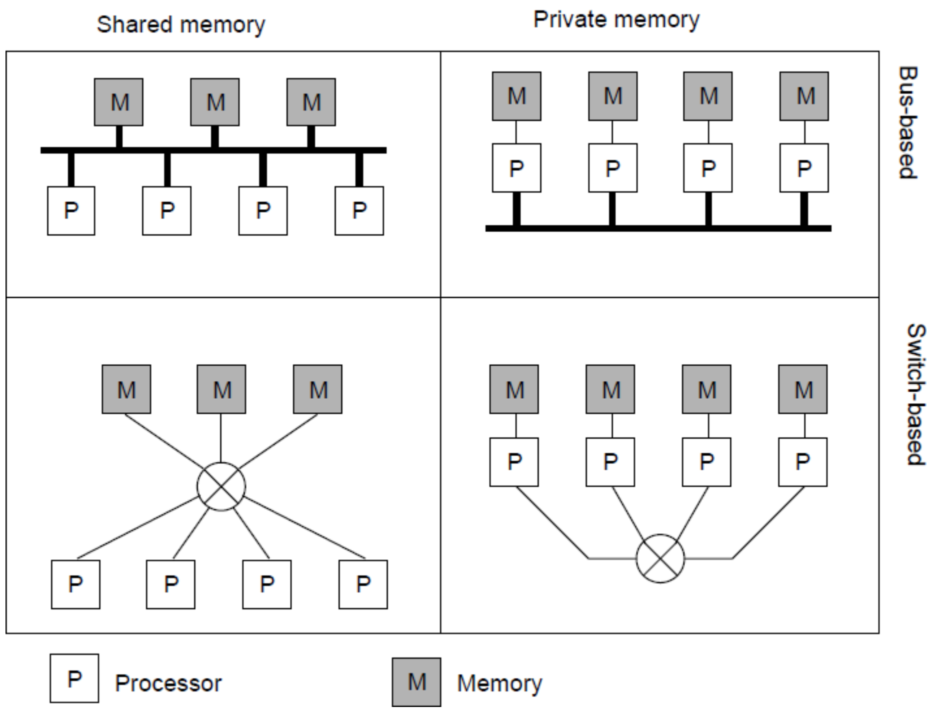
Distributed Systems: Hardware Concepts

- Multiprocessors
- Multicomputers

Networks of Computers

Multiprocessors and Multicomputers
Distinguishing features:

- Private versus shared memory
- Bus versus switched interconnection



Networks of Computers

High degree of node heterogeneity:

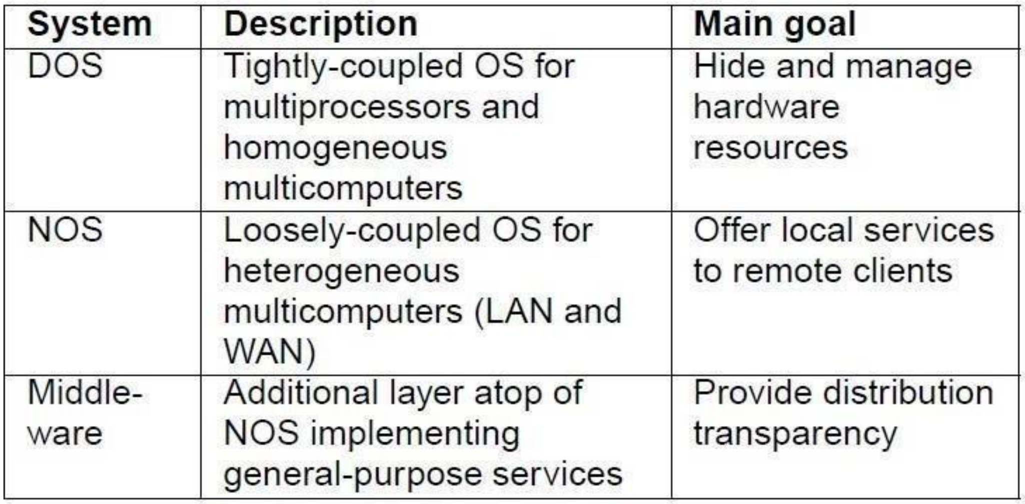
- High-performance parallel systems (multiprocessors as well as multicomputers)
- High-end PCs and workstations (servers)
- Simple network computers (offer users only network access)
- Mobile computers (palmtops, laptops)
- Multimedia workstations

High degree of network heterogeneity:

- Local-area gigabit networks
- Wireless connections
- Long-haul, high-latency connections
- Wide-area switched megabit connections

Distributed Systems: Software Concepts

Distributed operating system
_ Network operating system
_ Middleware

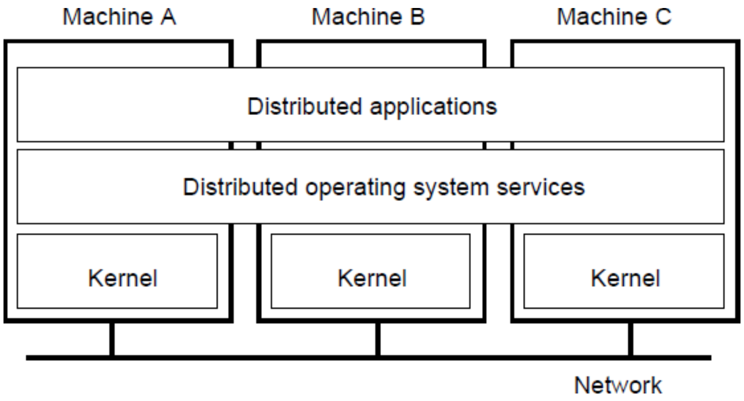


System	Description	Main goal
DOS	Tightly-coupled OS for multiprocessors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled OS for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middle-ware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

Distributed Operating System

Some characteristics:

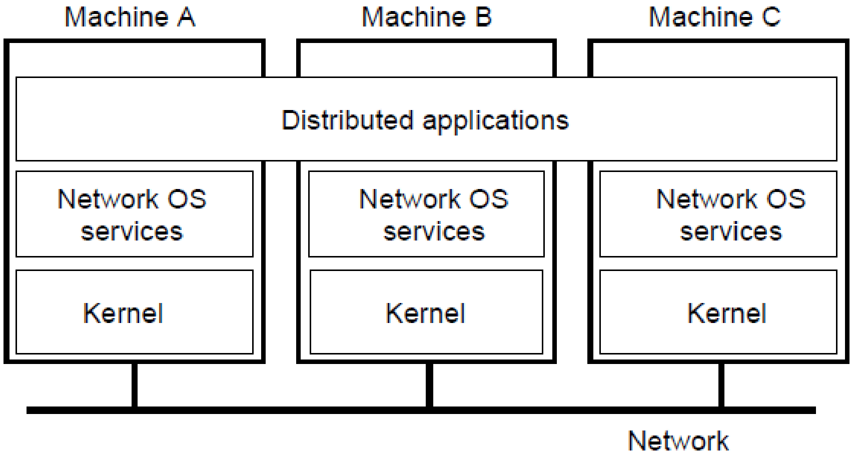
_ OS on each computer knows about the other computers
_ OS on different computers generally the same
_ Services are generally (transparently) distributed across computers



Network Operating System

Some characteristics:

_ Each computer has its own operating system with networking facilities
_ Computers work independently (i.e., they may even have different operating systems)
_ Services are tied to individual nodes (ftp, telnet, WWW)
_ Highly file oriented (basically, processors share *only* files)

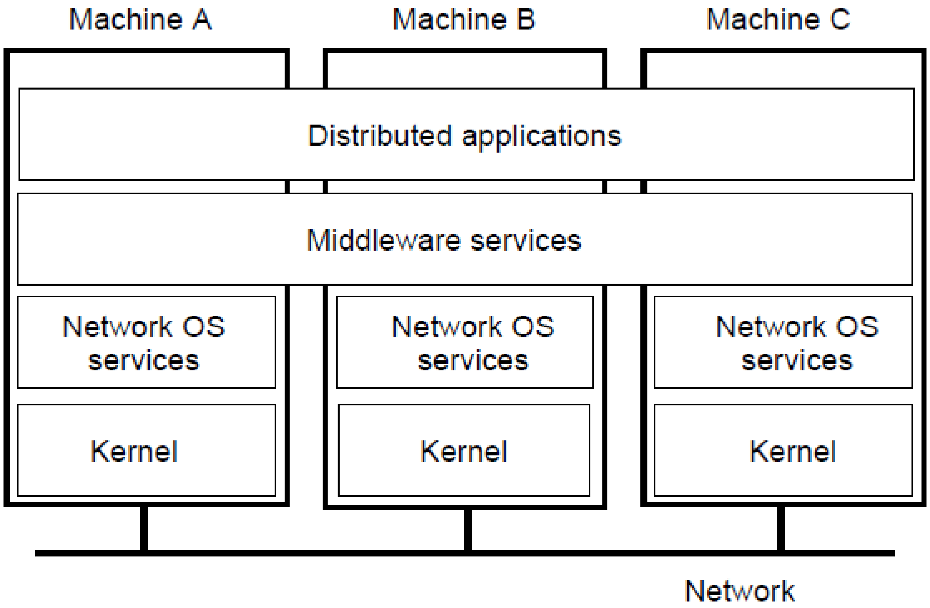


Distributed System (Middleware)

Some characteristics:

_ OS on each computer need not know about the other computers
_ OS on different computers need not generally be the same
_ Services are generally (transparently) distributed across computers

-



Need for Middleware

Motivation: Too many networked applications were hard or difficult to integrate:

_ Departments are running different NOSs
_ Integration and interoperability only at level of primitive NOS services
_ Need for federated information systems:
– Combining different databases, but providing a single view to applications
– Setting up enterprise-wide Internet services, making use of existing information systems
– Allow transactions across different databases
– Allow extensibility for future services (e.g., mobility, teleworking, collaborative applications)
_ Constraint: use the existing operating systems, and treat them as the underlying environment (they provided the basic functionality anyway)

Communication services: Abandon primitive socket based message passing in favor of:

_ Procedure calls across networks
_ Remote-object method invocation
_ Message-queuing systems
_ Advanced communication streams
_ Event notification service

Information system services: Services that help manage data in a distributed system:

_ Large-scale, system wide naming services
_ Advanced directory services (search engines)
_ Location services for tracking mobile objects
_ Persistent storage facilities
_ Data caching and replication

Control services: Services giving applications control over when, where, and how they access

_ Distributed transaction processing
_ Code migration

Security services: Services for secure processing and communication:

_ Authentication and authorization services
_ Simple encryption services
_ Auditing service