Chapter 4

DISTRIBUTED DATABASES, NOSQL SYSTEMS AND BIGDATA

Chapter Outline

After comprehensive study of this chapter, you will be able to:

- Distributed Database Concepts and Advantages
- Data Fragmentation, Replication and Allocation Techniques for Distributed Database Design
- Types of Distributed Systems
- Distributed Database Architecture
- Introduction to NOSQL Systems
- The CAP Theorem
- Document-based, Key-value Stores, Column-based and Graph-based System
- Big Data
- MapReduce
- Hadoop

Scanned with CamScanner

DISTRIBUTED DATABASE CONCEPT AND ADVANTAGES

A major motivation behind the development of database systems is the desire to integrate the operational data of an organization and to provide controlled access to the data. Although integration and controlled access may imply centralization, this is not the intention. In fact, the development of computer networks promotes a decentralized mode of work. This decentralized approach mirrors the organizational structure of many companies, which are logically distributed into divisions, departments, projects, and so on, and physically distributed into offices, plants, factories, where each unit maintains its own operational data. The share ability of the data and the efficiency of data access should be improved by the development of a distributed database system that reflects this organizational structure, makes the data in all units accessible, and stores data proximate to the location where it is most frequently used.

Distributed DBMSs should help resolve the islands of information problem. Databases are sometimes regarded as electronic islands that are distinct and generally inaccessible places, like remote islands. This may be a result of geographical separation, incompatible computer architectures, incompatible communication protocols, and so on. Integrating the databases into a logical whole may prevent this way of thinking.

A **distributed database (DDB)** is a collection of multiple, logically interrelated databases distributed over a computer network. A **distributed database management system** (DDBMS) is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users. The distributed database (DDB) and distributed database management system (DDBMS) together is called **Distributed database system (DDBS).**

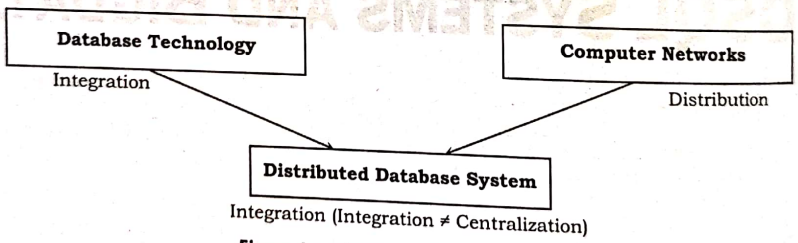

Figure 4.1: Distributed Database System

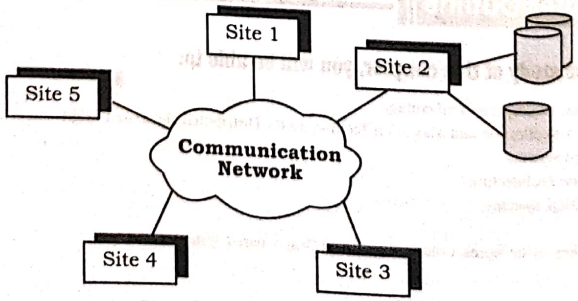

Figure 4.2: Centralized Environment

Scanned with CamScanner

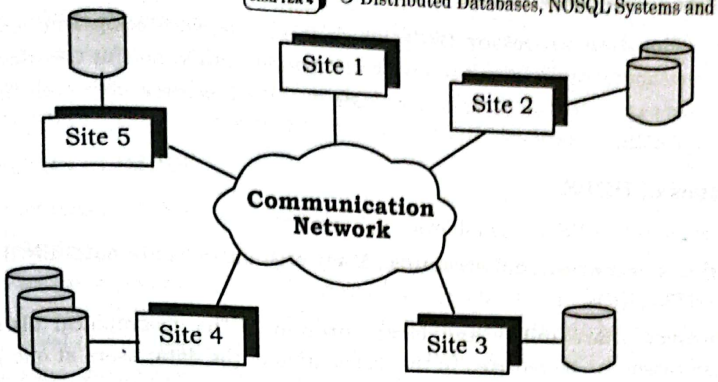

Figure 4.3: Distributed Environment

Characteristics of Distribute Database System

The Distributed Database System has following characteristics:

- A collection of logically related shared data;
- The data is split into a number of fragments;
- Fragments may be replicated;
- Fragments/replicas are allocated to sites;
- The sites are linked by a communications network;
- The data at each site is under the control of a DBMS;
- The DBMS at each site can handle local applications, autonomously;
- Each DBMS participates in at least one global application.

Components of Distributed Database System

The different components of DDBS are as follows:

- **Computer workstations or remote devices (sites or nodes)** that form the network system. The distributed database system must be independent of the computer system hardware.
- **Network hardware and software components** that reside in each workstation or device. The network components allow all sites to interact and exchange data. Because the components—computers, operating systems, network hardware, and so on—are likely to be supplied by different vendors, it is best to ensure that distributed database functions can be run on multiple platforms.
- **Communications media** that carry the data from one node to another. The DDBMS must be communications media-independent; that is, it must be able to support several types of communications media.
- **The transaction processor (TP)**, which is the software component found in each computer or device that requests data. The transaction processor receives and processes the application's data requests (remote and local). The TP is also known as the application processor (AP) or the transaction manager (TM).

Scanned with CamScanner

- **The data processor (DP)**, which is the software component residing on each computer or device that stores and retrieves data located at the site. The DP is also known as the data manager (DM). A data processor may even be a centralized DBMS.

Advantages of DDBS

The advantages of DDBS are as follows:

1. **Reflects organizational structure**: Many organizations are naturally distributed over several locations.
2. **Improved share-ability and local autonomy:** The geographical distribution of an organization can be reflected in the distribution of the data; users at one site can access data stored at other sites. Data can be placed at the site close to the users who normally use that data. In this way, users have local control of the data and they can consequently establish and enforce local policies regarding the use of this data. A global DBA is responsible for the entire system. Generally, part of this responsibility is devolved to the local level, so that the local DBA can manage the local DBMS.
3. **Improved availability:** In a centralized DBMS, a computer failure terminates the operations of the DBMS. However, a failure at one site of a DDBMS or a failure of a communication link making some sites inaccessible does not make the entire system inoperable. Distributed DBMSs are designed to continue to function despite such failures. If a single node fails, the system may be able to reroute the failed node's requests to another site.
4. **Improved reliability:** Because data may be replicated so that it exists at more than one site, the failure of a node or a communication link does not necessarily make the data inaccessible. Improved performance as the data is located near the site of "greatest demand," and given the inherent parallelism of distributed DBMSs, speed of database access may be better than that achievable from a remote centralized database. Furthermore, since each site handles only a part of the entire database, there may not be the same contention for CPU and I/O services as characterized by a centralized DBMS.
5. **Economics:** The potential cost saving occurs where databases are geographically remote and the applications require access to distributed data. In such cases, owing to the relative expense of data being transmitted across the network as opposed to the cost of local access, it may be much more economical to partition the application and perform the processing locally at each site.
6. **Modular growth:** In a distributed environment, it is much easier to handle expansion. New sites can be added to the network without affecting the operations of other sites. This flexibility allows an organization to expand relatively easily.
7. **Integration:** At the start of this section, we noted that integration was a key advantage of the DBMS approach, not centralization. The integration of legacy systems is one particular example that demonstrates how some organizations are forced to rely on distributed data processing to allow their legacy systems to coexist with their more modern systems. At the same time, no one package can provide all the functionality that an organization requires nowadays. Thus, it is important for organizations to be able to integrate software components from different vendors to meet their specific requirements.

Scanned with CamScanner

8. **Remaining competitive:** There are number of relatively recent developments that rely heavily on distributed database technology such as e-business, computer supported collaborative work and workflow management. Many enterprises have had to reorganize their businesses and use distributed database technology to remain competitive.

Disadvantages of DDBS

The disadvantages of DDBS are as follows:

1. **Complexity:** A distributed DBMS that hides the distributed nature from the user and provides an acceptable level of performance, reliability, and availability is inherently more complex than a centralized DBMS. The fact that data can be replicated also adds an extra level of complexity to the distributed DBMS. If the software does not handle data replication adequately, there will be degradation in availability, reliability, and performance compared with the centralized system, and the advantages we cited earlier will become disadvantages.

2. **Cost:** Increased complexity means that we can expect the procurement and maintenance costs for a DDBMS to be higher than those for a centralized DBMS. Furthermore, a distributed DBMS requires additional hardware to establish a network between sites. There are ongoing communication costs incurred with the use of this network. There are also additional labor costs to manage and maintain the local DBMSs and the underlying network.

3. **Security:** In a centralized system, access to the data can be easily controlled. However, in a distributed DBMS not only does access to replicated data have to be controlled in multiple locations, but the network itself has to be made secure. In the past, networks were regarded as an insecure communication medium. Although this is still partially true, significant developments have been made to make networks more secure.

4. **Integrity control more difficult**: Database integrity refers to the validity and consistency of stored data. Integrity is usually expressed in terms of constraints, which are consistency rules that the database is not permitted to violate. Enforcing integrity constraints generally requires access to a large amount of data that defines the constraint but that is not involved in the actual update operation itself. In a distributed DBMS, the communication and processing costs that are required to enforce integrity constraints may be prohibitive.

5. **Lack of standards**: Although distributed DBMSs depend on effective communication, we are only now starting to see the appearance of standard communication and data access protocols. This lack of standards has significantly limited the potential of distributed DBMSs. There are also no tools or methodologies to help users convert a centralized DBMS into a distributed DBMS.

6. **Lack of experience:** General-purpose distributed DBMSs have not been widely accepted, although many of the protocols and problems are well understood. Consequently, we do not yet have the same level of experience in industry as we have with centralized DBMSs. For a prospective adopter of this technology, this may be a significant deterrent.

7. **Database design more complex:** Besides the normal difficulties of designing a centralized database, the design of a distributed database has to take account of fragmentation of data, allocation of fragments to specific sites, and data replication.

Scanned with CamScanner

DISTRIBUTED DATABASE DESIGN

In this section, we discuss techniques that are used to break up the database into logical units, called fragments, which may be assigned for storage at the various nodes. We also discuss the use of data replication, which permits certain data to be stored in more than one site to increase availability and reliability; and the process of allocating fragments—or replicas of fragments—for storage at the various nodes. These techniques are used during the process of distributed database design. The information concerning data fragmentation, allocation, and replication is stored in a global directory that is accessed by the DDBS applications as needed.

Data Fragmentation

Fragmentation is the task of dividing a table into a set of smaller tables. The subsets of the table are called fragments. These fragments may be stored at different locations. Moreover, fragmentation increases parallelism and provides better disaster recovery. Fragmentation can be of three types:

- Vertical Fragmentation
- Horizontal Fragmentation
- Hybrid Fragmentation

Fragmentation should be done in a way so that the original table can be reconstructed from the fragments. This is needed so that the original table can be reconstructed from the fragments whenever required. This requirement is called "**reconstructiveness**".

Advantages of Fragmentation

- Since data is stored close to the site of usage, efficiency of the database system is increased.
- Local query optimization techniques are sufficient for most queries since data is locally available.
- Since irrelevant data is not available at the sites, security and privacy of the database system can be maintained.

Disadvantages of Fragmentation

- When data from different fragments are required, the access speeds may be very high.
- In case of recursive fragmentations, the job of reconstruction will need expensive techniques.
- Lack of back-up copies of data in different sites may render the database ineffective in case of failure of a site.

Vertical Fragmentation

In vertical fragmentation, the fields or columns of a table are grouped into fragments. In order to maintain re-constructiveness, each fragment should contain the primary key field(s) of the table. Vertical fragmentation can be used to enforce privacy of data.

Scanned with CamScanner

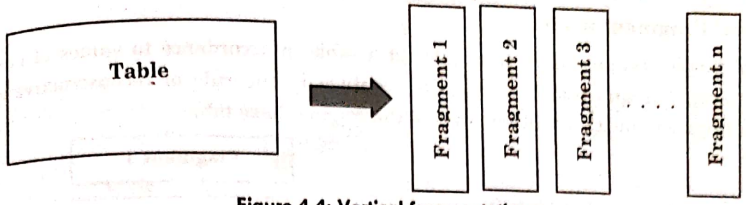

Figure 4.4: Vertical fragmentation

Example 4.1: Let us consider that a college management system keeps records of all registered students in a Student table having the following schema.

Student

Stu_id	Stu_name	Stu_address	Dept_id
10	Maya	Palpa	1
11	Abin	Ktm	2
12	Arav	Ktm	1
13	Ashna	Palpa	3
14	Anju	Pokhara	4
15	Manish	Banepa	2
16	Pinky	Ktm	1

Figure 4.5: Student table before fragmentation

Now, the address details are maintained in the admin section. In this case, the designer will fragment the database as follows:

```
CREATE TABLE Std_address AS
SELECT Stu_id, Stu_address
FROM Student;
```

By executing above query, we get the following result:

Std_address

Stu_id	Stu_address
10	Palpa
11	Ktm
12	Ktm
13	Palpa
14	Pokhara
15	Banepa
16	Ktm

Figure 4.6: Std_address Table (Vertical Data Fragment of Student Table)

Scanned with CamScanner

Horizontal Fragmentation

Horizontal fragmentation groups the tuples of a table in accordance to values of one or more fields. Horizontal fragmentation should also confirm to the rule of reconstructiveness. Each horizontal fragment must have all columns of the original base table.

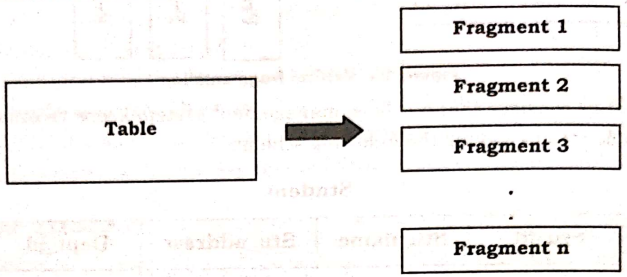

Figure 4.7: Horizontal fragmentation

Example 4.2: In the student schema which is shown in figure 7.6, if the details of all students of department 1 need to be maintained at the respective faculty, then the designer will horizontally fragment the database as follows:

```
CREATE TABLE Department AS
SELECT * FROM Student
WHERE Dept_id=1;
```

By executing above query, we get the following result:

Department

Stu_id	Stu_name	Stu_address	Dept_id
10	Maya	Palpa	1
12	Arav	Ktm	1
16	Pinky	Ktm	1

Figure 4.8: Department Table (Horizontal Fragment of Student Table)

Hybrid Fragmentation

In hybrid fragmentation, a combination of horizontal and vertical fragmentation techniques is used. This is the most flexible fragmentation technique since it generates fragments with minimal extraneous information. However, reconstruction of the original table is often an expensive task. Hybrid fragmentation can be done in two alternative ways:

1. At first, generate a set of horizontal fragments; then generate vertical fragments from one or more of the horizontal fragments.
2. At first, generate a set of vertical fragments; then generate horizontal fragments from one or more of the vertical fragments.

Scanned with CamScanner

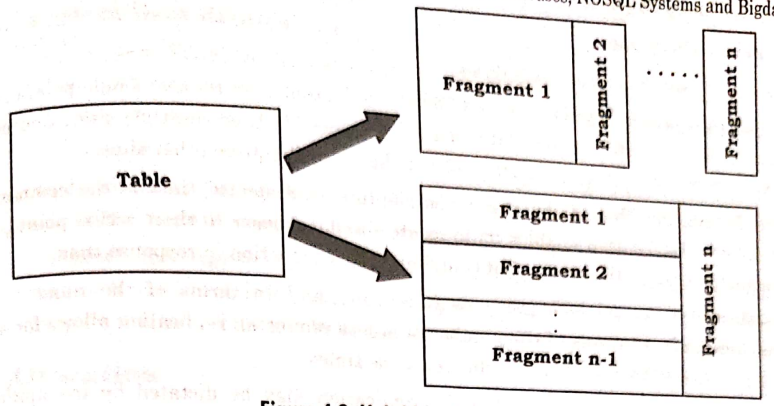

Figure 4.9: Hybrid fragmentation

Example 4.3:

```
CREATE TABLE Hybrid AS
SELECT  Stu_id, Stu_name FROM Student
WHERE Stu_id=12;
```

By executing above query, we get the following result:

Hybrid

Stu_id	Stu_name
12	Arav

Figure 4.10: Hybrid Table (hybrid fragment of Student Table)

Data Replication

Data Replication is the process of generating and reproducing multiple copies of data at one or more sites. Replication is an important mechanism, because it enables organizations to provide users with access to current data where and when they need it. It is intended to increase the fault tolerance of a system such that if one database fails another can continue to serve queries or update requests. Replication is sometimes described using the publishing industry metaphor of publishers, distributors, and subscribers.

- **Publisher:** A DBMS that makes data available to other locations through replication. The publisher can have one or more publications (made up of one or more articles), each defining a logically related set of objects and data to replicate.
- **Distributor:** A DBMS that stores replication data and metadata about the publication and in some cases acts as a queue for data moving from the publisher to the subscribers. A DBMS can act as both the publisher and the distributor.
- **Subscriber**: A DBMS that receives replicated data. A subscriber can receive data from multiple publishers and publications. Depending on the type of replication chosen, the subscriber can also pass data changes back to the publisher or republish the data to other subscribers.

Scanned with CamScanner

Replication Purpose

The purpose of replications are as follows:

1. **System availability:** Distributed Database System may remove single points of failure by replicating data, so that data items are accessible from multiple sites. Consequently, even when some sites are down, data may be accessible from other sites.
2. **Performance:** One of the major contributors to response time is the communication overhead. Replication enables us to locate the data closer to their access points, thereby localizing most of the access that contributes to a reduction in response time.
3. **Scalability:** As systems grow geographically and in terms of the number of sites (consequently, in terms of the number of access requests), replication allows for a way to support this growth with acceptable response times.
4. **Application requirements:** Finally, replication may be dictated by the applications, which may wish to maintain multiple data copies as part of their operational specifications.

Challenges in Replications

1. **Placement of replicas:** The major challenge in replication is where to put the replicas. There are three places to put replicas.
   - **Permanent replicas:** permanent replicas consist of cluster of servers that may be geographically dispersed.
   - **Server initiated replicas:** Server initiated caches include placing replicas in the hosting servers and server caches.
   - **Client initiated replicas:** Client initiated replicas include web browsers cache.
2. **Propagation of updates among replicas:** The net challenge is to how to propagate the updates in one replica among all the replicas efficiently and faster as possible.
   - **Push based propagation:** A replica in which update occurs pushes the updates to all other replicas.
   - **Pull based propagation:** A replicas requests another replica to send the newest data it has.
3. **Lack of consistency:** If a copy if modified, the copy becomes inconsistent from the rest of copies. It takes some time for all the copies to be consistent.

Advantages of Data Replication

- **Reliability:** In case of failure of any site, the database system continues to work since a copy is available at another site(s).
- **Reduction in Network Load:** Since local copies of data are available, query processing can be done with reduced network usage, particularly during prime hours. Data updating can be done at non-prime hours.
- **Quicker Response:** Availability of local copies of data ensures quick query processing and consequently quick response time.
- **Simpler Transactions:** Transactions require a smaller number of joins of tables located at different sites and minimal coordination across the network. Thus, they become simpler in nature.

Scanned with CamScanner

Disadvantages of Data Replication

- **Increased Storage Requirements:** Maintaining multiple copies of data is associated with increased storage costs. The storage space required is in multiples of the storage required for a centralized system.
- **Increased Cost and Complexity of Data Updating:** Each time a data item is updated, the update needs to be reflected in all the copies of the data at the different sites. This requires complex synchronization techniques and protocols.
- **Undesirable Application – Database coupling:** If complex update mechanisms are not used, removing data inconsistency requires complex co-ordination at application level. This results in undesirable application – database coupling.

Data Allocation

Each fragment or each copy of a fragment is stored at a particular site in the distributed system with "optimal" distribution. This process is called data distribution (**or data allocation**). The choice of sites and the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site.

Example 4.4: If high availability is required, transactions can be submitted at any site, and most transactions are retrieval only, a fully replicated database is a good choice. However, if certain transactions that access particular parts of the database are mostly submitted at a particular site, the corresponding set of fragments can be allocated at that site only. Data that is accessed at multiple sites can be replicated at those sites. If many updates are performed, it may be useful to limit replication. Finding an optimal or even a good solution to distributed data allocation is a complex optimization problem.

There are four alternative strategies regarding the placement of data: *centralized, fragmented, complete replication,* and *selective replication.*

Centralized

This strategy consists of a single database and DBMS stored at one site with users distributed across the network. Locality of reference is at its lowest as all sites, except the central site, have to use the network for all data accesses. This also means that communication costs are high. Reliability and availability are low, as a failure of the central site results in the loss of the entire database system.

Fragmented (or Partitioned)

This strategy partitions the database into disjoint fragments, with each fragment assigned to one site. If data items are located at the site where they are used most frequently, locality of reference is high. As there is no replication, storage costs are low; similarly, reliability and availability are low, although they are higher than in the centralized case, as the failure of a site results in the loss of only that site's data. Performance should be good and communications costs low if the distribution is designed properly.

Complete Replication

This strategy consists of maintaining a complete copy of the database at each site. Therefore, locality of reference, reliability and availability, and performance are maximized. However,

Scanned with CamScanner

storage costs and communication costs for updates are the most expensive. To overcome some of these problems, snapshots are sometimes used. A **snapshot** is a copy of the data at a given time. The copies are updated periodically—for example, hourly or weekly—so they may not be always up to date. Snapshots are also sometimes used to implement views in a distributed database to improve the time it takes to perform a database operation on a view.

Selective Replication

This strategy is a combination of fragmentation, replication, and centralization. Some data items are fragmented to achieve high locality of reference, and others that are used at many sites and are not frequently updated are replicated; otherwise, the data items are centralized. The objective of this strategy is to have all the advantages of the other approaches but none of the disadvantages. This is the most commonly used strategy, because of its flexibility.

TYPES OF DISTRIBUTED DATABASE SYSTEMS

Distributed databases can be broadly classified into **homogeneous** and **heterogeneous** distributed database environments.

Homogeneous DDBS

In a homogeneous system, all sites use the same DBMS product. Homogeneous systems are much easier to design and manage. This approach provides incremental growth, making the addition of a new site to the DDBMS easy, and allows increased performance by exploiting the parallel processing capability of multiple sites.

Example 4.5: Consider that we have three departments using Oracle 19c for DBMS. If some changes are made in one department, then, it would update the other department also.

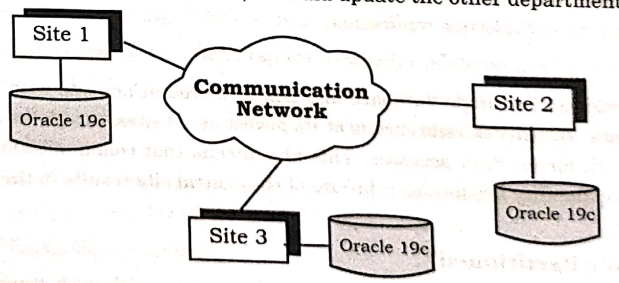

Figure 4.11: Homogeneous distributed system

Types of Homogeneous Distributed Database System

There are two types of homogeneous distributed database system:

- **Autonomous:** Each database is independent that functions on its own. They are integrated by a controlling application and use message passing to share data updates.
- **Non-autonomous:** Data is distributed across the homogeneous nodes and a central or master DBMS co-ordinates data updates across the sites.

Scanned with CamScanner

Heterogeneous DDBS

In a heterogeneous system, sites may run different DBMS products, which need not be based on the same underlying data model, and so the system may be composed of relational, network, hierarchical, and object-oriented DBMSs. Heterogeneous systems usually result when individual sites have implemented their own databases and integration is considered at a later stage. In a heterogeneous system, translations are required to allow communication between different DBMSs. To provide DBMS transparency, users must be able to make requests in the language of the DBMS at their local site. The system then has the task of locating the data and performing any necessary translation. Data may be required from another site that may have:

- Different hardware
- Different DBMS products
- Different hardware and different DBMS products.

Example 4.6: In the following diagram, different DBMS software are accessible to each other using generic connectivity (ODBC and JDBC).

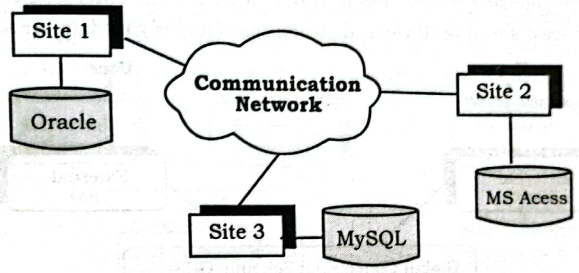

Figure 4.12: Heterogeneous distributed system

If the hardware is different but the DBMS products are the same, the translation is straightforward, involving the change of codes and word lengths. If the DBMS products are different, the translation is complicated involving the mapping of data structures in one data model to the equivalent data structures in another data model.

Example 4.7: If relations in the relational data model are mapped to records and sets in the network model. It is also necessary to translate the query language used (for example, SQL SELECT statements are mapped to the network FIND and GET statements). If both the hardware and software are different, then both these types of translation are required. This makes the processing extremely complex.

Types of Heterogeneous Distributed Databases System

- **Federated:** The heterogeneous database systems are independent in nature and integrated together so that they function as a single database system.
- **Un-federated:** The database systems employ a central coordinating module through which the databases are accessed.

Scanned with CamScanner

DISTRIBUTED DATABASE ARCHITECTURES

DDBS architecture are generally developed depending on three parameters:

- **Distribution:** It states the physical distribution of data across the different sites i.e., whether the components of the system are located on the same machine or not.
- **Autonomy:** It indicates the distribution of control of the database system and the degree to which each constituent DBMS can operate independently. Autonomy is a function of a number of factors such as whether the component systems (i.e., individual DBMSs) exchange information, whether they can independently execute transactions, and whether one is allowed to modify them.
- **Heterogeneity:** It refers to the uniformity or dissimilarity of the data models, system components and databases.

General Architecture of Pure Distributed Databases

Here, we discuss both the logical and component architectural models of a DDBS. In figure 4.13, which describes the generic schema architecture of a DDBS, the enterprise is presented with a consistent, unified view showing the logical structures of underlying data across all nodes.

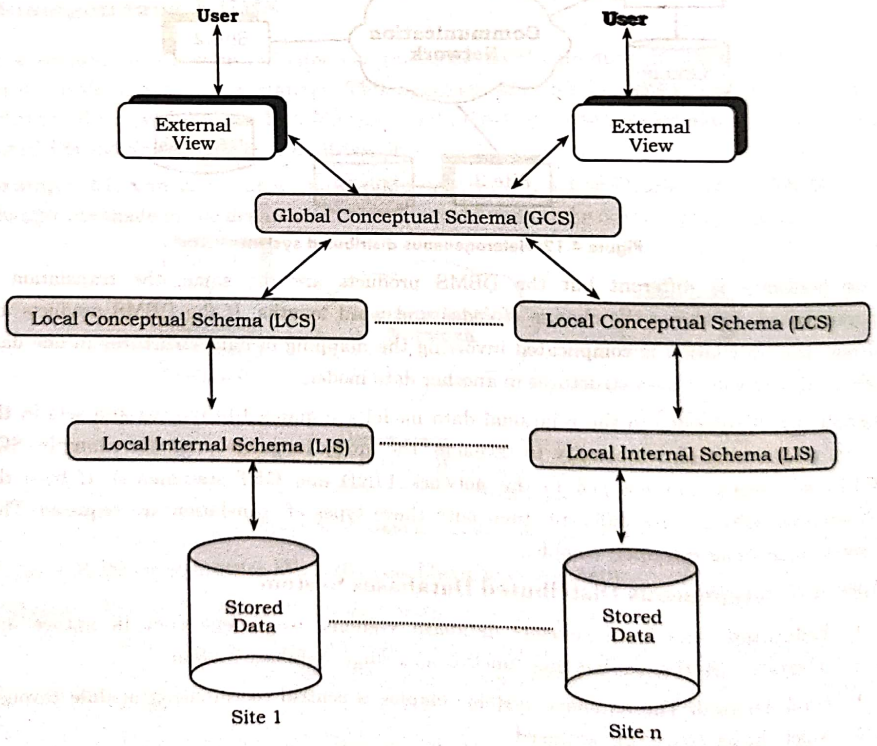

Figure 4.13: Schema architecture of distributed database

Scanned with CamScanner

This architecture generally has four levels of schemas:

- External View or Schema (EV or ES): Depicts user view of data.
- Global Conceptual Schema (GCS): Depicts the global logical view of data, which provides network transparency.
- Local Conceptual Schema (LCS): Depicts logical data organization at each site.
- Local Internal Schema (LIS): Depicts physical data organization at each site.

Federated Database Schema Architecture

Typical five-level schema architecture to support global applications in the FDBS environment is shown in figure 4.14. In this architecture, the **local schema** is the conceptual schema (full database definition) of a component database, and the **component schema** is derived by translating the local schema into a canonical data model or common data model (CDM) for the FDBS. Schema translation from the local schema to the component schema is accompanied by generating mappings to transform commands on a component schema into commands on the corresponding local schema. The **export schema** represents the subset of a component schema that is available to the FDBS. The **federated schema** is the global schema or view, which is the result of integrating all the shareable export schemas. The **external schemas** define the schema for a user group or an application, as in the three-level schema architecture.

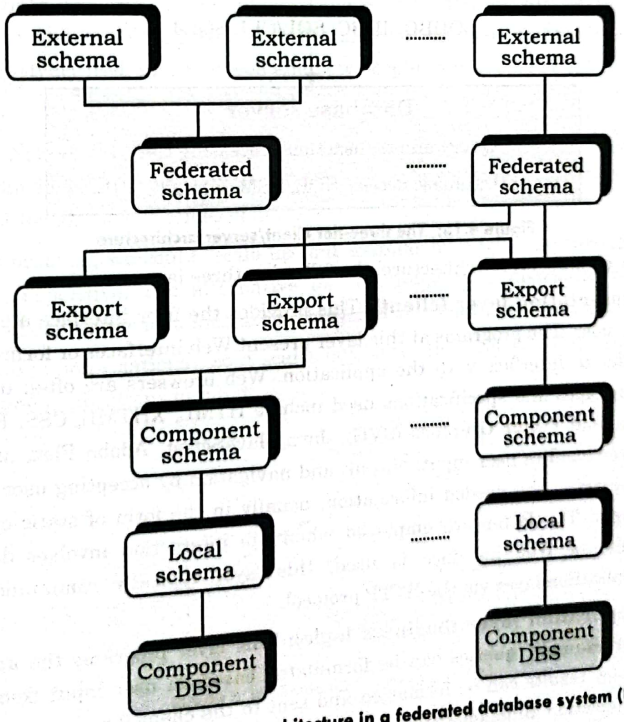

Figure 4.14: The five-level schema architecture in a federated database system (FDBS).

Scanned with CamScanner

Overview of Three -Tier Client/Server Architecture

Full-scale DDBMSs have not been developed to support all the types of functionalities that we have known so far. Instead, distributed database applications are being developed in the context of the client/server architectures. It is now more common to use a three-tier architecture rather than a two-tier architecture, particularly in Web applications. This architecture is illustrated in figure 4.15.

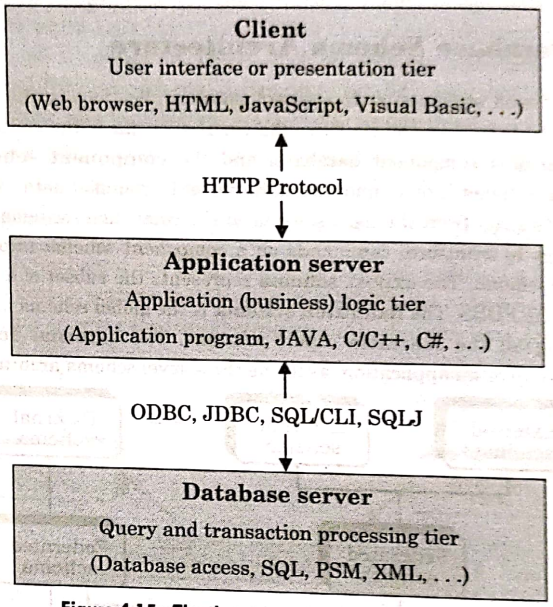

Figure 4.15: The three-tier client/server architecture

In the three-tier client/server architecture, the following three layers exist:

- **Presentation layer (client):** This provides the user interface and interacts with the user. The programs at this layer present Web interfaces or forms to the client in order to interface with the application. Web browsers are often utilized, and the languages and specifications used include HTML, XHTML, CSS, Flash, MathML, Scalable Vector Graphics (SVG), Java, JavaScript, Adobe Flex, and others. This layer handles user input, output, and navigation by accepting user commands and displaying the needed information, usually in the form of static or dynamic Web pages. The latter are employed when the interaction involves database access. When a Web interface is used, this layer typically communicates with the application layer via the HTTP protocol.
- **Application layer (business logic):** This layer programs the application logic. For example, queries can be formulated based on user input from the client, or query results can be formatted and sent to the client for presentation. Additional application functionality can be handled at this layer, such as security checks.

Scanned with CamScanner

identity verification, and other functions. The application layer can interact with one or more databases or data sources as needed by connecting to the database using ODBC, JDBC, SQL/CLI, or other database access techniques.

- **Database server:** This layer handles query and update requests from the application layer, processes the requests, and sends the results. Usually, SQL is used to access the database if it is relational or object-relational, and stored database procedures may also be invoked. Query results (and queries) may be formatted into XML when transmitted between the application server and the database server.

INTRODUCTION TO NOSQL SYSTEMS

A NoSQL database, which stands for "non SQL" or "non-relational," is a database that allows for data storage and retrieval. It avoids joins, and is easy to scale. The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs. NoSQL is used for big data and real-time web apps. For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.

Traditional RDBMS uses SQL syntax to store and retrieve data for further insights. Instead, a NoSQL database system encompasses a wide range of database technologies that can store structured, semi-structured, unstructured and polymorphic data.

Why NoSQL?

The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data. The system response time becomes slow when you use RDBMS for massive volumes of data.

To resolve this problem, we could "scale up" our systems by upgrading our existing hardware. This process is expensive. The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as "scaling out."

NoSQL database is non-relational, so it scales out better than relational databases as they are designed with web applications in mind.

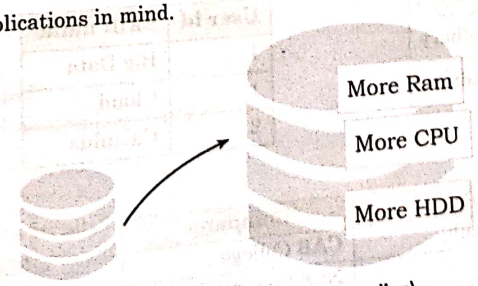

Figure 4.16: Scale up (Vertical scaling)

Scanned with CamScanner

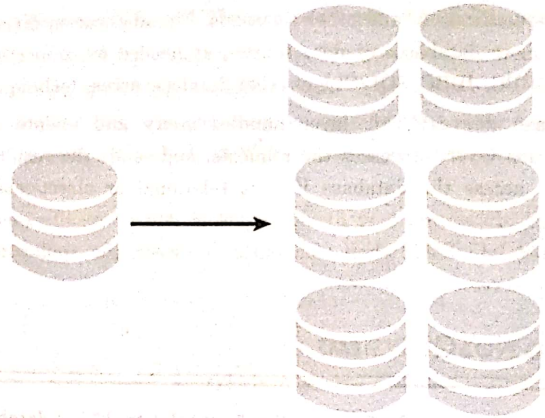

Figure 4.16: Scale-out (Horizontals Scaling)

Differences between RDBMS and NoSQL

RDBMS is called relational databases while NoSQL is called a distributed database. They do not have any relations ey, there is a proper method in NoSQL to use unstructured data. RDBMS is scalable vertically and NoSQL is scalable horizontally. Hence in RDBMS, servers have to be added and powbetween any of the databases. When RDBMS uses structured data to identify the primary ker has to be increased. This makes scalability an RDBMS expensive. While in NoSQL, we just need to add more machines and this does not make the database expensive.

Maintenance of RDBMS is expensive as manpower is needed to manage the servers added in the database. NoSQL is mostly automatic and does some repairs on its own. Data distribution and administration is less in NoSQL.

Example: Lets take data stored in RDBMS as,

User

Uid	Fname	Lname
1	Indra	Chaudhary
2	Chokraj	Dawadi

Skill

User id	Skill name
1	Big Data
1	Cloud
2	Calculus

Experience

User id	Role	Company
1	Full time faculty	CAB College
2	Principal	New Summit College
2	Visiting faculty member	KMC

Reading this profile would require the application to read 4 rows from three tables,

Scanned with CamScanner

Uid	Fname	Lname	Skill name	Role	Company
1	Indra	Chaudhary	Big Data	Full time faculty	CAB College
1	Indra	Chaudhary	Cloud	Full time faculty	CAB College
2	Chokraj	Dawadi	Calculus	Principal	New Summit College
2	Chokraj	Dawadi	Calculus	Visiting faculty	KMC

In NoSQL we can express above three tables data in the form of JSON as below,

```
{
    User: [
    {
        Uid:1
        First name: "Indra"
        Last name: "Chaudhary"
    }
    {
        Uid: 2
        First name: "Indra"
        Last name: "Chaudhary"
    }
    ],
    Skill: [ "Big Data", "Cloud", "Calculus"],
    Experience: [
    {
      Role: "Full time faculty"
      Company: "CAB College"
    }
    {
      Role: "Principal"
      Company: "New Summit College"
    }
    {
      Role: "Visiting faculty"
      Company: "KMC"
    }
    ]
}
```

Scanned with CamScanner

Major differences between them are tabulated below,

RDBMS	NoSQL
Users know RDBMS well as it is old and many organizations use this database for the proper format of data.	This is relatively new and experts in NoSQL are less as this database is evolving day by day.
User interface tools to access data is available in the market so that users can try with all the schema to the RDBMS infrastructure. This helps to interact with the data well and users will understand the data in a better manner.	User interface tools to access and manipulate data in NoSQL is very less and hence users do not have many options to interact with data.
RDBMS scalability and performance faces some issues if the data is huge. Servers may not run properly with the available load and this leads to performance issues.	It works well with high loads. Scalability is very good in NoSQL. This makes the performance of the database better when compared with RDBMS. A huge amount of data could be easily handled by users.
Multiple tables can be joined easily in RDBMS and this does not cause any latency in the working of the database. The primary key helps in this case.	Multiple tables cannot be joined in NoSQL as it is not an easy task for the database and does not work well with the performance of the data.
The availability of the database depends on the server performance and it is mostly available whenever the database is opened. The data provided is consistent and does not confuse users.	Though the databases are readily available, consistency provided in some databases is less. This results in the performance of the database and users should check the availability often.
Data analysis and querying can be done easily with RDBMS even though the queries are complex. Slicing and dicing can be done with the available data to make the proper analysis of the data given.	Data analysis is done also in NoSQL but it works well with real-time data analytics. Reports are not done in the database but if the application has to be built, then NoSQL is a solution for the same.
Documents cannot be stored in RDBMS because data in the database should be structured and in a proper format to create identifiers.	Documents can be stored in the NoSQL database as this is unstructured and not in rows and columns format.
Partitions cannot be created in the database. Key-value pairs are needed to identify the data in a particular format specified in the schema database.	Partitions can be created in the database easily and key-value pairs are not needed to identify the data in the source. Software as a service can be integrated with NoSQL.
Example: MySQL, Oracle, SQL Server etc.	**Example**: IBM Domino, Oracle NoSQL, Apache HBase etc.

Scanned with CamScanner

Types of NoSQL Databases

Several different varieties of NoSQL databases have been created to support specific needs and use cases. These fall into four main categories:

Document Databases

Document databases, like JSON (JavaScript Object Notation) objects, store data in documents. Each document has a set of field and value pairs. The values might be of many sorts, such as texts, integers, Booleans, arrays, or objects, and their structures are usually aligned with the objects that developers interact with within code.

Document databases are useful for a broad number of use cases and may be utilized as a general-purpose database due to their variety of field value types and strong query languages. They can expand out horizontally to accommodate enormous data volumes.

Key-Value Databases

Key-value databases are a simpler form of database that has keys and values for each item. Learning how to query for a certain key-value pair is usually straightforward because a value can only be accessed by referencing its key.

Key-value databases are ideal for situations in which you need to store a significant quantity of data but don't need to access it using complicated queries. Caching and saving user preferences are two common use cases. Popular key-value databases include Redis and DynamoDB.

Wide-Column stores

Wide-column NoSQL databases store data in tables with rows and columns similar to RDBMS, but names and formats of columns can vary from row to row across the table. Wide-column databases group columns of related data together. A query can retrieve related data in a single operation because only the columns associated with the query are retrieved. In an RDBMS, the data would be in different rows stored in different places on disk, requiring multiple disk operations for retrieval.

Graph Databases

Data is stored in nodes and edges in graph databases. Edges hold information about the relationships between nodes, whereas nodes store information about people, locations, and objects. A graph database uses graph structures to store, map, and query relationships. They provide index-free adjacency, so that adjacent elements are linked together without using an index.

Advantages of NoSQL

NoSQL databases offer enterprises important advantages over traditional RDBMS, including:

Scalability

NoSQL databases use a horizontal scale-out methodology that makes it easy to add or reduce capacity quickly and non-disruptively with commodity hardware. This eliminates the tremendous cost and complexity of manual sharding that is necessary when attempting to scale RDBMS.

Scanned with CamScanner

Performance

By simply adding commodity resources, enterprises can increase performance with NoSQL databases. This enables organizations to continue to deliver reliably fast user experiences with a predictable return on investment for adding resources—again, without the overhead associated with manual sharding.

High Availability

NoSQL databases are generally designed to ensure high availability and avoid the complexity that comes with a typical RDBMS architecture that relies on primary and secondary nodes. Some "distributed" NoSQL databases use a masterless architecture that automatically distributes data equally among multiple resources so that the application remains available for both read and write operations even when one node fails.

Global Availability

By automatically replicating data across multiple servers, data centers, or cloud resources, distributed NoSQL databases can minimize latency and ensure a consistent application experience wherever users are located. An added benefit is a significantly reduced database management burden from manual RDBMS configuration, freeing operations teams to focus on other business priorities.

Flexible Data Modeling

NoSQL offers the ability to implement flexible and fluid data models. Application developers can leverage the data types and query options that are the most natural fit to the specific application use case rather than those that fit the database schema. The result is a simpler interaction between the application and the database and faster, more agile development.

There is less management

Despite huge advancements in our DBMS domain over the years, relational databases rely heavily on database administrators, also known as DBAs. On the other hand, NoSQL databases are typically built from the ground up to eliminate needless management, automated data distribution, and simpler data models, resulting in lower administration and performance demands.

THE CAP THEOREM

CAP stands for Consistency, Availability, and Partitioning. It is very important to understand the limitations of NoSQL database. NoSQL cannot provide consistency and high availability together. This was first expressed by Eric Brewer in CAP Theorem. CAP theorem or Eric Brewers theorem states that we can only achieve at most two out of three guarantees for a database: Consistency, Availability and Partition Tolerance.

Consistency

Consistency is all about data consistency, or in other words, making sure that within a distributed environment, every node of the database has exactly the same information at any given time. Imagine having two nodes with purchase orders from your ecommerce site. If there is no data

Scanned with CamScanner

consistency amongst them and they're acting as a unique cluster, the moment your client app queries the outdated node, it might show you missing transactions. And if the code is not only showing, but also making calculations based on that data, the results might be disastrous.

So consistency is definitely an important characteristic of any distributed NoSQL database. However, not all of them can provide it. So what do they do instead? They go for something called "eventual consistency." Meaning that while at one point the cluster may not be consistent, it will eventually be so. This helps in making sure that you don't get the types of problems I mentioned before.

Availability

Availability stands for "high availability" or in other words the ability of the database to always be available, no matter what happens. This is not the same and should not be confused with "fault tolerance" however. A highly available database is usually one that has replicas in multiple geographical zones, that way if there is a big network outage, it'll still be accessible through one of its other replicas. For example, a system that's only installed and working on one of our servers can't be highly available because the moment that server fails, we'll lose our database.

Partitioning

Partitioning stands for "partitioning tolerance" or in other words, having the ability to support broken links within the cluster the database is distributed in. Think about a graph representing your database cluster. You have multiple nodes sharing data and working wonderfully and suddenly there is a problem and a section of that cluster fails. If the database is "partition tolerant" it'll still work despite the sudden lack of some of its nodes.

The CAP theorem categorizes systems into three categories:

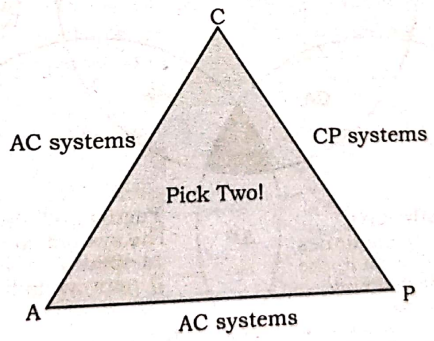

Figure 4.17: Visual representation of the CAP diagram

CP (Consistent and Partition Tolerant) 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.

Partition refers to a communication break between nodes within a distributed system. Meaning, if a node cannot receive any messages from another node in the system, there is a partition

Scanned with CamScanner

between the two nodes. Partition could have been because of network failure, server crash, or any other reason.

AP (Available and Partition Tolerant) 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.

CA (Consistent and Available) database: A CA delivers consistency and availability in the absence of any network partition. Often a single node's DB servers are categorized as CA systems. Single node DB servers do not need to deal with partition tolerance and are thus considered CA systems.

In any networked shared-data systems or distributed systems partition tolerance is a must. Network partitions and dropped messages are a fact of life and must be handled appropriately. Consequently, system designers must choose between consistency and availability.

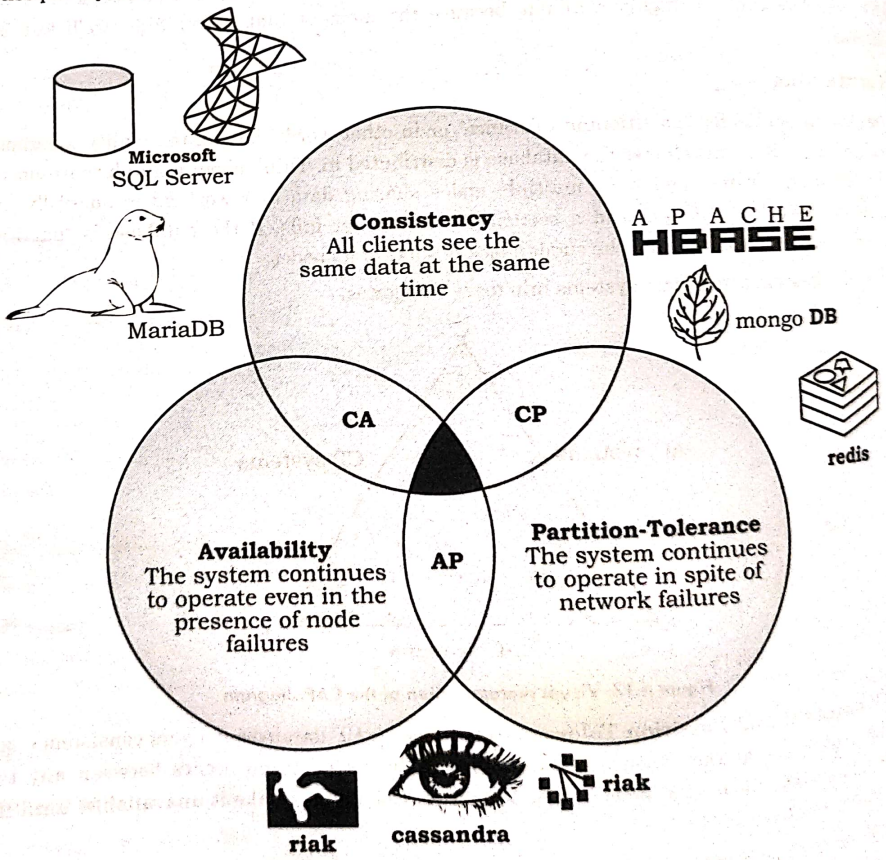

Figure 4.18: Classification of different databases based on the CAP theorem

Scanned with CamScanner

Disadvantages of NoSQL

Major disadvantages of NoSQL are described below:

- No standardization rules
- Limited query capabilities
- RDBMS databases and tools are comparatively mature
- It does not offer any traditional database capabilities, like consistency when multiple transactions are performed simultaneously.
- When the volume of data increases it is difficult to maintain unique values as keys become difficult
- Doesn't work as well with relational data
- The learning curve is stiff for new developers
- Open source options so not so popular for enterprises.

DOCUMENT-BASED, KEY-VALUE STORES, COLUMN-BASED, AND GRAPH-BASED SYSTEMS

Key-Value Database

Key-value databases are the simplest type of NoSQL database. Thanks to their simplicity, they are also the most scalable, allowing horizontal scaling of large amounts of data.

These NoSQL databases have a dictionary data structure that consists of a set of objects that represent fields of data. Each object is assigned a unique key. To retrieve data stored in a particular object, you need to use a specific key. In turn, you get the value (i.e. data) assigned to the key. This value can be a number, a string, or even another set of key-value pairs.

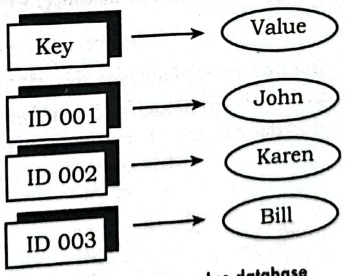

Figure 4.19: Key value database

Unlike traditional relational databases, key-value databases do not require a predefined structure. They offer more flexibility when storing data and have faster performance. Without having to rely on placeholders, key-value databases are a lighter solution as they require fewer resources.

Such functionalities are suitable for large databases that deal with simple data. Therefore, they are commonly used for caching, storing, and managing user sessions, ad servicing, and recommendations.

Scanned with CamScanner

Document Database

A document database is a type of NoSQL database that consists of sets of key-value pairs stored into a document. These documents are basic units of data which you can also group into collections (databases) based on their functionality.

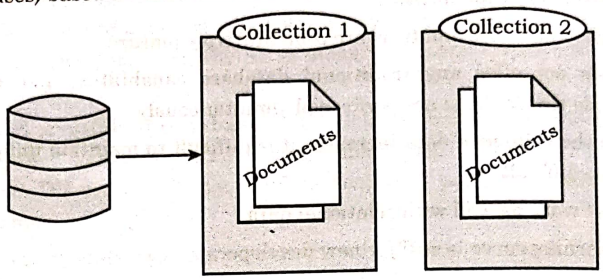

Figure 4.20: Document database

Being a NoSQL database, you can easily store data without implementing a schema. You can transfer the object model directly into a document using several different formats. The most commonly used are JSON, BSON, and XML.

Here is an example of a simple document in JSON format that consists of three key-value pairs:

```
{
"ID" : "001",
"Name" : "John",
"Grade" : "Senior",
}
```

What's more, you can also use nested queries in such formats, providing easier data distribution across multiple disks and enhanced performance. For instance, we can add a nested value string to the document above:

```
{
 "ID" : "001",
 "Name" : "John",
 "Grade" : "Senior",
 "Classes" : {
    "Class1" : "English"
    "Class2" : "Geometry"
    "Class3" : "History"
  }
}
```

Due to their structure, document databases are optimal for use cases that require flexibility and fast, continual development. For example, you can use them for managing user profiles, which differ according to the information provided. Its schema-less structure allows you to have different attributes and values. **Examples** of NoSQL document databases include MongoDB, CouchDB, Elasticsearch, and others.

Scanned with CamScanner

Wide-Column Database

Wide-column stores are another type of NoSQL database. In them, data is stored and grouped into separately stored columns instead of rows. Such databases organize information into columns that function similarly to tables in relational databases.

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

However, unlike traditional databases, wide-column databases are highly flexible. They have no predefined keys nor column names. Their schema-free characteristic allows variation of column names even within the same table, as well as adding columns in real-time.

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. **Examples** of popular wide-column databases include Apache Cassandra, HBase, and CosmoDB.

Graph Database

Graph databases use flexible graphical representation to manage data.

These graphs consist of two elements:

- Nodes (for storing data entities)
- Edges (for storing the relationship between entities)

These relationships between entities allow data in the store to be linked together directly and, in many cases, retrieved with one operation. Nodes and edges have defined properties, and by using these properties, you can query data easily.

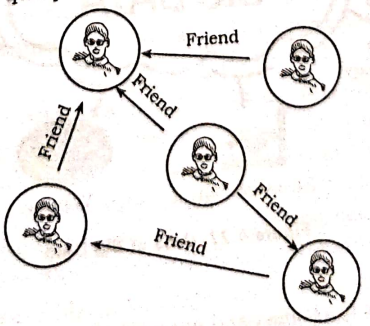

Figure 4.21: Graph database

Scanned with CamScanner

Since this type of data-storing is quite specific, it is not a commonly used NoSQL database. However, there are certain use cases in which having graphical representations is the best solution. For instance, social networks often use graphs to store information about how their users are linked. OrientDB, RedisGraph, and Neo4j are just some examples of graph databases you should consider using.

BIG DATA

Big Data is a collection of data that is huge in volume, yet growing exponentially with time. It is a data with so large size and complexity that none of traditional data management tools can store it or process it efficiently. Big data is also a data but with huge size.

Big data can be categorized as unstructured or structured. Structured data consists of information already managed by the organization in databases and spreadsheets; it is frequently numeric in nature. Unstructured data is information that is unorganized and does not fall into a predetermined model or format. It includes data gathered from social media sources, which help institutions gather information on customer needs.

Big data can be collected from publicly shared comments on social networks and websites, voluntarily gathered from personal electronics and apps, through questionnaires, product purchases, and electronic check-ins. The presence of sensors and other inputs in smart devices allows for data to be gathered across a broad spectrum of situations and circumstances.

Big data is most often stored in computer databases and is analyzed using software specifically designed to handle large, complex data sets. Many software-as-a-service (SaaS) companies specialize in managing this type of complex data.

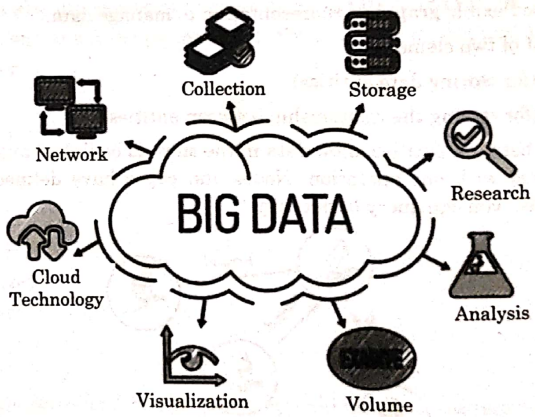

Figure 4.22: Uses of Big Data

Big Data Architecture

Big data architecture refers to the logical and physical structure that dictates how high volumes of data are ingested, processed, stored, managed, and accessed.

Scanned with CamScanner

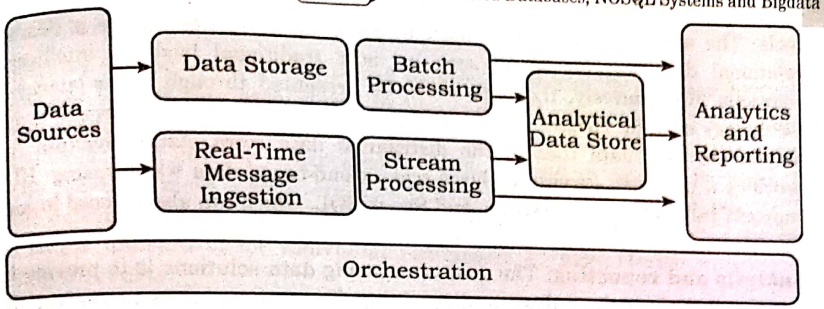

Figure 4.23: Big data architecture

Most big data architectures include some or all of the following components:

1. **Data sources:** All big data solutions start with one or more data sources. Examples include:
   - Application data stores, such as relational databases.
   - Static files produced by applications, such as web server log files.
   - Real-time data sources, such as IoT devices.
2. **Data storage:** Data for batch processing operations is typically stored in a distributed file store that can hold high volumes of large files in various formats. This kind of store is often called a data lake. Options for implementing this storage include Azure Data Lake Store or blob containers in Azure Storage.
3. **Batch processing**: Because the data sets are so large, often a big data solution must process data files using long-running batch jobs to filter, aggregate, and otherwise prepare the data for analysis. Usually these jobs involve reading source files, processing them, and writing the output to new files. Options include running U-SQL jobs in Azure Data Lake Analytics, using Hive, Pig, or custom Map/Reduce jobs in an HDInsight Hadoop cluster, or using Java, Scala, or Python programs in an HDInsight Spark cluster.
4. **Real-time message ingestion:** If the solution includes real-time sources, the architecture must include a way to capture and store real-time messages for stream processing. This might be a simple data store, where incoming messages are dropped into a folder for processing. However, many solutions need a message ingestion store to act as a buffer for messages, and to support scale-out processing, reliable delivery, and other message queuing semantics. Options include Azure Event Hubs, Azure IoT Hubs, and Kafka.
5. **Stream processing:** After capturing real-time messages, the solution must process them by filtering, aggregating, and otherwise preparing the data for analysis. The processed stream data is then written to an output sink. Azure Stream Analytics provides a managed stream processing service based on perpetually running SQL queries that operate on unbounded streams. You can also use open source Apache streaming technologies like Storm and Spark Streaming in an HDInsight cluster.
6. **Analytical data store:** Many big data solutions prepare data for analysis and then serve the processed data in a structured format that can be queried using analytical

Scanned with CamScanner

tools. The analytical data store used to serve these queries can be a Kimball-style relational data warehouse, as seen in most traditional business intelligence (BI) solutions. Alternatively, the data could be presented through a low-latency NoSQL technology such as HBase, or an interactive Hive database that provides a metadata abstraction over data files in the distributed data store. Azure Synapse Analytics provides a managed service for large-scale, cloud-based data warehousing. HDInsight supports Interactive Hive, HBase, and Spark SQL, which can also be used to serve data for analysis.

7. **Analysis and reporting:** The goal of most big data solutions is to provide insights into the data through analysis and reporting. To empower users to analyze the data, the architecture may include a data modeling layer, such as a multidimensional OLAP cube or tabular data model in Azure Analysis Services. It might also support self-service BI, using the modeling and visualization technologies in Microsoft Power BI or Microsoft Excel. Analysis and reporting can also take the form of interactive data exploration by data scientists or data analysts. For these scenarios, many Azure services support analytical notebooks, such as Jupyter, enabling these users to leverage their existing skills with Python or R. For large-scale data exploration, you can use Microsoft R Server, either standalone or with Spark.

8. **Orchestration:** Most big data solutions consist of repeated data processing operations, encapsulated in workflows that transform source data, move data between multiple sources and sinks, load the processed data into an analytical data store, or push the results straight to a report or dashboard. To automate these workflows, you can use an orchestration technology such Azure Data Factory or Apache Oozie and Sqoop.

Characteristics of Big Data

Three characteristics define Big Data: volume, variety, and velocity. Together, these characteristics define "Big Data". They have created the need for a new class of capabilities to augment the way things are done today to provide a better line of sight and control over our existing knowledge domains and the ability to act on them.

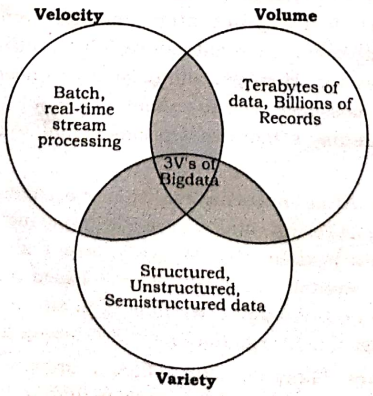

Figure 4.24: Three V big data model

Scanned with CamScanner

1. **The Volume of Data**

The sheer volume of data being stored today is exploding. In the year 2000, 800,000 petabytes (PB) of data were stored in the world. Of course, a lot of the data that's being created today isn't analyzed at all and that's another problem that needs to be considered. This number is expected to reach 35 zeta bytes (ZB) by 2020. Twitter alone generates more than 7 terabytes (TB) of data every day, Facebook 10 TB, and some enterprises generate terabytes of data every hour of every day of the year. It's no longer unheard of for individual enterprises to have storage clusters holding petabytes of data.

The name Big Data itself is related to an enormous size. Big Data is a vast 'volumes' of data generated from many sources daily, such as business processes, machines, social media platforms, networks, human interactions, and many more.

Facebook can generate approximately a billion messages, 4.5 billion times that the "Like" button is recorded, and more than 350 million new posts are uploaded each day. Big data technologies can handle large amounts of data.

2. **Variety**

Big Data can be structured, unstructured, and semi-structured that are being collected from different sources. Data will only be collected from databases and sheets in the past, But these days the data will comes in array forms, that are PDFs, Emails, audios, SM posts, photos, videos, etc.

3. **Velocity**

Velocity plays an important role compared to others. Velocity creates the speed by which the data is created in real-time. It contains the linking of incoming data sets speeds, rate of change, and activity bursts. The primary aspect of Big Data is to provide demanding data rapidly.

Big data velocity deals with the speed at the data flows from sources like application logs, business processes, networks, and social media sites, sensors, mobile devices, etc.

Advantages and disadvantages of Big Data

The advantages of Big Data Analytics below show and how retrieving information and collecting data is useful.

1. **Voluminous Collection**

A large amount of market data can be generated using Big Data analytics, and various graphical and mathematical representations can be made for easy analysis. This massive information is further helpful for deriving market-based conclusions and predict consumer behavior. However, new technology needs to ramp up in this field as traditional software cannot process big data.

2. **Future Insights are mentionable Advantages of Big Data**

With the predictions and statistical data obtained, a business can control prospects. The growth and future problems of a business can be well handled using this analysis. Using these datasets, a company can plan its launches also create new

Scanned with CamScanner

products and services. Scientists foresee the same benefits of Big Data analytics in the healthcare industry and societal affairs.

3. **Big Data Analytics is Cost Efficient**

It is essential to understand the changing trends of the market and to do that, a market analysis needs to be done. At a particular time, a specific direction is followed while others are remaining constant or decreasing. A business is all dependent on the actual demand, and if they can predict it, they can have control over the production. It can save costs incurred in storing raw material and finished products.

4. **Research will take less time**

New software can easily analyze and interpret data sets, which helps make decisions and saves a lot of time. Also, new data can be generated automatically in bulk with updated information and trends. This can help businesses to stay stable in the long run.

5. **Fraud Detection and Prevention**

Big Data is capable of stopping fraudulent transactions, as in the case of banking services. The frauds are getting smarter, and people need to know not to share personal information; the automated software can detect fraudulent accounts and cards. Based on recurring patterns and the spending behavior of consumers, it would also be possible to track what's usually missed manually.

Disadvantages of Big Data

Since all the information collected requires a lot of effort and resources, storing it before it can be examined needs a vast space. Although the analysis of enormous information seems possible, some significant disadvantages of Big Data come to light in terms of space, cost, and user security.

1. **Unstructured Data**

The data collected can be arranged or present in the form of random information. More variations in data can create difficulty in processing results and generating solutions. If the information is broken or unstructured, many users can get neglected while deriving future outcomes or analyzing present scenarios.

2. **Security Concerns are most dreaded disadvantages of Big Data**

For highly secured data or confidential information, highly secured networks are needed for its transfer and storage. Furthermore, with the increased global politics and complex situations between nations, leaked data can be used as an advantage by enemies, so keeping it secure is essential and requires building such a network.

3. **Expensive**

The process of data generation and its analysis is costly without the surety of favorable results. The top businesses can mainly research this field as the space sector, where the wealthiest companies and individuals carry out research. The Cost of setting up super-computers is one of the leading disadvantages of Big Data

Scanned with CamScanner

analytics. Even if the cost is incurred somehow, the information usually residing on the cloud has to be arranged for and will require maintenance.

4. **Skilled Analysts**

The professionals needed to carry out research and run complicated software are highly paid and hard to find. There is a scarcity of individuals skilled for the data analyst job despite the increasing scope in this area of knowledge. Data is the resource of the new generation as to remain in the market; it is necessary to keep yourself updated with further information.

5. **Hardware and Storage**

The servers and hardware needed to store and run high-quality software are very costly and hard to build. Also, the information is available in bulk with continuous changes, and processing requires faster software and applications. And we cannot forget the uncertainty involved with getting accurate results.

MAPREDUCE

Map-Reduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. Map-Reduce programs are written in a particular style influenced by functional programming constructs, specifically idioms for processing lists of data. This module explains the nature of this programming model and how it can be used to write programs which run in the Hadoop environment.

MapReduce is a Hadoop framework used for writing applications that can process vast amounts of data on large clusters. It can also be called a programming model in which we can process large datasets across computer clusters. This application allows data to be stored in a distributed form. It simplifies enormous volumes of data and large scale computing.

There are two primary tasks in MapReduce: **map** and **reduce**. We perform the former task before the latter. In the map job, we split the input dataset into chunks. Map task processes these chunks in parallell. The map we use outputs as inputs for the reduce tasks. Reducers process the intermediate data from the maps into smaller tuples, which reduces the tasks, leading to the final output of the framework.

The MapReduce framework enhances the scheduling and monitoring of tasks. The failed tasks are re-executed by the framework. This framework can be used easily, even by programmers with little expertise in distributed processing. MapReduce can be implemented using various programming languages such as Java, Hive, Pig, Scala, and Python.

How MapReduce in Hadoop works

An overview of MapReduce Architecture and MapReduce's phases will help us understand how MapReduce in Hadoop works.

MapReduce architecture

Scanned with CamScanner

The following diagram shows a MapReduce architecture.

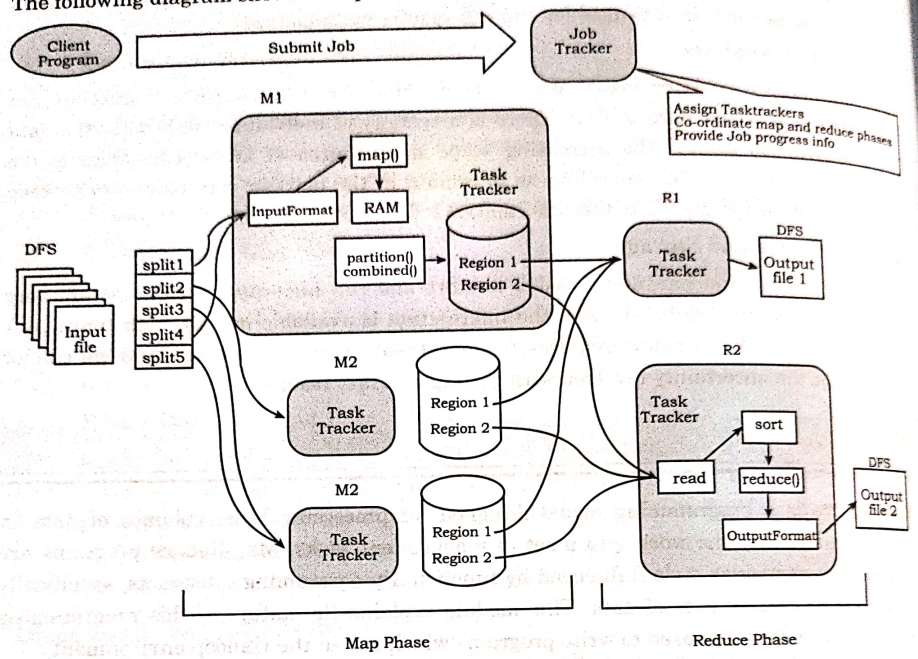

Figure 4.25: MapReduce architecture

MapReduce architecture consists of various components. A brief description of these components can improve our understanding on how MapReduce works.

- **Job:** This is the actual work that needs to be executed or processed
- **Task:** This is a piece of the actual work that needs to be executed or processed. A MapReduce job comprises many small tasks that need to be executed.
- **Job Tracker:** This tracker plays the role of scheduling jobs and tracking all jobs assigned to the task tracker.
- **Task Tracker:** This tracker plays the role of tracking tasks and reporting the status of tasks to the job tracker.
- **Input data:** This is the data used to process in the mapping phase.
- **Output data:** This is the result of mapping and reducing.
- **Client:** This is a program or Application Programming Interface (API) that submits jobs to the MapReduce. MapReduce can accept jobs from many clients.
- **Hadoop MapReduce Master:** This plays the role of dividing jobs into job-parts.
- **Job-parts:** These are sub-jobs that result from the division of the main job.

Scanned with CamScanner

In the MapReduce architecture, clients submit jobs to the MapReduce Master. This master will then sub-divide the job into equal sub-parts. The job-parts will be used for the two main tasks in MapReduce: mapping and reducing.

Phases of MapReduce

The MapReduce program is executed in three main phases: mapping, shuffling, and reducing. There is also an optional phase known as the combiner phase.

Mapping Phase

This is the first phase of the program. There are two steps in this phase: splitting and mapping. A dataset is split into equal units called chunks (input splits) in the splitting step. Hadoop consists of a RecordReader that uses TextInputFormat to transform input splits into key-value pairs.

The key-value pairs are then used as inputs in the mapping step. This is the only data format that a mapper can read or understand. The mapping step contains a coding logic that is applied to these data blocks. In this step, the mapper processes the key-value pairs and produces an output of the same form (key-value pairs).

Shuffling phase

This is the second phase that takes place after the completion of the Mapping phase. It consists of two main steps: sorting and merging. In the sorting step, the key-value pairs are sorted using the keys. Merging ensures that key-value pairs are combined.

The shuffling phase facilitates the removal of duplicate values and the grouping of values. Different values with similar keys are grouped. The output of this phase will be keys and values, just like in the Mapping phase.

Reducer phase

In the reducer phase, the output of the shuffling phase is used as the input. The reducer processes this input further to reduce the intermediate values into smaller values. It provides a summary of the entire dataset. The output from this phase is stored in the HDFS.

The following diagram shows an example of a MapReduce with the three main phases. Splitting is often included in the mapping stage.

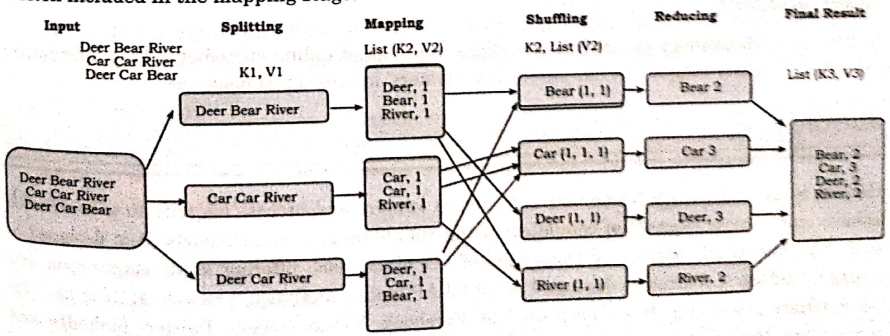

Figure 4.26: Overall MapReduceWord count process

Scanned with CamScanner

Combiner phase

This is an optional phase that's used for optimizing the MapReduce process. It's used for reducing the pap outputs at the node level. In this phase, duplicate outputs from the map outputs can be combined into a single output. The combiner phase increases speed in the Shuffling phase by improving the performance of Jobs.

Benefits of Hadoop MapReduce

- **Speed:** MapReduce can process huge unstructured data in a short time.
- **Fault-tolerance:** The MapReduce framework can handle failures.
- **Cost-effective:** Hadoop has a scale-out feature that enables users to process or store data in a cost-effective manner.
- **Scalability:** Hadoop provides a highly scalable framework. MapReduce allows users to run applications from many nodes.
- **Data availability:** Replicas of data are sent to various nodes within the network. This ensures copies of the data are available in the event of failure.
- **Parallel Processing:** In MapReduce, multiple job-parts of the same dataset can be processed in a parallel manner. This reduces the time taken to complete a task.

Applications of Hadoop MapReduce

The following are some of the practical applications of the MapReduce program.

E-commerce

E-commerce companies such as Walmart, E-Bay, and Amazon use MapReduce to analyze buying behavior. MapReduce provides meaningful information that is used as the basis for developing product recommendations. Some of the information used include site records, e-commerce catalogs, purchase history, and interaction logs.

Social networks

The MapReduce programming tool can evaluate certain information on social media platforms such as Facebook, Twitter, and LinkedIn. It can evaluate important information such as who liked your status and who viewed your profile.

Entertainment

Netflix uses MapReduce to analyze the clicks and logs of online customers. This information helps the company suggest movies based on customers' interests and behavior.

ADOOP

Hadoop is an open-source framework that allows to store and process big data in a distributed environment across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Hadoop is written in Java and is not OLAP (online analytical processing). It is used for batch/offline processing. It is being used by Facebook, Yahoo, Google, Twitter, LinkedIn and many more. Moreover it can be scaled up just by adding nodes in the cluster.

Scanned with CamScanner

It is at the center of an ecosystem of big data technologies that are primarily used to support advanced analytics initiatives, including predictive analytics, data mining and machine learning. Hadoop systems can handle various forms of structured and unstructured data, giving users more flexibility for collecting, processing, analyzing and managing data than relational databases and data warehouses provide.

Hadoop's ability to process and store different types of data makes it a particularly good fit for big data environments. They typically involve not only large amounts of data, but also a mix of structured transaction data and semi-structured and unstructured information, such as internet clickstream records, web server and mobile application logs, social media posts, customer emails and sensor data from the internet of things (IoT).

Hadoop is an open source software project that enables the distributed processing of large data sets across clusters of commodity servers. It is designed to scale up from a single server to thousands of machines, with a very high degree of fault tolerance. Rather than relying on highend hardware, the resiliency of these clusters comes from the software's ability to detect and handle failures at the application layer.

Hadoop Distributed File System (HDFS)

Hadoop Distributed File System (HDFS) is a file system that spans all the nodes in a Hadoop cluster for data storage. It links together the file systems on many local nodes to make them into one big file system.

Hadoop enables a computing solution that is:

- **Scalable**: New nodes can be added as needed, and added without needing to change data formats, how data is loaded, how jobs are written, or the applications on top.
- **Cost effective:** Hadoop brings massively parallel computing to commodity servers. The result is a sizeable decrease in the cost per terabyte of storage, which in turn makes it affordable to model all your data.
- **Flexible:** Hadoop is schema-less, and can absorb any type of data, structured or not, from any number of sources. Data from multiple sources can be joined and aggregated in arbitrary ways enabling deeper analyses than any one system can provide.
- **Fault tolerant:** When you lose a node, the system redirects work to another location of the data and continues processing without missing a fright beat.

Hadoop Daemons

Daemons mean Process. Hadoop Daemons are a set of processes that run on Hadoop. Hadoop is a framework written in Java, so all these processes are Java Processes. Apache Hadoop 2 consists of the following Daemons:

- NameNode
- DataNode
- Secondary nameNode
- Job tracker
- Task tracker

Scanned with CamScanner

"Running Hadoop" means running a set of daemons, or resident programs, on the different servers in your network. These daemons have specific roles; some exist only on one server, some exist across multiple servers.

Name node

Hadoop employs a master/slave architecture for both distributed storage and distributed computation. The distributed storage system is called the Hadoop File System, or HDFS. The NameNode is the master of HDFS that directs the slave DataNode daemons to perform the low-level I/O tasks. The NameNode is the bookkeeper of HDFS; it keeps track of how your files are broken down into file blocks, which nodes store those blocks, and the overall health of the distributed file system. The function of the NameNode is memory and I/O intensive. As such, the server hosting the NameNode typically doesn't store any user data or perform any computations for a MapReduce program to lower the workload on the machine. It's a single point of failure of your Hadoop cluster. For any of the other daemons, if their host nodes fail for software or hardware reasons, the Hadoop cluster will likely continue to function smoothly or you can quickly restart it. Not so for the NameNode.

DataNode

Each slave machine in cluster host a DataNode daemon to perform work of the distributed file system, reading and writing HDFS blocks to actual files on the local file system. Read or write a HDFS file, the file is broken into blocks and the NameNode will tell your client which DataNode each block resides in. Job communicates directly with the DataNode daemons to process the local files corresponding to the blocks. Furthermore, a DataNode may communicate with other DataNodes to replicate its data blocks for redundancy.

Secondary NameNode

Each slave machine in cluster host a DataNode daemon to perform work of the distributed file system, reading and writing HDFS blocks to actual files on the local file system. Read or write a HDFS file, the file is broken into blocks and the NameNode will tell your client which DataNode each block resides in. Job communicates directly with the DataNode daemons to process the local files corresponding to the blocks. Furthermore, a DataNode may communicate with other DataNodes to replicate its data blocks for redundancy.

JobTracker

The JobTracker daemon is the liaison (mediator) between your application and Hadoop. Once you submit your code to your cluster, the JobTracker determines the execution plan by determining which files to process, assigns nodes to different tasks, and monitors all tasks as they're running. Should a task fail, the JobTracker will automatically re launch the task, possibly on a different node, up to a predefined limit of retries. There is only one JobTracker daemon per Hadoop cluster. It's typically run on a server as a master node of the cluster.

Task tracker

The JobTracker daemon is the liaison (mediator) between your application and Hadoop. Once you submit your code to your cluster, the JobTracker determines the execution plan by determining which files to process, assigns nodes to different tasks, and monitors all tasks as

Scanned with CamScanner

they're running. Should a task fail, the JobTracker will automatically re launch the task, possibly on a different node, up to a predefined limit of retries. There is only one JobTracker daemon per Hadoop cluster. It's typically run on a server as a master node of the cluster.

Hadoop configuration modes

Local (standalone mode)

Standalone mode is the default mode in which Hadoop run. Standalone mode is mainly used for debugging where you don't really use HDFS. You can use input and output both as a local file system in standalone mode. You also don't need to do any custom configuration in the files- mapred-site.xml, core-site.xml, hdfs-site.xml. Standalone mode is usually the fastest Hadoop modes as it uses the local file system for all the input and output. Here is the summarized view of the standalone mode:

- Used for debugging purpose
- HDFS is not being used
- Uses local file system for input and output
- No need to change any configuration files
- Default Hadoop Modes

Pseudo-distributed mode

The pseudo-distribute mode is also known as a single-node cluster where both NameNode and DataNode will reside on the same machine. In pseudo-distributed mode, all the Hadoop daemons will be running on a single node. Such configuration is mainly used while testing when we don't need to think about the resources and other users sharing the resource. In this architecture, a separate JVM is spawned for every Hadoop components as they could communicate across network sockets, effectively producing a fully functioning and optimized mini-cluster on a single host. Here is the summarized view of pseudo distributed Mode-

- Single Node Hadoop deployment running on Hadoop is considered as pseudo distributed mode
- All the master & slave daemons will be running on the same node
- Mainly used for testing purpose
- Replication Factor will be ONE for Block
- Changes in configuration files will be required for all the three files- mapred-site.xml, core-site.xml, hdfs-site.xml

Fully distributed mode (Multi-Node Cluster)

This is the production mode of Hadoop where multiple nodes will be running. Here data will be distributed across several nodes and processing will be done on each node. Master and Slave services will be running on the separate nodes in fully-distributed Hadoop Mode.

- Production phase of Hadoop
- Separate nodes for master and slave daemons
- Data are used and distributed across multiple nodes

Scanned with CamScanner

In the Hadoop development, each Hadoop Modes have its own benefits and drawbacks. Definitely fully distributed mode is the one for which Hadoop is mainly known for but again there is no point in engaging the resource while in testing or debugging phase. So standalone and pseudo-distributed Hadoop modes are also having their own significance.

Exercise

1. Describe the evolution from centralized DBMSs to distributed DBMSs.
2. What are the advantages and disadvantages of the DDBMS?
3. Explain the difference between a distributed database and distributed processing.
4. What are the components of a DDBMS?
5. Explain Different levels of data and process distribution.
6. Define and explain the different types of distribution transparency.
7. Explain the difference between distributed databases and client/server architecture.
8. What additional functions does a DDBMS have over a centralized DBMS?
9. What are the main software modules of a DDBMS? Discuss the main functions of each of these modules in the context of the client/server architecture.
10. What is meant by data allocation in distributed database design? What typical units of data are distributed over sites?
11. How is a horizontal partitioning of a relation specified? How can a relation be put back together from a complete horizontal partitioning?
12. How is a vertical partitioning of a relation specified? How can a relation be put back together from a complete vertical partitioning?
13. What are the main categories of NOSQL systems? List a few of the NOSQL systems in each category.
14. What are the main characteristics of NOSQL systems in the areas related to data models and query languages?
15. What are the main characteristics of NOSQL systems in the areas related to distributed systems and distributed databases?
16. What is the CAP theorem? Which of the three properties (consistency, availability, partition tolerance) are most important in NOSQL systems?
17. Describe the architecture of Hadoop in your own words.
18. What are the different releases of Hadoop? In what way are big data and cloud technology complementary to one another?
19. What is Big Data, and where does it come from? How does it work?
20. Why businesses are using Big Data for competitive advantage. Explain the features of Hadoop.

□□□

Scanned with CamScanner