Lecture Notes: Introduction to Distributed Systems

Introduction to Distributed Systems

1.1 Distributed Systems

A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system.

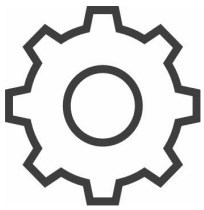
Distributed systems are used to solve problems that cannot be solved by a single system.

These problems are broadly categorised into the following categories:

- Computation
- Storage

Computation

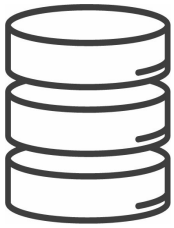
This seems like a trivial problem but becomes complex when the volume and velocity of data increases to an extent that it cannot be efficiently handled by a single machine. Typical use cases include Machine Learning and Big Data problems.



Computation

Storage

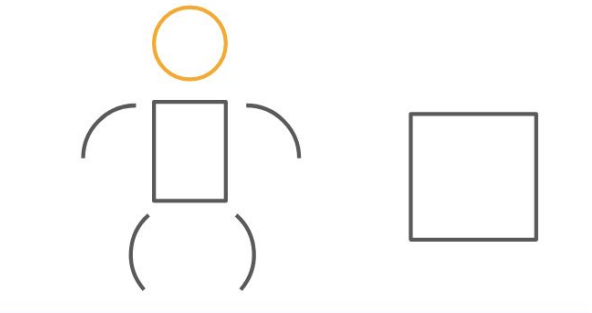
The storage in a single machine can only be expanded to a certain point. After that the cost of implementing a larger storage is so high that it becomes impractical.



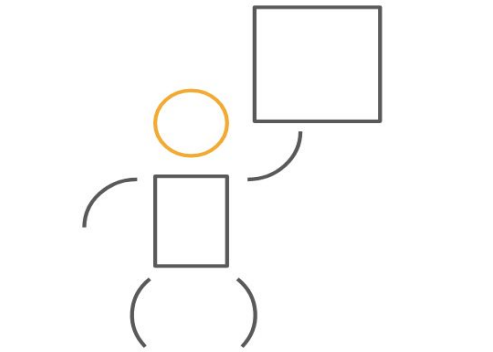
Storage

Distributed Systems Example

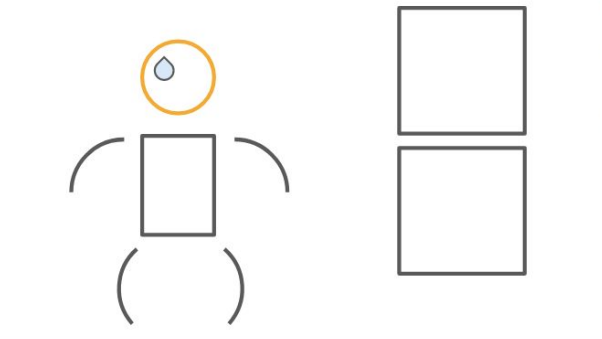
Let’s understand the use of a distributed system through the following example. You know we use distributed systems when the problem is too large to handle.



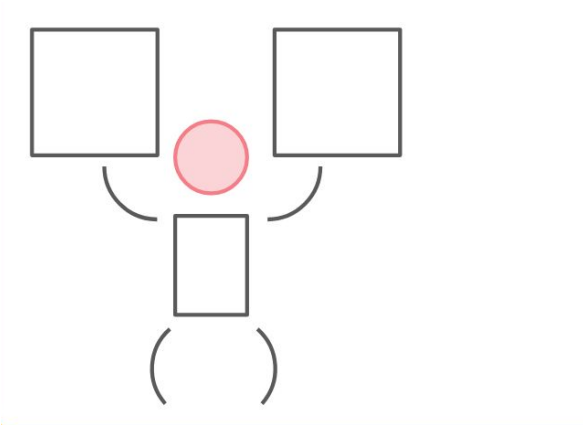
Can Mr X lift one block?



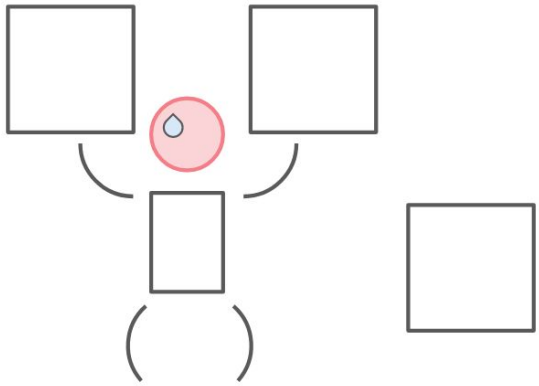
Yes. As a single block is light in weight, Mr X can lift it easily.



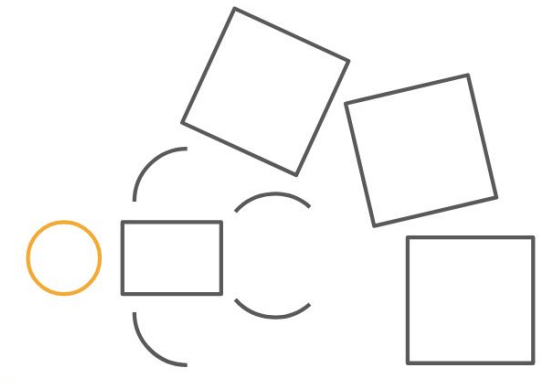
Can Mr X lift two blocks?



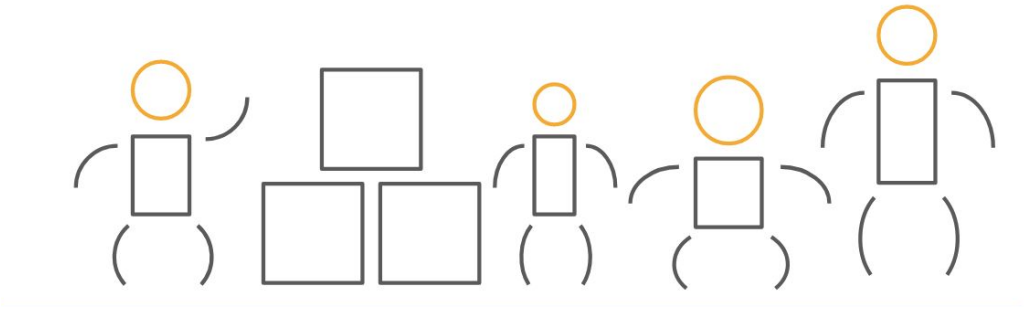
Yes, he can lift them but this time more effort is required as the weight has increased.



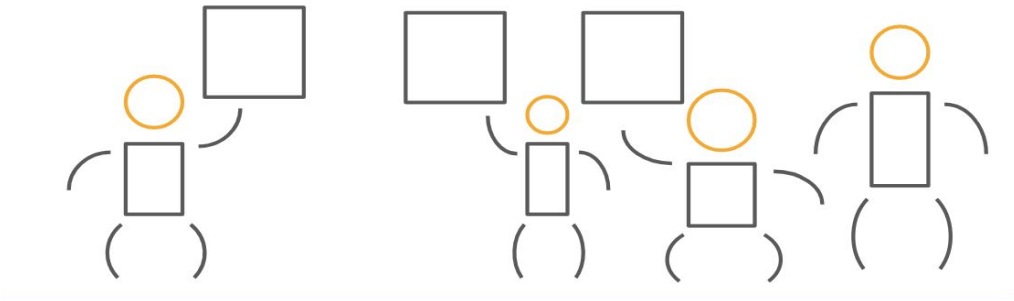
But will he be able to lift one more block?



No. He tries but fails as the weight has increased and it is difficult to lift such a heavy weight altogether.

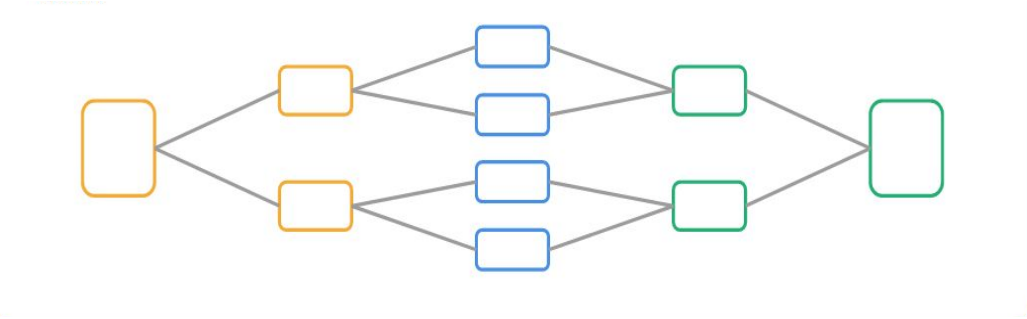


Now, Mr X calls his friends for help. Can they lift three blocks?



Yes, they can easily lift all the three blocks as the weight of all the blocks is divided among Mr X and his friends.

Through this example you have understood that it is difficult for the single system to handle heavy workload. This is why we use distributed systems and divide the workload among different systems.
As we just learnt, distributed systems work on the concept of divide and conquer.



- You take a complex problem that is too large to handle through a single machine.

- Then divide the problem into small segments manageable by a single machine.
- Next, you distribute the workload among a number of autonomous machines and solve the problem.

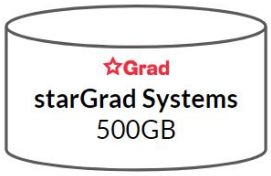
1.2 Need of Distributed Systems

You can understand the need of distributed systems through the following example.

- You are an entrepreneur working on a start-up called starGrad.
- It is a unique online higher education platform providing rigorous industry-relevant programs.
- Your website stores a ton of text-based content and also provides on-demand streaming services to your learners.
- Initially, you only provide services in your locality and your entire model is based on a **single centralised system with 500 TB of storage**

Now, due to the unique selling point of your courses, starGrad becomes quite popular.

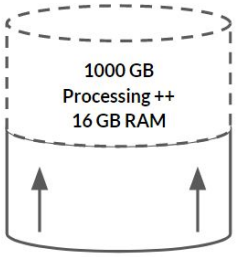
- As starGrad becomes more popular, the traffic on the website increases rapidly as more learners join.
- As an excellent entrepreneur you also want to expand the number courses offered on the platform.
- The 500 TB of storage in your centralised system is not enough to hold the amount of content you intend to provide. You also need more computational power to handle the increased traffic.



Now, you have two choices – vertically scale your system or shift from a centralised system to distributed systems.

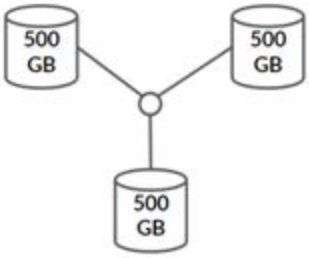
Vertically Scale

Scaling vertically means adding more storage to your centralised system to handle the increased workload. Here, in vertical scale, we are adding 1,000 GB of storage to a centralised system.



Shift to Distributed Systems

Shifting to distributed systems means using different machines and dividing the workload among these machines. Here, we are using three different machines of 500 GB each to divide the workload.



1.3 Centralised Systems vs Distributed Systems

Centralised Systems

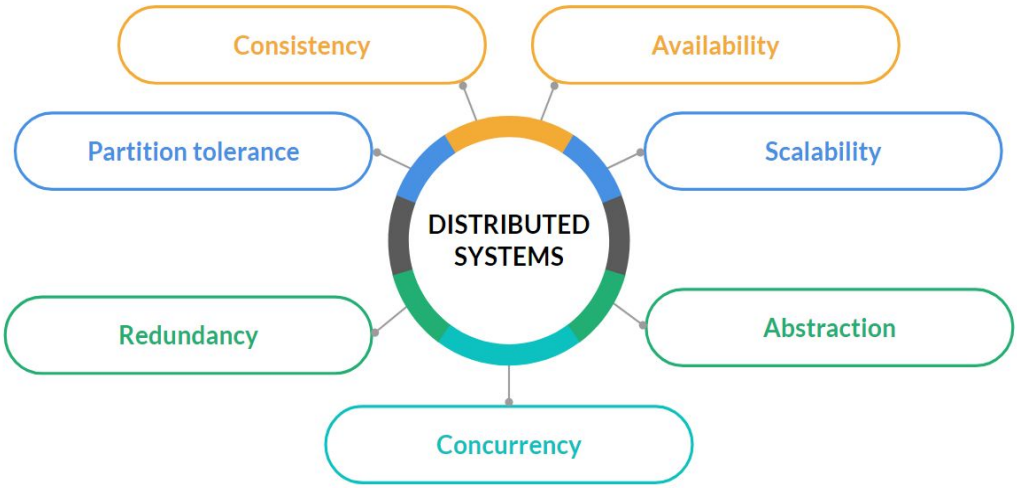
- Limited storage capacity and computational capability
- Only vertically scalable which is very expensive after a limit and involves downtime
- Single point of failure
- Not fault tolerant

Distributed Systems

- No upper limit to storage and computational capabilities in theory
- Offers both vertical and horizontal scalability and, hence, easily scalable
- No single point of failure
- Fault tolerant

1.4 Expectations from Distributed Systems

Following are the features provided by distributed systems.



Redundancy

- Redundancy is about having more than one copy of the data.
- It helps in avoiding the loss of data due to a new partition or node failure.

Hence, in the use case for starGrad, let's say one of the machines in which your '**Machine Learning**' course content is stored crashes. If your distributed system has redundancy, it will have a backup copy of the course in another machine. This makes your model more reliable and failure proof.

Hence, duplicating data so as to ensure having a backup is the main purpose of redundancy.

Consistency

- Consistency means that if two people request for the same data at any given time they should be able to get the exact same result.
- Simply put, performing a read operation will return the value of the most recent write operation, causing all the nodes to return the same data.

For starGrad, let's say a learner reports a mistake in a graded question and you update your platform to rectify the error. Now, to be fair to everyone, the latest update under question should be consistent for all the learners.

Availability

- Availability means the system should be responsive all the time.
- Distributed systems should ensure its availability, that is, even when a component of the system fails, the system should not fail, and another component must replace it and keep the system running.

For starGrad, let's say the nodes in Delhi are down every Friday for maintenance. This should not mean that the learner cannot access the platform on those days. A distributed system allows you to ensure platform availability at all times, which is essential for a great learning experience.

Partition Tolerance

- Partition tolerance means that the system should be responsive and working even if one partition is disconnected from the rest due to certain network or communication issues.
- A partition-tolerant system can sustain any number of network failures but does not result in an entire network failure.

For starGrad, once you move to a distributed system, your content will be spread across different machines in faraway physical locations. Let's say there are some network issues in Mumbai. This creates a partition in your network. If all the content of a particular course and its backups are stored in Mumbai, your system will not be partition tolerant.

Scalability

- Scalability means that the system should be able to scale easily even with an increase in workload.

- By adding cheap commodity hardware, the distributed systems can easily implement horizontal scaling when the workload increases.

For starGrad to accommodate more learners and expand the current list of programs, the platform must be easily scalable. Distributed systems allow you to easily scale up the storage and computation capabilities by adding more machines to the system.

Concurrency

- Concurrency means that the system should be able to perform parallel processing or computation.
- Concurrency in computation allows distributed systems to perform complex tasks in a fast and efficient manner.

For starGrad, as the platform grows and the number of learners increase, computational requirements will also rise. Through concurrency, distributed systems can easily tackle this issue.

Abstraction

- Abstraction means the system should seem like a single system to the user.
- Distributed systems provide a **global view** of all the underlying machines as a single machine and **hide the internal working** of the system from the end users.

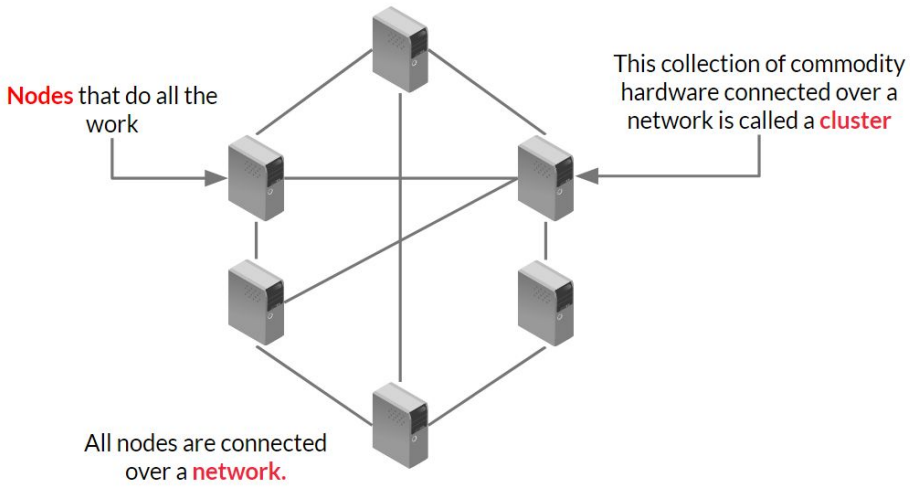
For starGrad, the learner should see your platform as a single isolated service. They should not have to request data from the Bangalore server for the '**Machine Learning**' course or from the Mumbai server for the '**Web Development**' course.

Implementation of Distributed Systems

2.1 Distributed Systems Components

Nodes: Nodes are clients, servers or peers depending upon the type of system.

Cluster: Cluster is defined as a group of nodes working together and configured in such a way that they appear as a single system.



2.2 Architecture of Distributed Systems Architecture

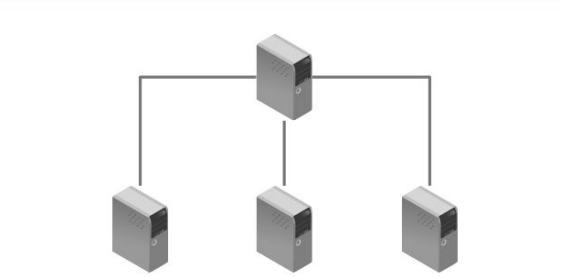
Since these machines share the workload, there needs to be some structure in the arrangement of these clusters to ensure proper resource utilisation and bookkeeping.

Following are the two main ways a cluster of machines are arranged in a distributed system:

- Master-slave architecture
- Master-master architecture

Master-Slave Architecture

- In this architecture, a master node controls and coordinates all the other nodes (slaves) in the network.
- Typically, the master is used for **write** operations while the slaves are used for **read** operations.
- The master ensures consistency among slaves.
- Typically, there is only one master making it a single point of failure.



The architecture can be classified into the following layers:

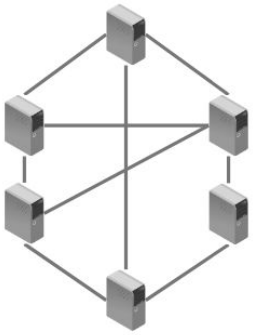
- The top layer, that is, the master layer
- The bottom layer, that is, the slave layer

The master is responsible to make the important decisions and then to coordinate with all the slave nodes in order to accomplish the task. Typically, the master is used for the **WRITE** operations, while the slaves are used for the **READ** operations. Also, the master is responsible to ensure consistency among the slaves. One drawback of this architecture is that if the master node goes down, it will affect the working of all the slave nodes. In order to mitigate this, redundancy is added at the server level by adding a standby master. The examples of master-slave architecture are as follows:

- When MySQL-Postgres is scaled or shared, it behaves in the master-slave order.
- Similarly, MongoDB follows the master-slave model in the NoSQL world.
- Hadoop makes use of the master-slave model for storage (HDFS) as well as data processing (MapReduce).
- Apache Airflow (a pipeline automation tool) makes use of a distributed master-slave architecture by using a scheduler as a master node and workers as the slave nodes.

Master-master Architecture

- In this architecture, all the nodes are the same. They keep the information about others through different communication protocols.
- Data is equally shared by all nodes. Typically, data is distributed by the means of consistent hashing.
- The client can connect to any of the nodes for **Read**/**Write** irrespective of whether data is present in that node or not.



Hence, in this type of architecture, all the nodes are the same and they maintain information about the others by using different communication protocols. The data is equally shared among all the nodes and is distributed by means of consistent hashing. The client can connect to any of the nodes for **READ** or **WRITE** irrespective of whether the data is present in that node or not.

Since all the nodes are equal, they can be easily replaced in case of failure. This makes the model fault tolerant easily. But, a drawback is that there is no central coordinator in the master-master model, which makes it less consistent.

A few examples that leverage the master-master architecture are as follows:

- Cassandra: A NoSQL database with the master-master architecture
- DynamoDB: Amazon's NoSQL database system
- Elasticsearch: An open-sourced distributed search engine

2.3 Challenges in Implementation

The major challenges faced while implementing distributed systems are as follows:

- Performance
- Scalability
- Fault tolerance and reliability
- Security

Performance

- Since distributed systems involve multiple machines connected over a network, the added network hop results in added latency.
- Total time = computation time + network delay

- Bad design/planning or inefficient resource utilisation in distributed systems can result in extremely poor performance or a high response time.

In distributed systems, total time of computation increases because of network delay.

To handle this challenge, you need to follow certain procedures:

- Reduce computation time by using concurrent/parallel computation
- Reduce network delay by increasing bandwidth and redundancy in the network
- Implement proper load-balancing and partitioning schemes
    - Load balancing: Distribute the tasks to available resources in an efficient manner
    - Partitioning schemes: Divide data into logical partitions for easier access
- Optimal redundancy: Too much redundancy can increase write time by a lot, thus, only an optimal number of replications must be made

Example:

- When starGrad initially moved to a distributed model, the response time for individual systems increased.

- To counter this:
    - You leveraged the potential of distributed computing to solve parallel problems using multiple machines.
    - You increased the network bandwidth to deal with network delays.

- This solved the problem more efficiently and managed to bring down the response time to acceptable levels.

Scalability

- A good scalable system should be capable of smoothly increasing the capacity with increasing load.
- Adding a new node to the cluster requires rearrangement/rebalancing of the cluster.
- For example, if the data is distributed among the nodes by simple hashing, adding a new node to this cluster would require rebalancing.

To achieve scalability, you have to perform proper partitioning with the use of **consistent hashing**.

- In scalable distributed systems, the size of system constantly increases which makes simple hashing a poor choice.
- In scalable distributed systems though, the size constantly increases which makes simple hashing a poor choice.
- Consistent hashing is a special type of hashing, in which only a fraction of the keys need to be remapped every time the size changes.

To handle this, you can also use **sharding** which is horizontal partitioning. It refers to the breaking of a huge data set into small partitions.

Vertical Scaling:

- Vertical scaling means adding more power to the machine by increasing the processor or storage capacity.
- It is also known as **scaling up**.

Horizontal scaling:

- Horizontal scaling means scaling by adding more resources to the machine which shares the processing and workload across multiple devices.
- It is also known as **scaling out**.

Example:

- As discussed, starGrad has scaled using horizontal scaling by adding more machines into the cluster.

- This takes care of the computational requirements, but the size of the data sets your company is working on has also grown too large for any single computer in your network. In this case, we can make use of sharding to distribute large data sets among different nodes.

- NoSQL database programs like Cassandra make use of consistent hashing to make their distributed systems elastically scalable.

Fault Tolerance and Reliability

Fault Tolerance:

Distributed systems are based on the following:

- Nodes or commodity hardware
- Network

Both of these make the distributed systems highly susceptible to failures.

- Fault tolerance is a system that is built to provide uptime at all times.
- Fault tolerance can be achieved by incorporating redundancy carefully.

Since storage and computation responsibilities are shared by all the nodes in the distributed system, making the model failure tolerant becomes a challenge. Network failures might result in partitions in the network.

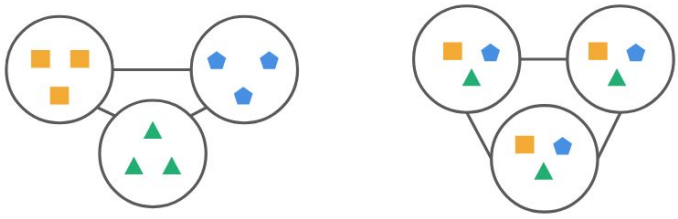
Reliability:

Reliability refers to the property of a system that runs continuously without failure. However, distributed systems are susceptible to fault and the reliability of the system is put at stake.

- Reliability = 1 - probability of failure
- Reliability can be ensured through the following:
    - High fault tolerance
    - Ease of use and maintenance
- Higher reliability results in higher availability but not vice versa

You can handle failure with the following steps:

- **Fault prediction**: Based on the use case, some failures can be predicted and prepared for.
- **Proper maintenance**: Since you are using a commodity hardware, maintenance is a must.
- **Auto recovery**: Planned or automated recovery procedures are used to get backup for the failed nodes or replace the node if its recovery is not possible.
- **Smart redundancy**: This refers to making multiple copies of the data and storing them in different machines that belong to separate partitions.



Example:

- Exam season in starGrad is particularly traffic heavy. Every learner is involved in some last-minute preparation. In such a scenario, you have to ensure your systems are reliable and fault tolerant so that the learners have access to all the resources they need.

- This can be done through the following:
    - Having backup copies of all the content that are distributed among the nodes in a partition-tolerant manner
    - Conducting maintenance checks before exam season when traffic is expected to be heavy

Security

Providing security in the distributed systems is a big challenge as compared to centralised systems as all the data in the centralised systems is present at a single place. However, in distributed systems, the data is dispersed.

Since the components are distributed, managing and maintaining security in a distributed manner is a challenge. In centralised systems, everything is packed in a single place. Hence, everything can be monitored easily.

In distributed systems, multiple systems with different security standards are present in a network, making it vulnerable to attacks in the following layers:

- The storage layer
- The transfer layer

Following are the ways to achieve security in distributed systems:

- Proper authentication: Transient token-based authentication should be done between systems during communication.

- Stateless systems: The nodes should not store any unnecessary data related to past communications/computations.
- Proper vigilance/monitoring: All the nodes must be continuously monitored, and any signs of vulnerability/failure should be fixed right away before it gets exploited.

Example:

- As the starGrad application grows in scale and popularity, it will be more susceptible to hacker attacks trying to steal resources or malicious learners trying to access classified content like solution documents or results.

- To ensure its continued success, you must ensure the platform has consistent security standards, and each node makes proper authentication before relaying any data. For example, only authorised IP addresses can read the data.

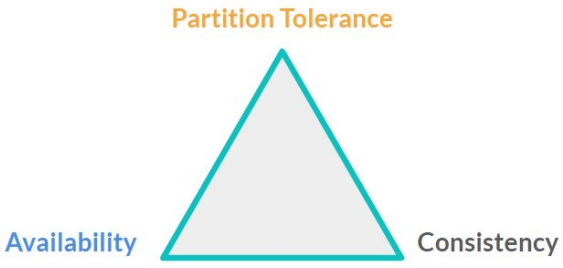
- To understand the concept of stateless systems, you can take the example of a simple telephone network.

2.4 CAP Theorem

CAP Theorem states that at any time in a system, we can only support two of the following three entities:

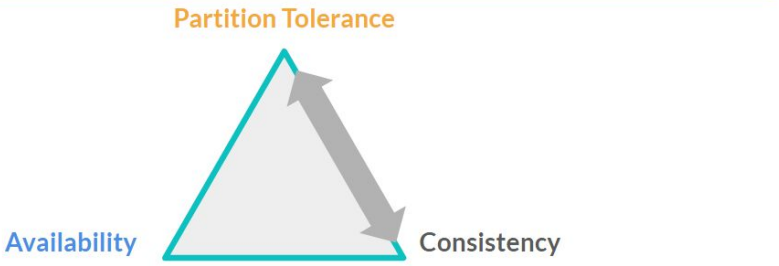
- Consistency
- Availability
- Partition Tolerance

You cannot achieve consistency and availability without partition tolerance. Hence, partition tolerance is a must in distributed systems. According to the CAP theorem, you can support either consistency or availability.



CP systems

In CP systems, C stands for consistency and P stands for partition tolerance. Thus, CP systems have high consistency along with partition tolerance.

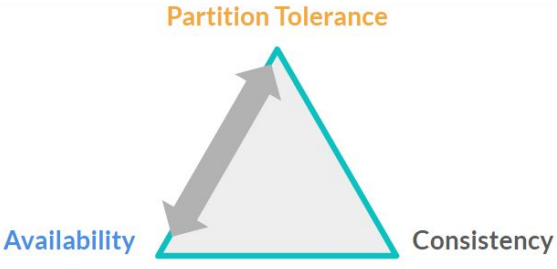


Some examples of CP systems are as follows:

- In a multiplayer game, the position of all the other players needs to be consistent even at the cost of the game being unavailable to new players during high traffic time.
- IRCTC ticket booking needs to be highly consistent to avoid multiple bookings. Here, availability is compromised upon frequently.

AP systems

In AP systems, A stands for availability and P stands for partition tolerance. Thus, AP systems have high consistency along with partition tolerance.



Some examples of AP systems are as follows:

- A social media app such as Twitter or Facebook does not need the number of likes to be strictly consistent for all users. On the other hand, availability is important in such platforms. Hence, an AP model is more suitable.

- Search engines such as Bing and Google are also AP systems as availability is a must, whereas consistency can be compromised on.

Industry Examples of Distributed Systems

3.1 Requirements from a Distributed System

To implement the distributed functionality in the starGrad application, we will need tools to help us in the following areas:

- Storage
- Computation
- Messaging
- Database management

These distributed tools do not have to be created from scratch. There are companies like Amazon, Apache, etc. that provide these functionalities as services.

3.2 Storage: Amazon S3

S3 is Amazon's cloud storage solution where data is stored over a cluster of machines. Some features of S3 are as follows:

- Data is logically grouped into buckets
- Replication/redundancy support to provide fault-tolerance
- No upper limit to the amount of data being stored
- Programming neutral

The distribution and retrieval of data across multiple machines are based on consistent hashing. Also, S3 is eventually a consistent system. S3 is available to end users as APIs over HTTP.

S3 is used when you have to store structured/unstructured data of very high volume/velocity. On the other hand, S3 cannot be used for live applications. Instead, it is used for analytical purposes.

S3 can be used by dumping the log files of different components and applications of the starGrad application. Also, it can be used to store more user data.

Some competitors of Amazon S3 are as follows:

- IBM Cloud (object storage)
- Microsoft Azure (blob storage)
- Google Cloud Storage

3.3 Computation: Apache Spark

Apache Spark provides a powerful distributed computational system and is widely used in the industry due to its high-speed computational ability.

Spark embodies the fault-tolerance of distributed systems and offers lightning-fast speed owing to its in-memory computation model and other optimisation techniques. Also, Spark follows a master-slave architecture, with a master node along with a cluster manager acting as a master and several worker nodes acting as slaves. Let's watch the next video in order to learn more about the use cases of Apache Spark.

The use case of Apache Spark includes:

- Batch data processing,

- Real-time data processing using Spark Streaming,
- Complex ML computations and complicated graph algorithms, and
- PySpark, the Python API for Spark, which is widely used for data science pertaining to Big Data applications.

Following are the alternatives to Spark Streaming:

- Flink
- Amazon Kinesis

3.4 Messaging: Apache Kafka

Apache Kafka is an open-source distributed event streaming platform used as a messaging system for distributed systems.

Kafka is a messaging/streaming system. Following are the salient features of Kafka:

- Fault-tolerant and highly available system
- Provides message ordering guarantee over a distributed design and hence is reliable
- Connects distributed producers and consumers
- Horizontally scalable, i.e., there is no upper limit to the amount (volume/velocity) of messages served which can be attributed to its distributed foundation

Kafka can be used in the following systems/scenarios:

- Building non-blocking systems
- Asynchronous systems
- Streaming data ingestion systems where data is collected in real time

The starGrad application uses the following use cases of Kafka:

- Non-blocking system
- Streaming solutions

Following are the competitors of Kafka:

- ActiveMQ
- RabbitMQ

3.5 DBMS: MongoDB

MongoDB is a distributed NoSQL database management system. MongoDB can be used as a storage layer for live applications (it satisfies transactional online transaction processing needs). It can support analytical (online analytical process) requirements as well owing to its distributed nature. Also, MongoDB has the advantage of both SQL and NoSQL databases.

MongoDB can be used to deal with structured data, but the volume of data that needs to be stored cannot fit in a single machine.

Note: MongoDB uses JSON-like documents with **optional schemas**. This can be considered semi-structured.

Database management systems are used to store the information, which is required to run the applications, whereas Amazon S3 is a data warehouse where all the data of an application can be dumped. The competitors of MongoDB are as follows:

- Cassandra
- Amazon DynamoDB
- IBM Cloudant