Distributed Computing on Distributed Systems

Distributed Systems

A **distributed system** consists of:

- Multiple *autonomous processing nodes* (compute nodes, computers)
- Network connectivity between them
- Software for coordinating computing activities across autonomous processing nodes

Goal: be perceived as single computing facility by users.

Characteristics of Distributed Systems. Distributed systems have the following characteristics:

- Multiple components working autonomously from each other.
- Software runs on different processors, and on different nodes.
- Software runs concurrently on multiple different nodes.
- Software runs asynchronously on multiple different nodes.
- System has multiple points of control.
- System has multiple points of failure.

Distributed System Considerations. When developing distributed systems system designers have to pay attention to the following.

- Architecture of control points.

- Distribution of tasks and load balancing.
- Resource sharing between compute nodes.
- The CAP theorem.
- Consistency of data.
- Synchronization.
- Unreachability of resources.
- Communication between nodes.

Distributed System Benefits. Why perform build distributed systems? Because they help facilitate

- Resource sharing
- Concurrency of execution
- Scalability
- Fault Tolerance

Resource Sharing. As a **benefit**:

- Distributed Systems can be deployed on commodity hardware.
- Distributed Systems can operate with larger quantities of data.
- Distributed Systems allow for concurrent work on multiple tasks.

As a **consideration**:

- Type of sharing architecture:
  - Shared Memory (very rare)
  - Shared Disk (MapReduce)
  - Shared Nothing (distributed DBMS, DNS)
- Resource distribution among compute nodes:
  - Sharding of data.
  - Replication of data.

Concurrency. This is the core **benefit** of a distributed system:

> Distributed Systems execute their processing tasks as concurrent processes on separate compute nodes.

Concurrency of execution is subject to resource sharing policies.

Concurrency of exectution leads to asynchronicity, which, in turn, may lead to Inconsistency of data.

Scalability. Distribution of compute power (and as a side effect — of data storage) leads to:

- Increased sizes of input that can be processed.
- Increased throughput: individual tasks are completed faster.
- Increased bandwidth: more tasks can be operated on at the same time.

Fault Tolerance. Distributed Systems are our first line of defense against hardware and software failure:

- Hardware, software, network are all **prone to failure**.
- Distributed systems add points of failure.
- But they also decrease the criticality of each point of failure.
- This makes well-designed Distributed Systems *fault tolerant*: failure of individual components does not lead to failure of the overall system.
- Fault Tolerance is facilitated by:
  - Redundancy (of data, of components, of compute nodes)
  - Recoverability

Architecture of control points. There are different architectures of Distributed Systems based on how control over distributed system operations is structured and exerted.

- **Centralized control** systems have a dedicated compute node responsible for tasking other compute nodes.
- **Distributed control** systems have multiple equal-authority dedicated compute nodes dedicated to tasking other compute nodes.
- **Decentralized** systems do not have specific compute nodes control distribution of work. All compute nodes in such systems run in exactly the same way.

Task Distribution and Load Balancing. The ability of a Distributed System to spread the computational tasks evenly among individual compute nodes is an important issue, as it affects the effectiveness of Distributed Systems and facilitates Scaleability.

- Load balancing is the computational process within a distributed system that determines which nodes receive which specific computational tasks. It is designed to work in a way that tasks each available compute node with roughly equal amounts of computations to complete.

The CAP Theorem. There are three crucial properties a Distributed System must aspire to:

Consistency: Every request for data receives the most recent version of it (or error).

Availability: Every request receives a (non-error) response.

Partition-tolerance: The system continues to operate despite an arbitrary number of messages between compute nodes being dropped/delayed by the network.

The **CAP** theorem (in its most up-to-date form) states this:

> In Partition-tolerant systems either Availability or Consistency will not be achieved.

Note: Earlier version of the **CAP** theorem was "Consitency. Availability. Partition-tolerance. Pick two." However, since the notion of the **CAP theorem** has been introduced into the field of Distributed Systems, researchers reached an agreement that a Distributed System *must be Partition Tolerant*. This leaves us with one of the other two properties to pick as *achievable* in a specific Distributed System.

The **CAP theorem** gives rise to two core types of distributed systems.

Consistent Partition-Tolerant Systems (CP Systems). These Distributed Systems concentrate on achieving the property of Consistency by ensuring that no response to user request is given until the system validates the consistency/correctness of the response message.

Example. Bank ATM networks are an example of a CP System. ATM networks must ensure proper handling of bank accounts of their users on each transaction. Therefore, they cannot afford inconsistent transactions (e.g., dispensing money before certifying that a customer has enough on their account, or accepting money before confirming that the money can be deposited to the right account). MapReduce frameworks (e.g., Hadoop, Spark) are CP Systems too: they distribute computation of specific data processing tasks. Those computations are not allowed to return incorrect results (but may fail to proceed, if correct computation cannot be completed).

Available Partition-Tolerant Systems (AP Systems.) These Distributed Systems concentrate on achieving the Availability property by ensuring that each request receives the *best guess* of the system about what the right response message should be.

Example. DNS networks are an example of an AP System. Each DNS node has a local database that allows it to resolve a specific domain name to an IP address. If a change has been made to one of the DNS servers, that has not been propagated yet, a second DNS server will NOT attempt to determine if its own mapping of a domain name to an IP address is stale. It will simply return its current mapping, even if it is no longer the correct one. NoSQL DBMS delivering data for Facebook to put on your wall work as an AP System.

Consistency of Data. From the above it is clear that Data Consistency plays a **major role** in development of distributed systems.

- CP Systems implement Consistency, i.e., the guarantee no response to a user request returns stale data.
- AP Systems implement Eventual Consistency, i.e., the guarantee that for every change of the data, the new value will *eventually* become the one being returned by the system[1].

Consistency verification protocols are complicated by...

Synchronization. Distributed Systems run as a collection of independent computing processes on multiple hardware compute nodes. Multiple processes running on multiple nodes must run *asynchronously.* However, the actual tasks these processes are trying perform may require synchronization between the outcomes.

Resource unreachability. See our discussion of Fault tolerance and Partition tolerance. Distributed Systems must

- Be able to detect when certain computational or strorage resources are no longer reachable by the nodes in the system.
- Be able to restart any processes that have been executed on the no longer reachable compute nodes.

[1] Unless superceded by a newer change.

- Be able to re-assign important roles (process control, load balancing, etc) to a new compute node if other compute nodes responsible for these decisions are no longer reachable.

Communication between nodes. At a lower level, unless there is communication between compute nodes in a Distributed System, there really is no Distributed System. Distributed Systems must:

- Have proper communiction protocols in place for individual nodes to pass information to each other. These depend on the overall architecuture of the system.
- Have clear rules for data transfer between compute nodes (if needed).
- Have clear rules for determining when communications between nodes fail, i.e., when compute nodes become unavailable.