System Models

Types of System Models

Architectural Models

It describes a system in terms of the computational and communication tasks performed by its computational elements; the computational elements being individual computers or aggregates of them supported by appropriate network interconnections.

Fundamental Models

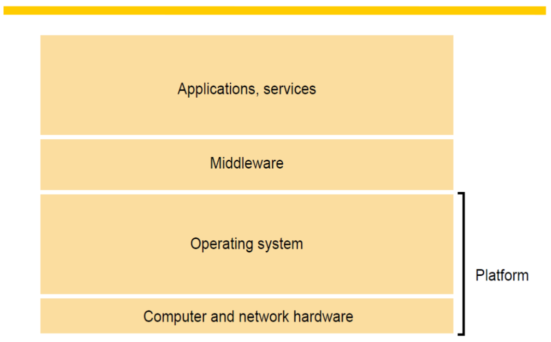
It takes an abstract perspective, in order to examine individual aspects of a distributed system. It is a model that contains only the essential ingredients that we need to consider in order to understand and reason about some aspects of a system's behavior.

Three important aspects of Fundamental Models:

- **Interaction Models:** It considers the structure and sequencing of the communication between the elements of the system.
- **Failure Models:** It considers the ways in which a system may fail to operate correctly.
- **Security Models:** It considers how the system is protected against attempts to interfere with its correct operation or to steal its data.

Q3. Types of hardware and software resources that can be shared in a distributed system

Figure 2.7
Software and hardware service layers in distributed systems



Hardware

CPU: Most servers, such as filer servers, do some computation for their clients, hence their CPU is a shared resource.

Memory: Cache server (holds recently-accessed web pages in its RAM, for faster access by other local computers).

Disk: File server, virtual disk server, video on demand server, etc

Printer: Networked printers accept print jobs from many computers. managing them with a queuing system.

Network Capacity: Packet transmission enables many simultaneous communication channels (streams of data) to be transmitted on the same circuits.

Graphics card: As machine learning models require heavy computation hardware and graphics cards are a costly piece of hardware, making it a shared resource will make it cost-effective.

Software

Web Page: Web servers enable multiple clients to share read-only page content (usually stored in a file, but sometimes generated on the fly).

File: File servers enable multiple clients to have read-write access to the same file.

Object: Possibilities for software objects are limitless. E.g. shared whiteboard, room booking system, etc.

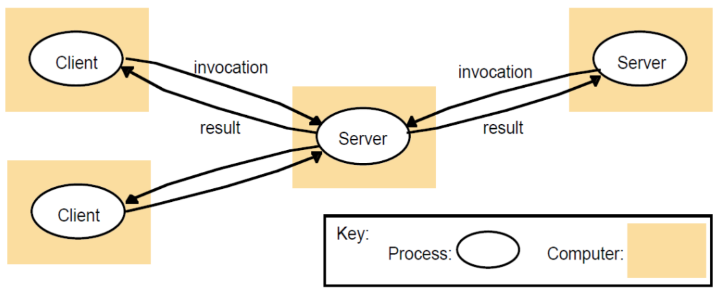
Database: Databases are used to store useful data about and for the users. Databases can be accessed and used concurrently in a distributed system.

Video / Audio Stream: Servers can store entire videos on disk and deliver them at playback speed to multiple clients simultaneously.

Exclusive Lock: A system-level object provided by a lock server, enabling several clients to coordinate their use of a resource (such as a printer that does not include a queuing scheme).

Client-Server Model

Figure 2.3
Clients invoke individual servers



The system is structured as a set of processes, called servers, that offer services to the users, called clients.

- The client-server model is usually based on a simple request/reply protocol, implemented with send/receive primitives or using remote procedure calls (RPC) or remote method invocation (RMI)
- The client sends a request (invocation) message to the server asking for some service
- The server does the work and returns a result (e.g. the data requested) or an error code if the work could not be performed.
- A server can itself request services from other servers; thus, in this new relation, the server itself acts as a client.
- For example: Search engines, which enable users to look up summaries of information available on web pages at sites throughout the Internet.
- These summaries are made by programs called web crawlers, which run in the background at a search engine site using HTTP requests to access web servers throughout the Internet.
- Thus a search engine is both a server and a client: it responds to queries from the browser clients and it runs web crawlers that act as clients of other web servers.

Q4. Discuss the Software Layers of distributed system architectural model.

- Software architecture refers to services offered and requested between processes located in the same or different computers.

- It is the structure of software as layers or modules in a single computer.
  - **Layer:** A group of related functional components.
  - **Service:** Functionality provided to the next layer.
- **Platform:** The lowest-level hardware and software layers are often referred to as a platform for distributed systems and applications.
  - These low-level layers provide services to the layers above them, which are implemented independently in each computer.
  - These low-level layers bring the system’s programming interface up to a level that facilitates communication and coordination between processes.
  - Ex: Intel x86/Windows, Intel x86/Linux, etc.
- **Middleware:** It is a layer of software whose purpose is:
  - To provide a convenient programming model to application developers.
  - To mask heterogeneity presented in distributed systems and provide interoperability between the lower layer and upper layer.

Q5. What are variations of the client-server model?

There are four main categories of client-server computing:

- **One-Tier architecture**: It consists of a simple program running on a single computer without requiring access to the network. User requests don’t manage any network protocols, therefore the code is simple and the network is relieved of the extra traffic.
- **Two-Tier architecture**: It consists of the client, the server, and the protocol that links the two tiers. The GUI code resides on the client host and the business logic resides on the server host.
- **Three-Tier architecture**: It consists of a presentation tier, which is the User Interface layer, the application tier, which is the service layer that performs detailed processing, and the data tier, which consists of a database server that stores information.
- **N-Tier architecture:** It divides an application into logical layers, which separate responsibilities and manage dependencies, and physical tiers, which run on separate machines, improve scalability and add latency from the additional network communication.

Q6. Describe the Interaction Model of a distributed system

Interaction models are for handling time i. e. for process execution, message delivery, clock drifts, etc.

There are two variants of the Interaction Model

Synchronous Distributed Systems:

- The time taken to execute each step of a process has known lower and upper bounds.
- Each message transmitted over a channel is received within a known bounded time.
- Each process has a local clock whose drift rate from real-time has a known bound.

Asynchronous Distributed Systems:

- The time taken to execute each step of a process can be arbitrarily long.
- Each message transmitted over a channel may be received after an arbitrarily long time.
- Each process has a local clock whose drift rate from real-time may be arbitrarily long.

Q7. Write the design requirements for Distributed architectures.

i) Performance Issues

Performance issues arising from the limited processing and communication capacities of computers and networks are considered under the following subheading:

- **Responsiveness:** Time the system takes to process a request. How quickly the system acknowledges a request.
- **Throughput:** It is the rate at which computational work gets done.
- **Balancing of Computational Loads:** Using several computers to host a single service. Ex: Using applets on clients, to remove the load on the server.

ii) Quality of service

The ability of systems to meet deadlines. It depends on the availability of the necessary computing and network resources at the appropriate time.
The main properties of the quality of the service are:

- Reliability
- Security
- Adaptability
- Performance

iii) Use of caching and replication

Distributed systems overcome the performance issues by the use of data replication and caching. Cached copies of resources should be kept up-to-date when the resource at a server is updated.

iv) Dependability Issues

Dependable applications should continue to function correctly in the presence of faults in hardware, software, and networks.

v) Fault Tolerance

Dependable applications should continue to function in the presence of faults in hardware, software, and networks.

Q8. Describe the failure model of the distributed system

- Failures can occur both in processes and communication channels. The reason can be both software and hardware faults.
- Fault models are needed in order to build systems with predictable behavior in case of faults (systems that are fault-tolerant).
- Such a system will function according to the predictions, only as long as the real faults behave as defined by the “fault model”.

Variants of the Failure Model

Omission Failures:

- A process or channel fails to perform the actions that it is supposed to do.

- Process Omission Failure:
    - A process completes a send-operation, but the message is not put into the outgoing message buffer.
    - A message is put into a process's incoming message buffer, but the process does not receive it.
- Communication Omission Failure:
    - It is a 'fail-stop' failure, which means the server only exhibits crash failures, but at the same time, any correct server in the system can detect that this particular server has failed.

Arbitrary Failures:

- The term arbitrary or Byzantine failure is used to describe the worst possible failure semantics, in which any type of error may occur.
- An arbitrary failure of a process is one in which it arbitrarily omits intended processing steps or takes unintended processing steps.
- Process or channel may exhibit arbitrary behaviour when failing:
    - To send/receive arbitrary messages at arbitrary intervals.
    - A process may halt or perform "faulty" steps.

Timing Failures:

- Timing failures are applicable in synchronous distributed systems where time limits are set on process execution time, message delivery time, and clock drift rate.
- Timing Failures can be classified as:

Class of Failure	Affects	Description
Clock	Process	Process’s local clock exceeds the bounds on its rate of drift from real-time.
Performance	Process	Process exceeds the bounds on the interval between two processing steps.
Performance	Channel	A message’s transmission takes longer than the stated bounds.

Masking Failures:

- Knowledge of the failure characteristics of a component can enable a new service to be designed to mask the failure of the components on which it depends.
- A service masks a failure either by hiding it altogether or by converting it into a more acceptable type of failure.
- Example: Multiple servers that hold replicas of data can continue to provide a service when one of them crashes.