Distributed Systems

- A set of computers (hosts or nodes) connected through a communication network.
- Nodes may have different speeds and computing capabilities.
- Some nodes may be mobile in the network; i.e. their addresses may change frequently.
- Nodes may be under the control of different organizations or individuals.
- State of the system cannot be maintained at one central location because of size and network latency.
- Nodes may be resource constrained: small memory, power limitation (PDAs, sensor nodes).
- Each node provides some basic resource management functions to support facilities for implementing a middleware for building distributed applications.
Characteristics of Distributed Systems

• Not possible to have the most up-to-date view of the system state:
  – Distributed nature of the state, and latency in communication
• A centralized control and administration may not be always feasible due to size and scale of a system.
  – For example the Domain Name System
• Some nodes may be connected to the system intermittently and may become available at times due to failures or network partition.

Characteristics of Distributed Systems

• In a large-scale system such as a datacenter, component failures cannot be avoided.
  – Failures are normal and common
• Systems have to be designed to cope with component failures using suitable recovery mechanisms.
Goals in Distributed System Designs

• Scalability
  – Performance of the system does not degrade significantly as the system size increases

• Transparency
  – There are several aspects of transparency that a distributed system may strive to support

• High Availability and Reliability

Different aspects of transparency

• Network/Location Transparency
  – Local and remote resources are access using the same protocol – users are not aware of network location of the resource
    • System level notion
  – Many of today’s application require location-aware access

• Failure Transparency
  – Mask failures of system components through redundancy.

• Replication Transparency
  – Hide the fact that a service or resource is replicated at multiple nodes for increased reliability and performance.

• Transparency of concurrent access of services by other applications
Models of Synchrony

• Synchronous Model:
  – Message sent by a process is delivered to the destination process within a known bound.
    • “Certain communication”
  – Bounds can be established on the execution time for a task at a node.
  – Process failures are in the form of crashes.
    • Crash-failure semantics

Models of Synchrony

• Asynchronous Model:
  – No bounds exist on communication delays.
  – A task may take arbitrary amount of time at a node.
    – It is a time-free model.
  – In such a system it is not possible to differentiate between a failed node and an arbitrarily slow node.
Models of Synchrony

  - It is a model of asynchronous systems with a notion of time bounds on a service request-reply.
  - All services (processes) perform their operations within in specified time bounds and a response is delivered to the client within a specified time bound.
  - Communication can be through unreliable channel that may omit to deliver a message. Omission failures
  - Crash-failure semantics for processes.

Fundamental Limitations

**CAP Theorem** by Eric Brewer

No system can support all of the following three properties simultaneously:
1. Consistency
2. Availability
3. Partition-tolerance
Paradigms for System Structuring and Organization

- Request-Reply Paradigm
  - Client-Server Model
  - Remote Procedure Call Model
- Group Communication Models
- Peer-to-Peer Computing Models
- Grid Computing Model

Request-Reply Model

**Client**
- Send request
- Receive response
- Client is blocked

**Server**
- Receive request
  - Process request, execute the requested operation
  - Send response

Remote Procedure Call (RPC) Model is based on client-server model
Client Server Model

• It is based on the request-reply model of interaction between two processes.
• Client sends a request message to the server process.
• The request message contains the desired service operation.
• Server performs the requested operation, if it is valid request from an authorized user. It then sends a response message to the client.
• **Synchronous Communication** – client is blocked until the response message is received.

Client Server Model

• A server is managed by some administrative authority or organization.
  – Clients may have some degree of trust in the server through mechanisms such as reputation or certification.
• **Stateless services** vs **Stateful services**
• Stateless servers have the advantage in quickly recovering from a crash as no client-specific data is lost.
  – However, each interaction has to provide full context information based on prior interactions.
Failure Cases

- Client may crash and restart, and then send the request again.
  - Server may execute the operation twice

- Server may crash
  - Before receiving/processing a request, or
  - After receiving/processing request but before sending a reply to the client

Reliability Semantics for Remote Procedure Calls

There are several possible reliability semantics that an RPC design.

- **MAY-BE:**
  - Zero, one, or more execution of the procedure by the server

- **AT LEAST ONCE:**
  - If there are no permanent crash of the server, then the operation would be performed at least once, and one or more responses may be returned to the client.

- **AT MOST ONCE:**
  - If the server does not crash permanently, then the operation would be performed exact once, and one response message would be delivered to the client.

- **EXACTLY ONCE:**
  - Server executes the operation exactly once.
RPC Reliability: “at most once” execution semantics

- A unique sequence number can be attached to each request message by the client.
- Server needs to maintain for each client the unique sequence numbers of all the requests from that client that it has processed.
  - This makes the server stateful.
  - How long should a server keep the record of the sequence number of past requests processed by it?
    - Client may acknowledge responses.
  - If the goal is to achieve “AT MOST ONCE” semantics then server also need to keep a copy of the response message, which can then be sent to the client if the duplicate copy of a request is received.

Idempotent Operation

- An idempotent operation has the property that any number of repeated executions of it has the same effect as a single execution.
  - If a request is for an idempotent operations then ensuring “AT LEAST ONCE” semantics is sufficient.
Group Communication

Different semantics for ordering the delivery of messages to a group are possible:

Delivery Semantics

• Is a message delivered to all of the group members that have not failed?

• Are all messages delivered to the group members in the same order?
  – Process A receives “green” message before the “red” message, but process Z receives them in the reverse order.

• Do all correct processes receive exactly the same sequence of message?
Group Communication Models

- **Total Order Broadcast**
  - All messages are delivered in the same order to every process
  - Also called Atomic Broadcast

- **FIFO Broadcast**
  - Messages from the same sender are delivered in the same order to every process

- **Causal Broadcast**
  - Causal relationships among messages are preserved in delivery

Peer-to-Peer Computing

- In the client-server model, the servers are controlled and managed by well-known authorities, organizations, and network locations.
  - Server usually offers some degree of reliability, availability, and security.
  - Assymetry

- In P2P computing, anyone can run a server or a client process.
  - Such server processes may be highly transient.
  - No guarantee of security or reliability.
  - No centralized control
  - It may not be possible to determine the exact number of nodes and their configuration in a P2P system due to their transient nature and decentralized control.
  - Popularized by Gnutella and Napster for sharing music files.
Peer-to-Peer Computing

Two broad categories of approaches for structuring such systems:

1. Unstructured P2P systems, where a node can connect with any other nodes, and there is no organizational relationships among nodes.
   - Gnutella, Napster

2. Structured P2P Systems: There is organizational relationship among nodes, which allows to design search and communication algorithms with known performance characteristics:
   - Distributed Hash Table Schemes
   - Examples: Chord, Pastry (P2P storage systems)

Reliability and Availability

- Availability indicates the probability of a service being accessible and available to a client.
  - Availability of a service may be different for clients at different network locations.
  - A service may become unavailable for several reasons:
    - Server crash/failure
    - Network partition
Availability

- Mean-time-between-failures (MTTB)
- Mean-time-to-repair (MTTR)
- Mean-time-to-failure (MTTF) also called Mean-Up-Time (MUT)

Availability = MTTF / MTTB

Reliability

- It is a measure of a system’s ability to satisfy its functional as well as non-functional specifications.
  - Executes its functions correctly
  - It satisfies performance specifications,
    - It executes its functions within the specified time bounds
- A system accepts a service request when it is available, but it may not be able to execute the requested operation correctly or in a timely fashion if it encounters a failure during the request processing.
Approaches to Scaling Systems

**Scaling:** Support larger workload, client load, and provide higher throughput

**Vertical Scaling:**
Deploy more power computers to scale a system. This has fundamental limitations as well budgetary limitations.

**Horizontal Scaling:**
Use a larger pool of commodity resources to deploy the application and use suitable partitioning of state

---

General Techniques for Building Scalable Systems

Some of the general techniques used for performance, scalability, reliability, and fault-tolerance:

- Hierarchy
- Partitioning (sharding)
- Replication and Caching
- Use of Hints
- Soft-state
- Weaker models of Consistency
Hierarchy and Partitioning

- Key to scalability is to distribute processing and management functions through:
  - Decentralized control by partitioning and delegating responsibilities to different entities in the network.
  - Hierarchical management and delegation of authorities:
    - An entity may further delegate responsibilities to other entities under its control.
- Domain Name System is one of the best examples.

Replication

- Replication of resources is one of the key approaches for avoiding any central point of failure in the system:
  - Replication of data
  - Replication of server processes implementing a service
- Replication is an important technique for improving reliability, availability, and performance in a system.
  - Place copies of data closer the clients at different network locations to reduce network latencies
  - Distribute client request load to different servers implementing a service.
Replication Management Issues

How to keep multiple replicas mutually consistent?
• This becomes the central issue in replication management.
• This issue arises with the replicated data when data is to be updated.
  – Primary copy approach
  – Quorum based approach
• In process replication, all server processes need to be kept mutually consistent:
  – Primary copy approach
  – Implement replicated server processes as a group, and deliver all request messages to each group member in the same order. "Virtual Synchrony"

Caching

• Another form of replication, but caching decisions are usually made at the client node.
  – Client keeps a local copy after using some data.
  – This avoids fetching data again if it has not been modified since its last use.
• Replication policies and decisions are usually made as part of the management policies for a service.
  – Replicas tend be far fewer than cached copies.
  – Each replica is controlled and managed by some trusted servers.
Hints

- Like a cached data, it is maintained by a client.
- Unlike a stale cached data, use of any stale “hint” does not affect the correctness of any operation.
- A good or correct hint helps in improving performance.

Soft State

- Soft state of system pertains to that part of the system state information which can be reconstructed when a node restarts after a crash.
  - This can be done by querying the state of the other nodes.
  - No need to store this kind of soft state in the secondary storage.

- See work by Armando Fox.
Atomic Actions and Transactions

- A transaction is a sequence of operations on some data.
- It satisfies the ACID property, which is key to fault-tolerance:
  - **Atomicity**
    - All or nothing property
      - *All* when transaction commits.
      - *None* when the transaction aborts.
  - **Consistency**
    - A committed transaction takes the system from one consistent state to another.
  - **Isolation**
    - Any intermediate states in the execution sequence are not visible to other processes.
  - **Durability**
    - Persistence of the effects of a committed transaction

Weak Consistency Models

- **ACID transactions** provide strong consistency guarantees and a simple and clear abstraction for structuring operations that change the state of a system.
- This costs performance and scalability due to distributed coordination in large-scale systems.
- Weaker models of consistency, such as eventual and causal consistency, do not provide strong guarantees of the system state consistency but can still be useful in many applications.
- **BASE transactions**
Agreement and Consensus in Distributed Systems

- This is an important notion in distributed system architectures, particularly in building reliable systems
- Agreement and consensus protocols ensure that members in a distributed group have a consistent view of the distributed system state.
- Examples:
  - Current membership in the group
  - A node has failed
  - Result of a computation task
  - Order of messages sent to the group