Chapter -4.2
NOSQL System and Big Data

- Introduction to NOSQL Systems
- The CAP Theorem
- Document-based, Key-value Stores, Column-based, and Graph-based Systems;
- BigData; MapReduce; Hadoop

Introduction to NOSQL System

- NoSQL, which stands for **"Not Only SQL,"** is a category of database management systems (DBMS) that provides a non-relational approach to data storage and retrieval. Unlike traditional relational databases, which use structured query language (SQL) and uses to a predefined schema to store data, NoSQL systems offer flexible data models that can handle a variety of data types, including structured, semi-structured, and unstructured data.
- Most NOSQL systems are distributed databases or distributed storage systems and it focuses on semi-structured data storage, high performance, availability, data replication, and scalability
- **Structured data** refers to data organized in a fixed format with a predefined schema. It follows a predefined schema or data model, which defines the structure, relationships, and constraints of the data. Structured data is typically stored in relational databases or spreadsheets. Semi-structured data refers to data that does not adhere to a rigid schema but still has some organizational properties. It contains both structured and unstructured elements. **Semi-structured** data may not have a fixed format but often includes tags, labels, or other markers that provide some level of meaning to the data. Semi-structured data is commonly represented in formats like JSON (JavaScript Object Notation), XML (eXtensible Markup Language) etc. **Unstructured data** refers to data that does not have a predefined or organized format. It lacks a consistent structure, making it challenging to store, process, and analyze using traditional relational databases or spreadsheet-like formats. Examples of unstructured data include text documents (such as Word documents or PDFs), audio and video recordings, images, sensor data streams etc.
- NoSQL databases are often used in applications where there is a high volume of data that needs to be processed and analyzed in real-time, such as social media analytics, e-commerce, and gaming. They can also be used for other applications, such as content management systems, document management, and customer relationship management.
- However, NoSQL databases may not be suitable for all applications, as they may not provide the same level of data consistency and transactional guarantees as traditional relational databases. It is important to carefully evaluate the specific needs of an application when choosing a database management system

Emergence of NOSQL System

- NOSQL systems focus on storage of "big data". Typical applications that use NOSQL
  - Social media
  - Web links
  - User profiles
  - Marketing and sales
  - Posts and tweets
  - Road maps and spatial data
  - Email
- Many companies and organizations are faced with applications that store vast amounts of data.
- Consider a free e-mail application, such as Google Mail or Yahoo Mail or other similar service—this application can have millions of users, and each user can have thousands of e-mail messages.
- There is a need for a storage system that can manage all these e-mails however a structured relational SQL system may not be appropriate because
  - SQL systems offer **too many services** (powerful query language, concurrency control, etc.), which this application may not need;
  - A structured data model such the traditional relational model may be **too restrictive.**

- Although newer relational systems do have more complex object-relational modeling options they still require schemas, which are not required by many of the NOSQL systems.
- As another example, consider an application such as Facebook, with millions of users who submit posts, many with images and videos; then these posts must be displayed on pages of other users using the social media relationships among the users. User profiles, user relationships, and posts must all be stored in a huge collection of data stores, and the appropriate posts must be made available to the sets of users that have signed up to see these posts. Some of the data for this type of application is not suitable for a traditional relational system and typically needs multiple types of databases and data storage systems.
- Some of the organizations that were faced with these data management and storage applications decided to develop their own systems. For example:
    - Google developed a proprietary NOSQL system known as BigTable (column based), which is used in many of Google's applications that require vast amounts of data storage, such as Gmail, Google Maps etc.
    - Amazon developed a NOSQL system called DynamoDB that is available through Amazon's cloud services. This innovation led to the category known as key-value data stores or sometimes key-tuple or key-object data stores.
    - Facebook developed a NOSQL system called Cassandra, which is now open source and known as Apache Cassandra. This NOSQL system uses concepts from both key-value stores and column-based systems.
    - MongoDB is developed and is available to the users who need it. This is classified as document-based NOSQL systems or document stores.
    - Another category of NOSQL systems is the graph-based NOSQL systems, or graph databases; these include Neo4J and GraphBase.
- NoSQL databases are a relatively new type of database management system that have gained popularity in recent years due to their scalability and flexibility. They are designed to handle large amounts of unstructured or semi-structured data and can handle dynamic changes to the data model. This makes NoSQL databases a good fit for modern web applications, real-time analytics, and big data processing.

Characteristics of NOSQL System

- **Non-relational data model:** NoSQL databases provide data models that differ from the structured, tabular model used in relational databases. NoSQL databases support various data models, such as key-value pairs, documents, columns, and graphs. This flexibility allows developers to choose the most suitable data model based on the specific requirements of their applications.
- **Dynamic schema:** NoSQL databases do not have a fixed schema and can accommodate changing data structures without the need for migrations or schema alterations. The flexibility of not requiring a schema is achieved in many NOSQL systems by allowing semi-structured, self-describing data There are various languages for describing semi structured data, such as JSON (JavaScript Object Notation) and XML (Extensible Markup Language)
- **Scalability:** In NOSQL systems, horizontal scalability is generally used. NoSQL databases are designed to scale out by adding more nodes to a database cluster, making them well-suited for handling large amounts of data and high levels of traffic.
- **Availability, Replication and Eventual Consistency:** Many applications that use NOSQL systems require continuous system availability. To accomplish this, data is replicated over two or more nodes in a transparent manner, so that if one node fails, the data is still available on other nodes. Replication improves data availability and can also improve read performance, because read requests can often be serviced from any of the replicated data nodes. However, write performance becomes more cumbersome because an update must be applied to every copy of the replicated data items; this can slow down write performance if serializable consistency is required. Many NOSQL applications do not require serializable consistency, so more relaxed forms of consistency known as eventual consistency are used. In an eventual consistency model, data updates may take some time to propagate across all nodes in the system. However this may introduce temporary data inconsistencies
- **Replication Models:** Two major replication models are used in NOSQL systems: master-slave and master-master replication. Master-slave replication requires one copy to be the master copy; all write operations must be applied to the master copy and then propagated to the slave copies, usually using eventual consistency (the slave copies will eventually be the same as the master copy). For read, the master-slave paradigm can be configured in various ways. One configuration requires all reads to also be at the master copy, so this would be similar to the primary site or primary copy methods of

distributed concurrency control with similar advantages and disadvantages. Another configuration would allow reads at the slave copies but would not guarantee that the values are the latest writes, since writes to the slave nodes can be done after they are applied to the master copy.

The master-master replication allows reads and writes at any of the replicas but may not guarantee that reads at nodes that store different copies see the same values. Different users may write the same data item concurrently at different nodes of the system, so the values of the item will be temporarily inconsistent. A reconciliation method to resolve conflicting write operations of the same data item at different nodes must be implemented as part of the master-master replication scheme.

- **Sharding of Files:** In many NOSQL applications, files (or collections of data objects) can have many millions of records (or documents or objects), and these records can be accessed concurrently by thousands of users. So, it is not practical to store the whole file in one node. Sharding (also known as horizontal partitioning) of the file records is often employed in NOSQL systems. This serves to distribute the load of accessing the file records to multiple nodes. The combination of sharding the file records and replicating the shards works in tandem to improve load balancing as well as data availability.
- **Distributed architecture:** NoSQL systems are designed to operate in distributed environments, where data is partitioned and distributed across multiple nodes.

Advantage of NOSQL

- **Scalability:** NoSQL databases are designed to scale horizontally, allowing for easy distribution of data across multiple servers. They can handle large volumes of data and high traffic loads more efficiently.
- **Flexibility in Data Models:** NoSQL databases offer flexible data models, such as key-value pairs, document stores, columnar databases, and graph databases. This flexibility enables developers to store and process data in a way that best suits their application requirements.
- **Performance:** NoSQL databases are optimized for read and write performance. They can handle high-speed data ingestion, real-time analytics, and fast retrieval of large datasets, making them suitable for use cases that require low latency and high throughput.
- **Handling Unstructured Data:** NoSQL databases excel at handling unstructured or semi-structured data, such as social media feeds, sensor data, logs, and multimedia content. They allow for easy storage and retrieval of diverse data types without the need for a predefined schema.
- **Distributed and Fault-Tolerant Architecture:** NoSQL databases are designed with a distributed architecture that ensures data availability and fault tolerance. They employ replication and sharding techniques to provide high availability even in the presence of hardware failures or network partitions.
- **Easy Scalability:** Adding more servers to a NoSQL database cluster can be relatively straightforward, allowing for seamless scalability as data volume and traffic increase. NoSQL databases can handle the demands of rapidly growing applications without sacrificing performance.
- **Cost-Effectiveness:** NoSQL databases can be more cost-effective for certain use cases due to their ability to run on commodity hardware and their horizontal scalability, which reduces the need for expensive hardware upgrades.

Disadvantage of NOSQL

- **Lack of Standardization:** Unlike SQL databases, which adhere to a standardized query language (SQL), NoSQL databases lack a unified query language. Each NoSQL database may have its own proprietary query language, making it more challenging for developers who are familiar with SQL to adapt to different databases.
- **Limited Querying Capabilities:** NoSQL databases often prioritize high scalability and performance over complex querying capabilities. They may have limited support for advanced queries, joins, and aggregations compared to SQL databases.
- **Data Integrity and Consistency:** NoSQL databases often sacrifice strict data consistency in favor of high availability and partition tolerance. They may use eventual consistency models, where data updates may take some time to propagate across the system, leading to temporary inconsistencies in data. This trade-off may not be suitable for applications that require strong data integrity and consistency guarantees.
- **Lack of Mature Ecosystem:** While popular NoSQL databases have gained significant adoption, the overall ecosystem, tooling, and community support may not be as mature as that of SQL databases. This can result in limited resources, fewer integration options, and potentially slower support for emerging technologies.

- **Learning Curve and Development Complexity:** NoSQL databases often require developers to learn new data models, query languages, and design patterns. This learning curve can add complexity to development efforts, especially for teams accustomed to working with SQL databases.
- **Limited Transaction Support:** NoSQL databases may have limited support for transactions, especially across multiple documents or entities. ACID (Atomicity, Consistency, Isolation, Durability) properties may not be fully guaranteed in all NoSQL databases, which can be a limitation for applications with strict transactional requirements.
- **Data Migration and Integration Challenges:** Moving data from existing SQL databases to NoSQL databases, or integrating NoSQL databases with existing systems, can be challenging. Data migration, transformation, and synchronization processes may require additional effort and considerations.

RDBMS vs NOSQL

- RDBMS (Relational Database Management System) and NoSQL (Not Only SQL) are two different types of database management systems, each with its own characteristics and use.
- RDBMS is a type of database management system that organizes and stores data in a structured manner using tables, rows, and columns. It follows the relational model and enforces relationships between tables using primary and foreign keys. RDBMS databases are based on the ACID (Atomicity, Consistency, Isolation, Durability) properties and ensure data integrity through normalization and constraints. SQL (Structured Query Language) is commonly used to manipulate and query data in RDBMS. Popular examples of RDBMS include MySQL, PostgreSQL, Oracle, and SQL Server.
- NoSQL databases are a broad category of databases that provide alternatives to traditional RDBMS systems. NoSQL databases are designed to handle unstructured or semi-structured data and offer flexibility in data models. They can be classified into various types based on the data model, such as key-value stores, document stores, columnar databases, and graph databases. NoSQL databases are often horizontally scalable, allowing for easy distribution of data across multiple servers. They may or may not support ACID properties, and their query languages can vary depending on the specific database. Popular examples of NoSQL databases include MongoDB, Cassandra, Redis, Elasticsearch, and Neo4j.

When to use RDBMS:

- When data has a well-defined structure and complex relationships.
- When maintaining data integrity and consistency is critical.
- When ACID compliance is required.
- When working with structured data in traditional enterprise applications.

When to use NoSQL:

- When dealing with large volumes of unstructured or semi-structured data.
- When scalability and high availability are crucial.
- When flexible data models are needed to adapt to evolving requirements.
- When real-time analytics or web applications require fast read and write performance.
- When traditional relational databases impose limitations or performance bottlenecks.

Parameter	RDBMS (Relational Database)	NoSQL (Non-Relational Database)
Data Model	Relational model with tables, rows, and columns	Flexible data models: key-value, document, columnar, graph, etc.
Schema	Requires a predefined schema	Typically schema-less or flexible schemas
Data Relationships	Enforces relationships between tables using primary/foreign keys	Relationships can be flexible, embedded, or graph-based
Scalability	Vertical scaling (adding more resources to a server)	Horizontal scaling (adding more servers to distribute load)
Query Language	SQL (Structured Query Language)	Varies depending on the NoSQL database
ACID Compliance	Supports ACID properties (Atomicity, Consistency, Isolation, Durability)	ACID properties may or may not be supported
Data Integrity	Ensures data integrity through normalization and constraints	Relies on application-level enforcement of data integrity
Scalability and Performance	Well-suited for structured data and complex queries	Well-suited for handling large volumes of unstructured data, high read/write scalability
Use Cases	Traditional enterprise applications, structured data	Web applications, real-time analytics, unstructured or semi-structured data
Examples	MySQL, PostgreSQL, Oracle	MongoDB, Cassandra, Redis, Elasticsearch, Neo4j, etc.

Example:
Let us take data of Person stored in RDBMS. In RDBMS, data is stored in tabular form as below.

Person

Name	Phone
Hari	98465
Sita	98137

Orders

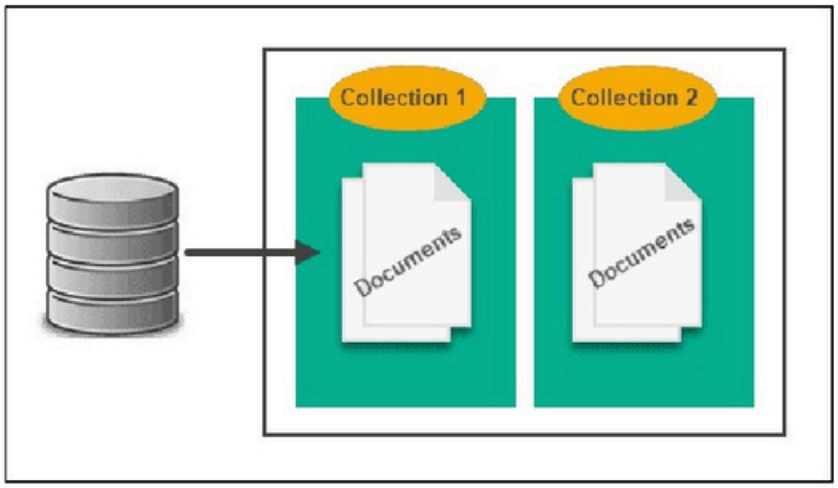
NameD	Date	Product
Hari	2079	Laptop
Sita	2080	Desktop
Sita	2080	Mouse

In, NoSQL, we can represent above three relations(structured format) in JSON (Semi structured format) as below.

```
{
        "Person":[
        {"Name":"Hari","Phone":98465, "Orders":{"Date":2079,"Product":"Laptop"}},
        {"Name":"Sita","Phone":98137,"Orders":[{"Date":2080,"Product":"Desktop"},{"Date":2080,"Product":"Mouse"}]}
        ]
}
```

Types of NOSQL Database

- NOSQL systems have been characterized into four major categories, with some additional categories that encompass other types of systems. The most common categorization lists the following four major categories:
  1. **Document-based NOSQL systems**:
     - These systems store data in the form of documents using well-known formats, such as JSON (JavaScript Object Notation).

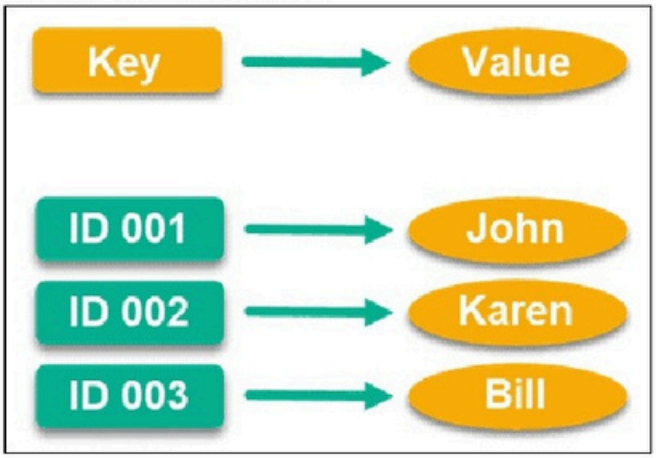


     - Document-based or document-oriented NOSQL systems typically store data as collections of similar documents. These types of systems are also sometimes known as document stores.
     - Documents are accessible via their document id, but can also be accessed rapidly using other indexes.
     - Although the documents in a collection should be similar, they can have different data elements (attributes), and new documents can have new data elements that do not exist in any of the current documents in the collection.
     - Documents can be specified in various formats, such as XML and JSON.
     - MongoDB is a popular document database that provides a flexible and scalable solution for handling document-oriented data.
     - The below example shows simple example of document in MongoDB.

```
{
 "_id": 1001,
 "name": "John Doe",
 "email": "johndoe@example.com",
 "age": 30,
 "posts": [
  {
   "_id": 1,
   "title": "Post 1",
   "content": "This is the content of Post 1"
  },
  {
   "_id": 2,
   "title": "Post 2",
   "content": "This is the content of Post 2"
  }
 ]
}
```

In this example, we have a "users" collection and a "posts" collection. Each user can have multiple posts associated with them. We can represent this relationship by embedding the posts within the user document. This structure allows us to retrieve all the users and their posts by querying using the document id or index.

2. **NOSQL key-value stores:**
   - These systems have a simple data model based on fast access by the key to the value associated with the key. Key value database are the simplest form of NOSQL database.



   - A key-value NoSQL database is a type of database that stores data as a collection of key-value pairs.
   - Each data entry consists of a unique identifier (key) and its corresponding value. The value can be any data type, such as a string, number, object etc.
   - In a key-value database, the key serves as the primary means of accessing and retrieving the associated value. This simple data model makes key-value databases highly efficient and scalable, as they can provide fast read and write operations.
   - Key-value NoSQL databases are designed to handle high volumes of data and are well-suited when quick and direct access to data based on its unique identifier is essential.
   - Amazon developed a NOSQL system called DynamoDB that is available through Amazon's cloud services. This innovation led to the category known as key-value data stores or sometimes key-tuple or key-object data stores.
   - Document-based NoSQL databases store data in flexible, nested documents and allow for complex data structures and querying, while key-value-based NoSQL databases store data as simple key-value pairs and provide efficient key-based lookup and retrieval.
   - Here is an example of key-value pair(user data) stored in DynamoDB.

```
{
  "user_id": "12345678",
  "name": "John Doe",
  "email": "johndoe@example.com",
  "age": 30
}
```

Queries in DynamoDB typically involve retrieving items based on the key.

3. **Column-based or wide column NOSQL systems:**
   - A wide-column database is a NoSQL database that organizes data storage into flexible columns (a form of vertical partitioning) that can be spread across multiple servers or database nodes, using multi-dimensional mapping to reference data by column, row, and timestamp.
   - The Google distributed storage system for big data, known as BigTable, is a well-known example of this class of NOSQL systems, and it is used in many Google applications that require large amounts of data storage, such as Gmail. Big-Table uses the Google File System (GFS) for data storage and distribution.
   - Columnar databases store each column in a separate file. Consider the example below: one file stores id and name, another only the grade and id another the gpa and id and so on. Each column in a row is governed by auto-indexing. Each column functions almost as an index, meaning a scanned/queried column offset corresponds to the other columnal offsets in that row in their respective files.

Row-oriented

ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

Column-oriented

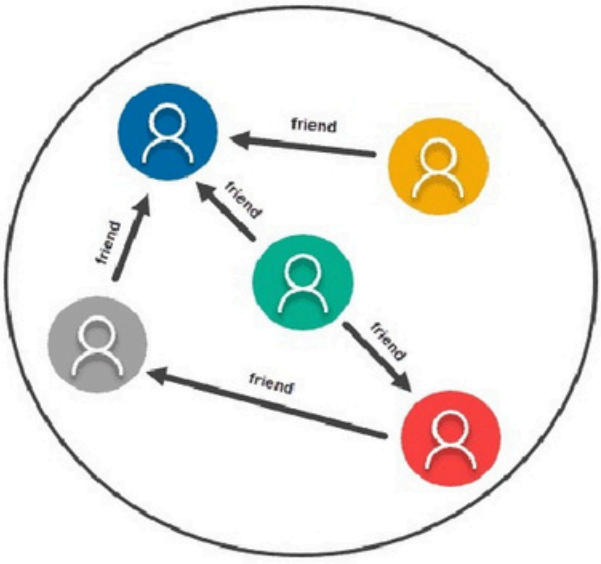
Name	ID
John	001
Karen	002
Bill	003

Grade	ID
Senior	001
Freshman	002
Junior	003

GPA	ID
4.00	001
3.67	002
3.33	003

- A relational database management system (RDBMS) stores data in a table with rows that all span a number of columns. If one row needs an additional column, that column must be added to the entire table, with null or default values provided for all the other rows. If you need to query that RDBMS table for a value that isn't indexed, the table scan to locate those values will be very slow.
- Wide-column NoSQL databases still have the concept of rows, but reading or writing a row of data consists of reading or writing the individual columns. A column is only written if there's a data element for it. Each data element can be referenced by the row key, but querying for a value is optimized like querying an index in a RDBMS, rather than a slow table scan.
- The most significant benefit of having column-oriented databases is that you can store large amounts of data within a single column. This feature allows you to reduce disk resources and the time it takes to retrieve information from it. They are also excellent in situations when you have to spread data across multiple servers.

4. **Graph-based NOSQL systems:**
   - Another category of NOSQL systems is known as graph databases or graph-oriented NOSQL systems.
   - In this model, data/entities/objects are represented as nodes (also known as vertices) connected by edges (also known as relationships). It allows for the efficient representation and traversal of complex relationships between entities.
   - A node can have properties associated with them. Relationships between nodes are represented as edges
   - Graph databases excel at managing highly interconnected data, such as social networks, recommendation systems, fraud detection, and knowledge graphs. They provide a powerful way to express and query complex relationships and patterns within the data. Example: Social networks often use graphs to store information about how their users are linked.



- Neo4j, which is used in many applications is an example of Graph-based NOSQLS database. Neo4j is an open-source system, and it is implemented in Java

The CAP Theorem

- In a system with data replication, concurrency control becomes more complex because there can be multiple copies of each data item.
- So, if an update is applied to one copy of an item, it must be applied to all other copies in a consistent manner.
- The possibility exists that one copy of an item X is updated by a transaction T1 whereas another copy is updated by a transaction T2, so two inconsistent copies of the same item exist at two different nodes in the distributed system. If two other transactions T3 and T4 want to read X, each may read a different copy of item X.
- The CAP theorem, which was originally introduced as the CAP principle, can be used to explain some of the competing requirements in a distributed system with replication.
- The three letters in CAP refer to three desirable properties of distributed systems with replicated data: **consistency, availability and partition tolerance.**
- **Consistency**
  - It means that the nodes will have the same copies of a replicated data item visible for various transactions.
  - In CAP, the term consistency refers to the consistency of the values in different copies of the same data item in a replicated distributed system.
  - There are various types of consistency models. Consistency in CAP refers to eventual consistency, a very strong form of consistency.
- **Availability**
  - It means that each read or write request for a data item will either be processed successfully or will receive a message that the operation cannot be completed.
  - Every non-failing node returns a response for all the read and write requests in a reasonable amount of time. In simple terms, every node (on either side of a network partition) must be able to respond in a reasonable amount of time.
- **Partition tolerance**
  - It means that the system can continue operating if the network connecting the nodes has a fault that results in two or more partitions, where the nodes in each partition can only communicate among each other.
  - That means, the system continues to function and upholds its consistency guarantees in spite of network partitions. Network partitions are a fact of life

- **The CAP theorem states** that it is not possible to guarantee all three of the desirable properties—consistency, availability, and partition tolerance—at the same time in a distributed system with data replication. If this is the case, then the distributed system designer would have to choose two properties out of the three to guarantee but not all three.
- It is generally assumed that in many traditional (SQL) applications, guaranteeing consistency through the ACID properties is important. On the other hand, in a NOSQL distributed data store, a weaker consistency level is often acceptable, and guaranteeing the other two properties (availability, partition tolerance) is important. Hence, weaker consistency levels are often used in NOSQL system instead of guaranteeing serializability. In particular, a form of consistency known as eventual consistency is often adopted in NOSQL systems. Eventual consistency is a property of distributed systems where updates to data eventually propagate across all nodes, ensuring that all replicas eventually reach the same state. In other words, it guarantees that, given enough time and absence of further updates, all replicas will eventually converge and become consistent. Unlike immediate consistency models, where every read operation always returns the latest written value, eventual consistency allows temporary inconsistencies or divergences between replicas. This means that different replicas may observe different versions of the data for a certain period until the updates propagate and all replicas synchronize.
- The following figure represents which database systems prioritize specific properties at a given time:

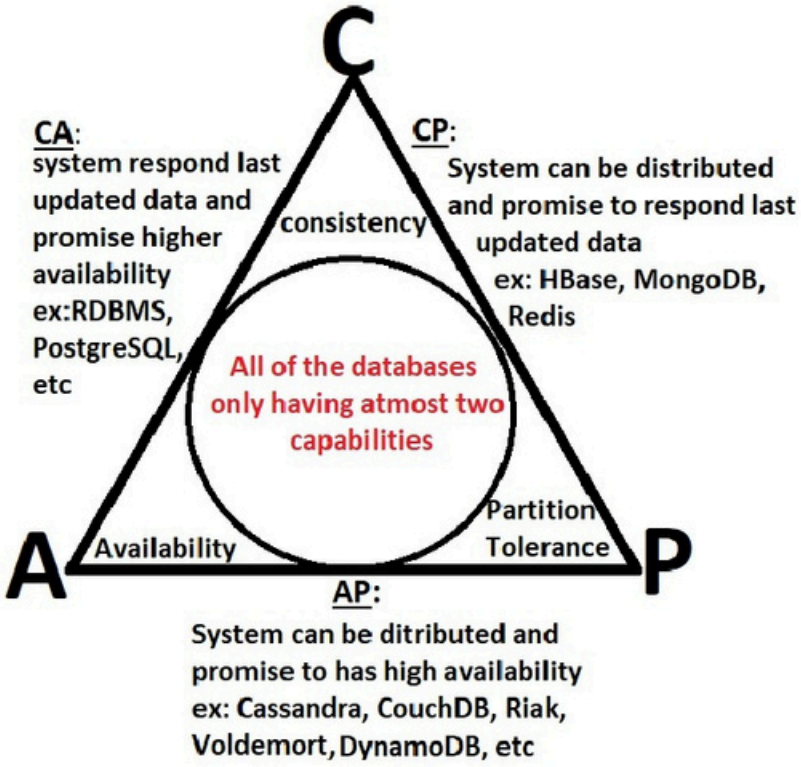


FIG: CAP Theorem Representation with Example

- NoSQL databases are classified based on the two CAP characteristics they support:
  1. CP database:
     - A CP database delivers consistency and partition tolerance at the expense of availability.
     - When a partition occurs between any two nodes, the system has to shut down the non-consistent node (i.e., make it unavailable) until the partition is resolved.
  2. AP database:
     - An AP database delivers availability and partition tolerance at the expense of consistency.
     - When a partition occurs, all nodes remain available but those at the wrong end of a partition might return an older version of data than others. (When the partition is resolved, the AP databases typically resync the nodes to repair all inconsistencies in the system.)
  3. CA database:
     - A CA database delivers consistency and availability across all nodes.
     - It can't do this if there is a partition between any two nodes in the system, however, and therefore can't deliver fault tolerance.

Big Data

- Big data is data that contains greater variety, arriving in increasing volumes and with more velocity. Big data is characterized by its volume, velocity, and variety, often known as the three Vs. Other characteristics, such as veracity and value, have been added to the definition by other researchers.
- Big data refers to extremely large and complex sets of data that cannot be effectively processed using traditional data processing techniques.
- It encompasses vast amounts of information collected from various sources, such as social media, sensors, IOT's, online transactions, and more.
- The 5V'of Bigdata are
  1. **Volume:**
     - Big data involves the collection and storage of massive amounts of data. With the explosion of digital technologies, organizations and individuals generate data at an unprecedented scale. This data includes structured data (e.g., databases) and unstructured data (e.g., text, images, videos). For some organizations, data volume might be tens of terabytes of data. For others, it may be hundreds of petabytes.

2. **Velocity:** Big data is generated and collected at a high velocity. Data streams in rapidly and requires real-time or near real-time processing. Examples of high-velocity data sources include stock market feeds, social media posts, and IoT devices. Some internet-enabled smart products operate in real time or near real time and will require real-time evaluation and action.
3. **Variety:** Big data encompasses diverse data formats and types, including structured, semi-structured, and unstructured data. Structured data can be organized in a predefined format (e.g., a table), while unstructured data lacks a predefined structure (e.g., text documents, audio files).

- Two more Vs have emerged over the past few years: **value** and **veracity**.

4. **Value**: Value" refers to the insights, knowledge, or actionable information that can be extracted from the data. The value of big data lies in the potential to gain meaningful insights, make informed decisions, and create business value. By analyzing large and diverse datasets, organizations can uncover patterns, correlations, trends, and other valuable information that can drive improvements, innovation, and competitive advantages. With an increased volume of big data, we can make more accurate and precise business decisions.
5. **Veracity:** Veracity has two built-in features: the credibility(reliability) of the source, and the suitability(fitness) of data for its target audience. It is closely related to trust listing veracity as one of the dimensions of big data amounts to saying that data coming into the so-called big data applications have a variety of trustworthiness, and therefore before we accept the data for analytical or other applications, it must go through some degree of quality testing and credibility analysis. Many sources of data generate data that is uncertain, incomplete, and inaccurate, therefore making its veracity questionable.

- To effectively process and analyze big data, technologies like distributed computing, parallel processing, and cloud computing are commonly used. Additionally, specialized software frameworks and tools, such as Apache Hadoop, Apache Spark, and NoSQL databases, have emerged to handle the challenges posed by big data.

Map Reduce and Hadoop

- Apache Hadoop is an open-source framework that is used to efficiently store and process large datasets ranging in size from gigabytes to petabytes of data. Instead of using one large computer to store and process the data, Hadoop allows clustering multiple computers to analyze massive datasets in parallel more quickly.
- The two core components of Hadoop are the MapReduce programming paradigm and HDFS, the Hadoop Distributed File System (HDFS).

MapReduce Programming Paradigm

- MapReduce is a programming model and a computational algorithm specifically designed to process and analyze large volumes of data in parallel across distributed systems.
- It was popularized by Google and has become a fundamental component of big data processing frameworks like Apache Hadoop.
- The map reduce model consist of following phases.

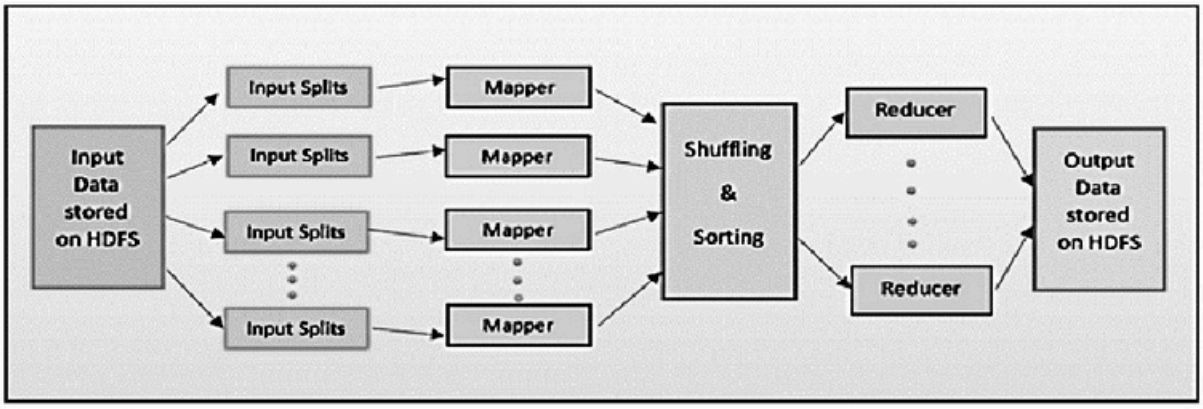


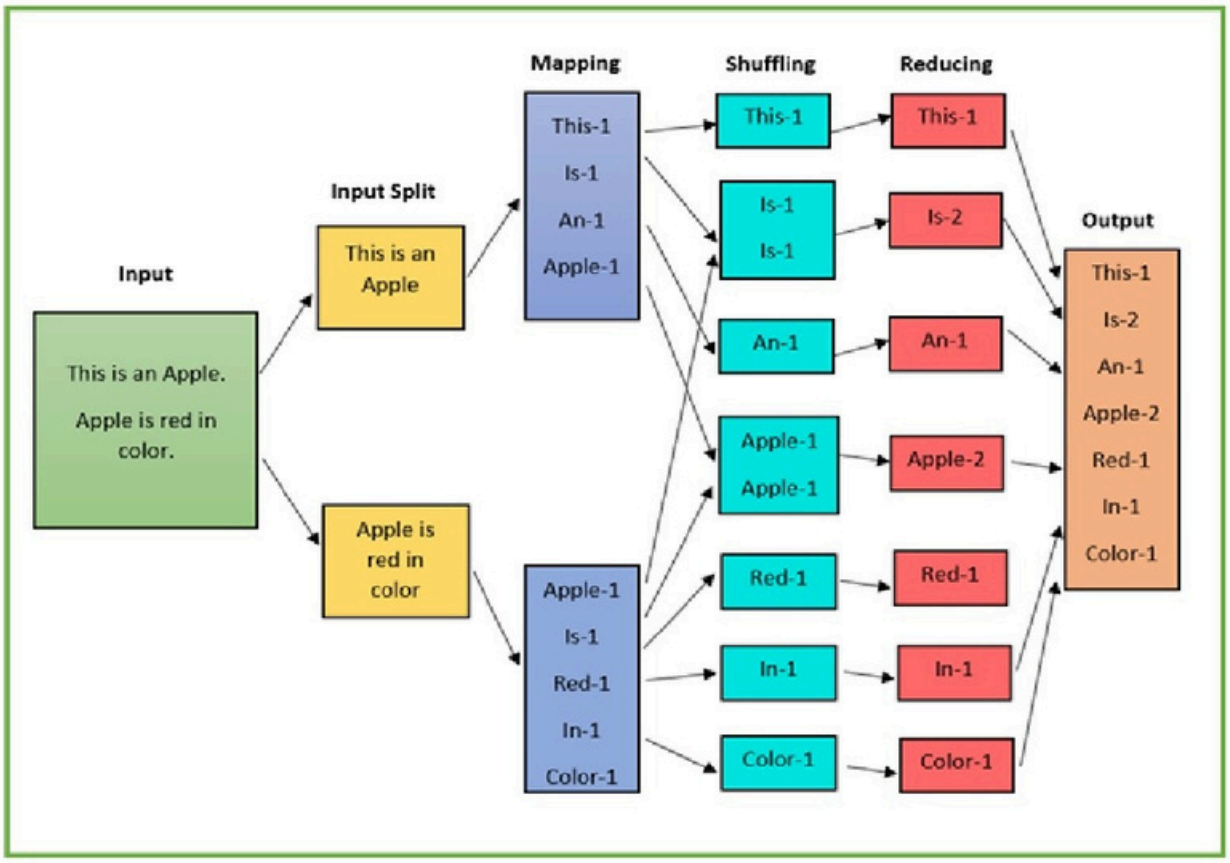
Fig: Map reduce Model

1. **Input split:** In this phase, MapReduce splits the input into smaller chunks called input splits, representing a block of work with a single mapper task.

2. **Map phase:** After the the input data is divided into smaller chunks, in this phase, a "map" function is applied to each chunk independently. The map function takes an input **key-value** pair and **generates intermediate key-value pairs** as output. It performs data transformation and filtering based on the specific task requirements.
3. **Shuffle and Sort:** After the Map phase, the intermediate key-value pairs are shuffled and sorted based on the keys. This ensures that all the values associated with a particular key are grouped together, preparing the data for the Reduce phase.
4. **Reduce phase**: In this phase, the "reduce" function is applied to each unique key, along with its corresponding set of values from the intermediate data. The reduce function aggregates, summarizes, or processes the values to produce the final output. The output of the reduce function is typically stored in a persistent storage system. HDFS is then used to store the final output.

Example:

Here's an example of using MapReduce to count the frequency of each word in an input text. The text is, "This is an apple.Apple is red in color.



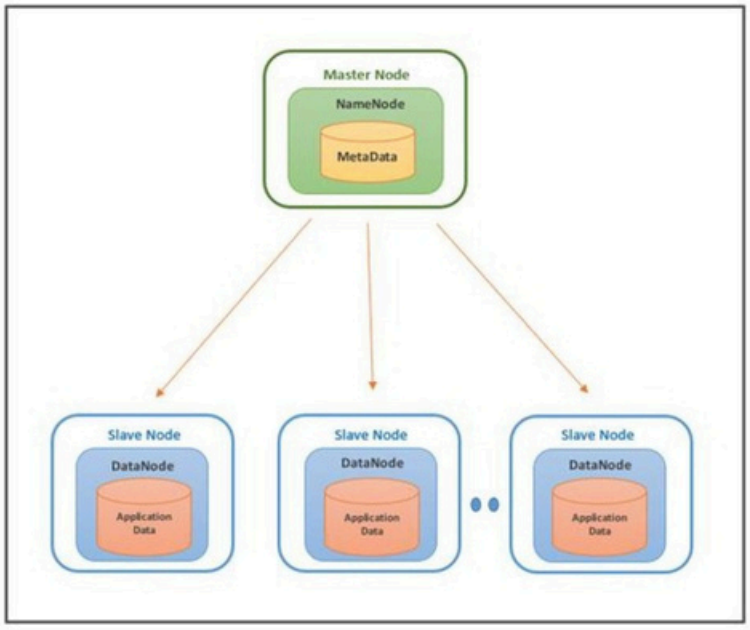
- The input data is divided into multiple segments, then processed in parallel to reduce processing time. In this case, the input data will be divided into two input splits so that work can be distributed over all the map nodes.
- The Mapper counts the number of times each word occurs from input splits in the form of key-value pairs where the key is the word, and the value is the frequency.
- For the first input split, it generates 4 key-value pairs: This, 1; is, 1; an, 1; apple, 1; and for the second, it generates 5 key-value pairs: Apple, 1; is, 1; red, 1; in, 1; color.
- It is followed by the shuffle phase, in which the values are grouped by keys in the form of key-value pairs. Here we get a total of 6 groups of key-value pairs.
- The same reducer is used for all key-value pairs with the same key.
- All the words present in the data are combined into a single output in the reducer phase. The output shows the frequency of each word.
- Here in the example, we get the final output of key-value pairs as This, 1; is, 2; an, 1; apple, 2; red, 1; in, 1; color, 1.
- The record writer writes the output key-value pairs from the reducer into the output files, and the final output data is by default stored on HDFS.

Advantage of Map Reduce

- MapReduce can define mapper and reducer in several different languages using Hadoop streaming.
- MapReduce facilitates automatic parallelization and distribution, reducing the time required to run programs.
- Processing of data using MapReduce is a cost-effective solution.
- MapReduce processes large volumes of unstructured data very quickly.
- MapReduce is flexible and works with several Hadoop languages to handle and store data.

Hadoop Distributed File System (HDFS)

- It is difficult to maintain huge volumes of data in a single machine. Therefore, it becomes necessary to break down the data into smaller chunks and store it on multiple machines.
- Filesystems that manage the storage across a network of machines are called distributed file systems.
- Hadoop Distributed File System (HDFS) is the primary data storage system used by Hadoop applications. All data stored on Hadoop is stored in a distributed manner across a cluster of machines.
- HDFS has two components, which are as follows:
  1. Namenode
  2. Datanode



- HDFS is based in a master-worker architecture, this means that there are one master node and several worker nodes in the cluster. The master node is the Namenode. And the worker node is data node.
- Namenode is the master node that runs on a separate node in the cluster. In HDFS, there is one active NameNode and one or more standby NameNodes. The Active NameNode serves all client requests, and the standby NameNode handles high availability configuration. Suppose **Namenode needs to be restarted**, which can happen in case of a failure. And during this restarting time, the filesystem would be offline. Therefore, to solve this problem, Secondary Namenode come into action.
- Functions of Namenode includes.
  - It manages the filesystem namespace which is the filesystem tree or hierarchy of the files and directories.
  - It stores metainformation information like owners of files, file permissions, etc. for all the files.
  - It is also aware of the locations of all the blocks of a file and their size.
  - It manages the client access requests for the actual data files.
- Datanodes are the worker nodes of HDFS, which can be n in number. They are inexpensive commodity hardware that can be easily added to the cluster. Datanodes are responsible for storing, retrieving, replicating, deletion, etc. of blocks when asked by the Namenode. Functions of DataNode includes.
  - It stores actual data in HDFS.
  - As instructed by the NameNode, the DataNodes are responsible for storing and deleting blocks and replicating these blocks.

- This node handles the client's read and writes requests.
- DataNodes are synchronized to communicate and ensure that data is balanced across the cluster, moving data for high replication, and copying data as needed. They periodically send heartbeats to the Namenode so that it is aware of their health.