CHAPTER 7

DATA IN THE CLOUD

CHAPTER OUTLINE

After studying this chapter, students will be able to understand the:

- Relational Databases
- Google File System (GFS)
- Hadoop File System (HDFS)
- Google Bigtable
- Apache Hbase
- Storage Mechanism in HBase
- Column Oriented and Row Oriented database
- Amazon Dynamo
- Google Cloud Datastore
- Amazon Simpledb

RELATIONAL DATABASES

A relational database is a form of database that stores and allows access to data elements that are connected. Relational databases are based on the relational model, which is an easy-to-understand method of expressing data in tables. In a relational database, each row in the table is a record with a unique ID called the key. The columns of the table carry data attributes, and each record generally includes a value for each attribute, making it simple to construct links between data points.

In a relational database, each table, also known as a relation, includes one or more data categories in columns, also known as attributes. Each row, also known as a record or tuple, includes a unique instance of data, or key, for the columns' stated categories. Each table has a unique primary key that identifies the data in the table. The relationship between tables may then be defined using foreign keys, which are fields in one table that are linked to the primary key of another table.

A normal business order entry database, for example, might have a table that identifies a client, with columns for name, address, phone number, and so on. Another table would explain an order: the product, the client, the date, the sales price, and so on. A user of a relational database can then acquire a view of the database that is tailored to their requirements. A branch office manager, for example, would want a view or report on all clients who purchased items after a given date. A financial services manager in the same organization may acquire a report on accounts that needed to be paid using the same tables.

Types of Databases

There are several database types, ranging from non-relational flat files to NoSQL to emerging graph databases that are regarded even more relational than traditional relational databases.

A flat-file database is made up of a single table of data with no interrelationships – commonly text files. Users can provide data properties such as columns and data types in this type of file.

Users may manage preset data relationships across various databases using standard relational databases. Microsoft SQL Server, Oracle Database, MySQL, and IBM DB2 are examples of popular relational databases. Cloud-based relational databases, also known as database as a service (DBaaS), are also popular because they allow businesses to outsource database maintenance, patching, and infrastructure support. Amazon Relational Database Service (RDS), Google Cloud SQL, IBM DB2 on Cloud, Microsoft Azure SQL Database, and Oracle Database Cloud Service are examples of cloud relational databases.

A NoSQL database is a relational database alternative that is especially suitable for working with massive volumes of dispersed data. These databases can accommodate a wide range of data structures, including key-value, document, columnar, and graph formats.A graph database goes beyond typical column and row-based relational data models; this NoSQL database employs nodes and edges to express connections between data relationships and can find new ones. Because graph databases are more intelligent than relational databases, their applications include fraud detection and online recommendation engines.

Advantages of Relational Databases

The primary benefits of relational databases are that they allow users to quickly classify and store data, which can then be searched and filtered to retrieve particular information for reports. Relational databases are extremely simple to expand and do not require physical organization. After the first database is created, a new data category can be added without modifying all current apps.

Advantages of Relational Database Include

- Accurate: Data is stored just once, which eliminates data duplication.
- Flexible: Complex queries are easy for users to carry out.
- Collaborative: Multiple users can access the same database.
- Trusted: Relational database models are mature and well-understood.
- Secure: Data in tables within relational database management systems can be limited to allow access by only particular users.

ACID and Relational Databases

Four crucial properties define relational database transactions: atomicity, consistency, isolation, and durability—typically referred to as ACID.

- Atomicity defines all the elements that make up a complete database transaction or none.
- Consistency defines the rules for maintaining data points in a correct state after a transaction.
- Isolation keeps the effect of a transaction invisible to others until it is committed, to avoid confusion.
- Durability ensures that data changes become permanent once the transaction is committed.

A Relational Database Example

Here is a simple example of two tables that a small firm may use to process product orders. The first table is a customer information table, which means that each entry contains a customer's name, address, shipping and payment information, phone number, and other contact information. Each piece of information (each attribute) is in its column, and each row is assigned a unique ID (a key) in the database. Each entry in the second table-a customer order table-includes the ID of the customer who made the purchase, the product ordered, the quantity, the size, and color are chosen, and so on—but not the client's name or contact information.

The only thing these two tables have in common is the ID column (the key). However, because of the shared field, the relational database may establish a link between the two tables. When the company's order processing application submits an order to the database, the database can go to the customer order table, pull the correct product order information, and use the customer ID from that table to look up the customer's billing and shipping information in the customer info table. The database system can then fetch the proper product, the client will receive the order on schedule, and the firm will be compensated.

The Relational Database of the Future: The Self-Driving Database

Relational databases have grown better, quicker, stronger, and easier to deal with throughout time. They have, however, become more complicated, and maintaining the database has long been a full-time job. Instead of focusing on designing creative applications that provide value to the company, developers have had to spend the majority of their time on the administrative activities required to improve database performance.

Today, autonomous technology is leveraging the relational model's capabilities to create a new form of a relational database. The self-driving database (also known as the autonomous database) retains the power and benefits of the relational model while employing artificial intelligence (AI), machine learning, and automation to monitor and enhance query performance and administration duties. To increase query speed, for example, the self-driving database may hypothesize and test indexes to make queries quicker, and then automatically put the best ones into production. These enhancements are made regularly by the self-driving database, with no human intervention required.

Developers have been freed from the boring responsibilities of database management thanks to autonomous technology. For example, they no longer need to plan ahead of time for infrastructure requirements. Instead, they may increase storage and computing resources as needed to accommodate database expansion using a self-driving database. Developers may simply establish an independent relational database in a few clicks, reducing the time required for application development.

Google File System (GFS)

The Google File System is a scalable distributed file system designed for big data-intensive distributed applications. It offers fault tolerance while running on low-cost commodity hardware and gives great aggregate performance to a large number of customers. While GFS has many of the same aims as past distributed file systems, its design has been influenced by observations of application workloads and the technical environment, both current and prospective, which represent a significant divergence from certain preceding file system assumptions. As a result, established options have been reexamined, and dramatically alternative design points have been explored.

Google File System (GFS) is a scalable distributed file system (DFS) designed by Google Inc. to meet Google's growing data processing needs. GFS supports huge networks and linked nodes with fault tolerance, dependability, scalability, availability, and performance. GFS is comprised of several storage systems constructed from low-cost commodity hardware components. It is designed to meet Google's various data consumption and storage requirements, such as their search engine, which creates massive volumes of data that must be kept. The Google File System took advantage of the strengths of off-the-shelf servers while reducing hardware flaws. GoogleFS is another name for GFS.

The storage requirements were satisfied satisfactorily by the file system. It is widely utilized within Google as a storage platform for the collection and processing of data utilized by our services as well as research and development projects requiring big data sets. The biggest cluster to date has hundreds of terabytes of storage spread across thousands of disks on over a thousand servers, and it is used concurrently by hundreds of clients.

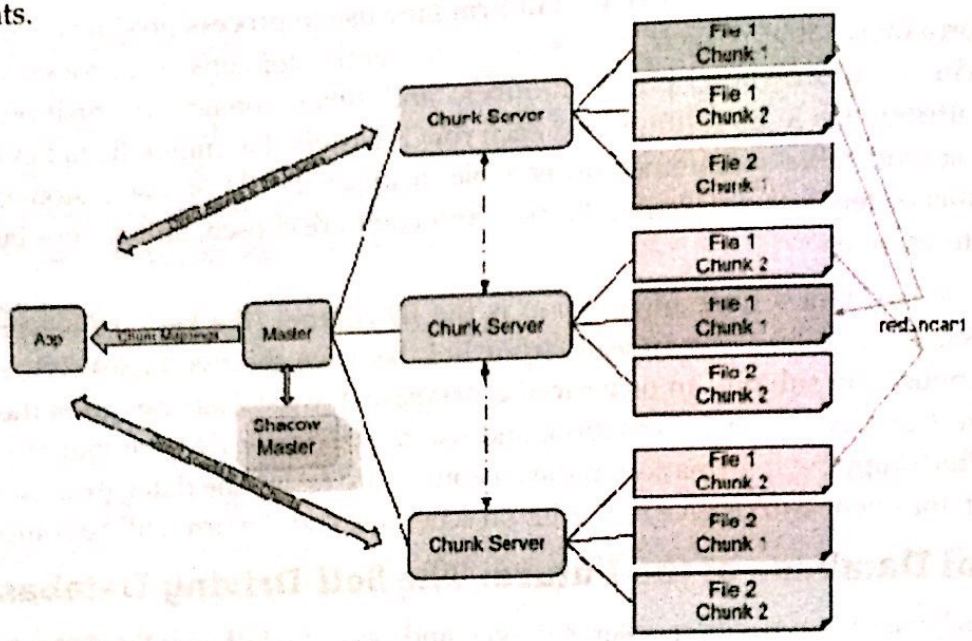

Figure 7.1: Design of GFS

The GFS node cluster consists of a single master and numerous chunk servers that are constantly accessed by various client systems. Data is stored on local drives by chunk servers as Linux files. Data is stored in big chunks (64 MB), which are duplicated at least three times on the network. Because of the huge chunk size, network overhead is reduced.

GFS is intended to meet Google's huge cluster requirements while without burdening apps. Pathnames are used to identify files in hierarchical folders. The master controls metadata such as namespace, access control data, and mapping information by interacting with and monitoring the status updates of each chunk server via scheduled heartbeat messages.

GFS Features Include

- Fault tolerance
- Critical data replication
- Automatic and efficient data recovery
- High aggregate throughput
- Reduced client and master interaction because of large chunk server size
- Namespace management and locking
- High availability

The biggest GFS clusters have over 1,000 nodes with a total disk storage capacity of 300 TB. Hundreds of people can use it constantly.

HADOOP FILE SYSTEM (HDFS)

The Hadoop File System was created with a distributed file system design. It runs on standard hardware. In contrast to other distributed systems, HDFS is extremely fault-tolerant and built with low-cost hardware. HDFS stores a big quantity of data and makes it easy to access. To store such large amounts of data, the files are spread over numerous computers. These files are kept in a redundant form to protect the system from data loss in the event of a breakdown. HDFS also enables parallel processing of applications.

Hadoop applications use the Hadoop Distributed File Solution (HDFS) as their primary data storage system. It implements a distributed file system that allows high-performance access to data across highly scalable Hadoop clusters using a NameNode and DataNode architecture. HDFS is an important component of the numerous Hadoop ecosystem technologies since it provides a dependable way of maintaining massive data pools and supporting associated big data analytics applications.

Features of HDFS

- It is suitable for the distributed storage and processing.
- Hadoop provides a command interface to interact with HDFS.
- The built-in servers of namenode and datanode help users to easily check the status of the cluster.
- Streaming access to file system data.
- HDFS provides file permissions and authentication.

HDFS Architecture

The architecture of a Hadoop File System is shown in figure 7.2. Those components of the architecture are described as below:

Namenode

The *namenode* is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks:

- Manages the file system namespace.
- Regulates client's access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories.

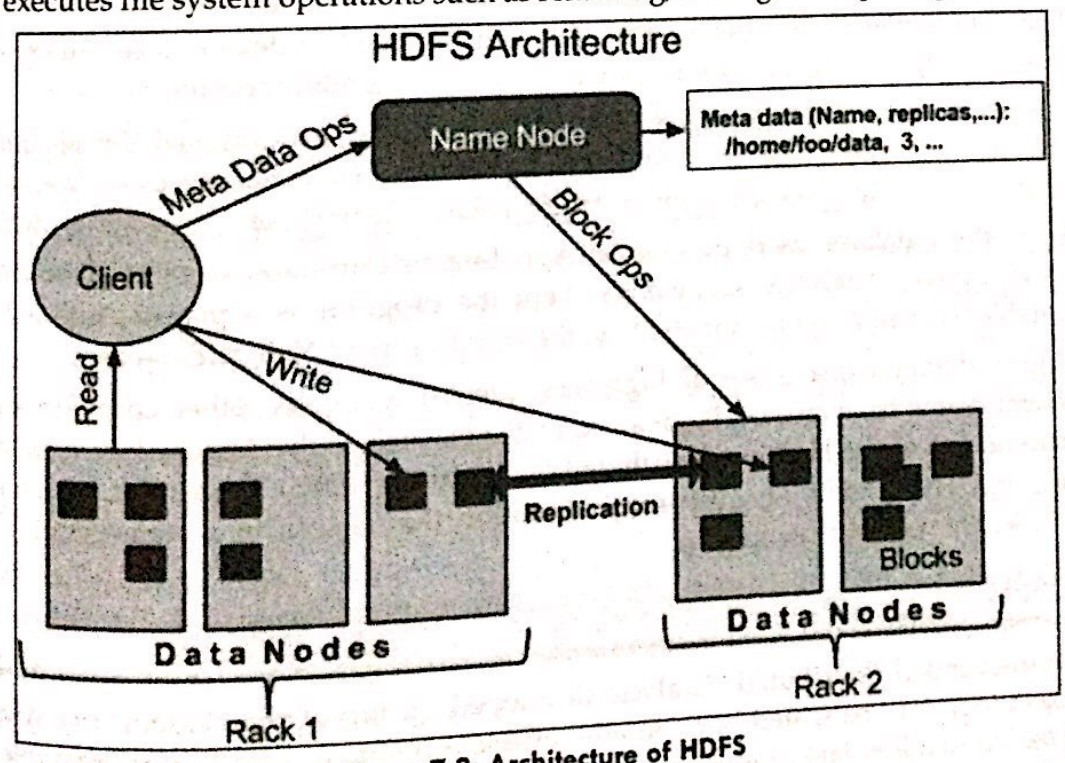

Figure 7.2: Architecture of HDFS

Datanode

The datanode is a piece of commodity hardware that runs the GNU/Linux operating system as well as the datanode software. A datanode will exist for each node (Commodity hardware/System) in a cluster. These nodes are in charge of their system's data storage.

- Datanodes perform read-write operations on the file systems, as per the client's request.
- They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

Block

In general, user data is saved in HDFS files. In a file system, the file will be partitioned into one or more segments and/or stored in separate data nodes. Blocks are the name given to these file chunks. In other terms, a Block is the smallest quantity of data that HDFS can read or write. The default block size is 64MB, although it may be adjusted in the HDFS setup as needed.

Goals of HDFS

- **Fault detection and recovery**: Because HDFS relies on a large number of commodity hardware components, component failure is common. As a result, HDFS should have techniques for rapid and automated failure detection and recovery.
- **Huge datasets**: To manage applications with large datasets, HDFS should have hundreds of nodes per cluster.
- **Hardware at data:** When the computation takes place near the data, a requested job can be completed quickly. It decreases network traffic while increasing throughput, especially when large datasets are involved.

GOOGLE BIGTABLE

Google Bigtable is a distributed, column-oriented data store developed by Google Inc. to manage massive volumes of structured data related to the company's Internet search and Web services operations.

Bigtable was created to enable applications that need large scalability; the technology was meant to be utilized with petabytes of data from its inception. The database was designed to run on clustered servers and has a basic data format described by Google as "a sparse, distributed, permanent multi-dimensional sorted map." The data is placed in order by row key, and the map's indexing is sorted by row, column keys, and timestamps. Algorithms for compression aid in achieving high capacity.

BigTable is a petabyte-scale, fully managed NoSQL database service designed for big analytical and operational workloads. Google Cloud Bigtable is a NoSQL Big Data database service. It's the same database that underpins many of Google's main services, such as Search, Analytics, Maps, and Gmail.'

Google Bigtable is the database used by Google App Engine Datastore, Google Personalized Search, Google Earth, and Google Analytics. Google has kept the program as a private, internal technology. Nonetheless, Bigtable has had a significant effect on the architecture of NoSQL databases.

Because of Google's detailed disclosure of Bigtable's internal workings, other corporations and open-source development teams have created Bigtable equivalents, such as the Apache HBase database, which is designed to run on top of the Hadoop Distributed File System (HDFS). Cassandra, which is developed at Facebook Inc., and Hypertable, an open-source solution that is sold in a commercial version as an alternative to HBase, are two more examples.

APACHE HBASE

HBase is a column-oriented distributed database developed on top of the Hadoop file system. It is an open-source project that may be scaled horizontally. HBase is a data architecture comparable to Google's big table that is meant to allow fast random access to massive volumes of structured data. It makes use of the Hadoop File System's fault tolerance (HDFS).

Apache HBase is a Hadoop-based distributed, scalable NoSQL big data storage. HBase is capable of hosting very large tables–billions of rows and millions of columns–and of providing real-time, random read/write access to Hadoop data. HBase is a multi-column data store inspired by Google Bigtable, the database interface to Google's proprietary File System. HBase adds Bigtable-like features to Hadoop-compatible file systems like MapR XD. HBase scales linearly over very large datasets and allows for the easy combination of data sources with heterogeneous topologies and schemas.

HBase is a Hadoop ecosystem component that allows random real-time read/write access to data in the Hadoop File System. Data may be stored in HDFS directly or through HBase. Using HBase, the data consumer reads/accesses the data in HDFS at random. HBase is a database that sits on top of the Hadoop File System and allows for reading and writing access.

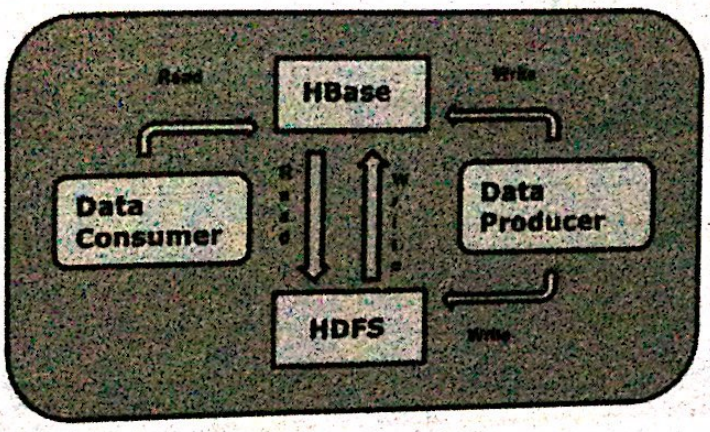

Figure 7.3: HBase and HDFS Interaction

HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS does not support fast individual record lookups.	HBase provides fast lookups for larger tables.
It provides high latency batch processing; no concept of batch processing.	It provides low latency access to single rows from billions of records (Random access).
It provides only sequential access to data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.

Storage Mechanism in HBase

HBase is a column-oriented database, with tables ordered by row. Only column families, which are key-value pairs, are defined in the table structure. A table contains many column families, each of which can include any number of columns. Subsequent column values are saved on the disk in a logical order. A timestamp is associated with each cell value in the table.

In a nutshell, in an HBase:

- The table is a collection of rows.
- The row is a collection of column families.
- Column family is a collection of columns.
- The column is a collection of key-value pairs.

An example schema of a table in HBase is provided below.

Row_ id	Column Family			Column Family			Column Family			Column Family		
	col1	col2	col3	col1	col2	col3	col1	col2	col3	col1	col2	col3
1												
2												
3												

COLUMN ORIENTED AND ROW ORIENTED DATABASE

Column-oriented databases, as opposed to row-oriented databases, store data tables as portions of columns of data. They will have column families.

Row-Oriented Database	Column-Oriented Database
It is suitable for the Online Transaction Process (OLTP).	It is suitable for Online Analytical Processing (OLAP).
Such databases are designed for a small number of rows and columns.	Column-oriented databases are designed for huge tables.

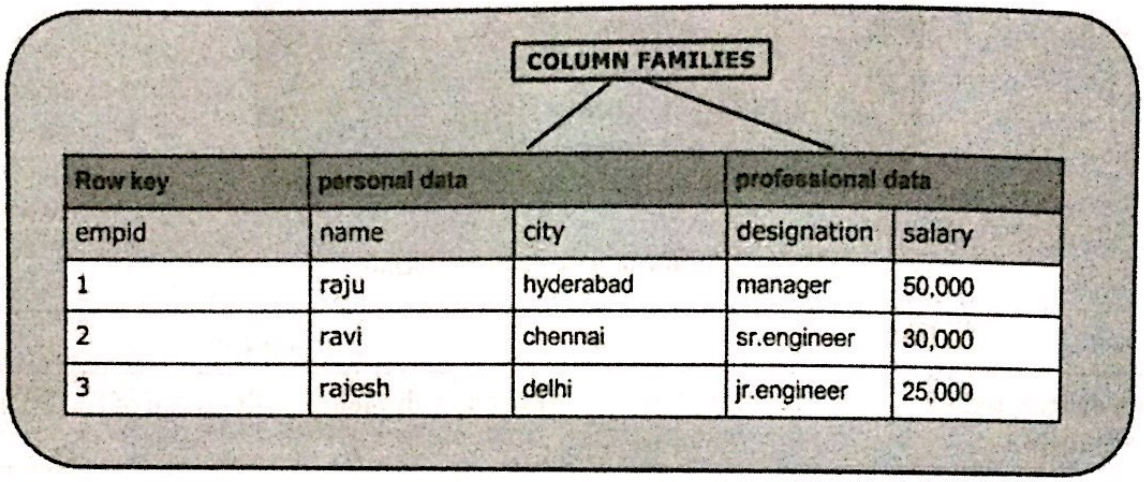

Figure 7.4: Column families in a column-oriented database

HBase	RDBMS
HBase is schema-less, it does not have the concept of fixed columns schema; defines only column families.	An RDBMS is governed by its schema, which describes the whole structure of tables.
It is built for wide tables. HBase is horizontally scalable.	It is thin and built for small tables. Hard to scale.
No transactions are there in HBase.	RDBMS is transactional.
It has de-normalized data.	It will have normalized data.
It is good for semi-structured as well as structured data.	It is good for structured data.

Features of HBase

- HBase is linearly scalable.
- It has automatic failure support.
- It provides consistent read and write.
- It integrates with Hadoop, both as a source and a destination.
- It has an easy java API for clients.
- It provides data replication across clusters.

Where to use HBase?

- Apache HBase is used to have random, real-time read/write access to Big Data.
- It hosts very large tables on top of clusters of commodity hardware.
- Apache HBase is a non-relational database modeled after Google's Bigtable. Bigtable acts up on Google File System, likewise, Apache HBase works on top of Hadoop and HDFS.

Applications of HBase

- It is used whenever there is a need to develop complex applications.
- HBase is used whenever we need to provide fast random access to available data.
- Companies such as Facebook, Twitter, Yahoo, and Adobe use HBase internally.

AMAZON DYNAMO

Amazon offers DynamoDB, a fully managed proprietary NoSQL database service that supports key-value and document data formats, as part of the Amazon Web Services portfolio. DynamoDB provides a comparable data architecture to Dynamo and gets its name from it, but it has a distinct underlying implementation. DynamoDB employs synchronous replication over many data centers for high durability and availability, and it has a multi-master design that requires the client to resolve version conflicts.

Amazon DynamoDB is a key-value and document database that promises performance in single-digit milliseconds at any size. It is a fully managed, multiregional, multi-master database for internet-scale applications that has built-in security, backup and restore, and in-memory caching. DynamoDB can handle more than 10 trillion requests per day, with peaks of more than 20 million requests per second. Many of the world's fastest-growing companies, like Lyft, Airbnb, and Redfin, as well as corporations like Samsung, Toyota, and Capital One, rely on DynamoDB's scale and performance to serve mission-critical workloads. Hundreds of thousands of Amazon Online Services customers have selected DynamoDB as their key-value and document database for mobile, web, gaming, ad tech, IoT, and other applications requiring low-latency data access at any scale. Create a new table for your application and leave the rest to DynamoDB.

DynamoDB is distinct from other Amazon services in that it allows developers to pay for a service based on throughput rather than storage. If Auto Scaling is enabled, the database will automatically scale. Administrators may also request throughput modifications, and DynamoDB can distribute data and traffic over many servers utilizing solid-state disks, resulting in predictable performance.

Benefits

Performance at scale - DynamoDB provides reliable, single-digit millisecond response speeds at any size, enabling it to serve some of the world's biggest scale applications. You can create apps with nearly limitless throughput and storage. DynamoDB global tables replicate your data across different AWS Regions to provide quick, local data access for globally dispersed applications.

Serverless - There are no servers to deploy, patch, or manage with DynamoDB, and no software to install, maintain, or run. DynamoDB adjusts tables up and down automatically to accommodate capacity while maintaining performance. Availability and fault tolerance are built-in, so you do not have to structure your applications to support these features. DynamoDB supports both provisioned and on-demand capacity options, allowing you to save money by setting capacity per job or paying for just the resources you use.

Enterprise-ready - DynamoDB provides ACID transactions, allowing you to create mission-critical applications at scale. By default, DynamoDB encrypts all data and offers fine-grained identity and access control for all of your tables. You can generate comprehensive backups of hundreds of terabytes of data in seconds with no impact on table performance and restore to any point in time in the last 35 days with no downtime. DynamoDB is also supported by a service level agreement to ensure uptime.

GOOGLE CLOUD DATASTORE

Google Cloud Datastore is a Google Cloud Platform service that provides a highly scalable, fully managed NoSQL database. Cloud Datastore is based on Bigtable and Megastore technologies from Google. Cloud Datastore is a NoSQL document database designed for online and mobile applications that require automated scalability, excellent performance, and simplicity of development.

Features of Google Cloud Datastore include:

- **Atomic transactions.** Cloud Datastore can perform a series of operations in which all or none of them succeed.
- **High availability of reads and writes.** Cloud Datastore is hosted in Google data centers, which employ redundancy to reduce the effects of single points of failure.
- **Massive scalability with high performance.** To manage scalability automatically, Cloud Datastore has a distributed design. Cloud Datastore employs a combination of indexes and queries limitations to ensure that your queries grow with the size of your result set rather than the size of your data set.
- **Flexible storage and querying of data.** Cloud Datastore is easily mapped to object-oriented and scripting languages, and it is accessible to applications via a variety of clients. It also has a query language that is similar to SQL.
- **Balance of strong and eventual consistency.** Cloud Datastore assures that entity lookups using key and ancestor searches always provide highly consistent results. All other searches become consistent in the end. The consistency models enable your application to provide an excellent user experience even when dealing with massive volumes of data and users.
- **Encryption at rest.** Cloud Datastore automatically encrypts all data before it is written to disk and immediately decrypts the data when accessed by an authorized user. For further information, see Server-Side Encryption.
- **Fully managed with no planned downtime.** Google manages the Cloud Datastore service, allowing you to focus on your application. When the service is scheduled for an upgrade, your application can continue to use Cloud Datastore.

AMAZON SIMPLEDB

Amazon SimpleDB is a highly available NoSQL data storage that relieves database administrators of their duties. Developers just use web service requests to save and access data objects, and Amazon SimpleDB handles the rest. Amazon SimpleDB, unlike relational databases, is designed to provide high availability and flexibility with minimal or no administration overhead.Amazon SimpleDB automatically builds and manages numerous globally dispersed duplicates of your data behind the scenes to provide high availability and data durability. The service only costs you for the resources used in storing your data and delivering your requests. You may alter your data model on the fly, and data is indexed for you automatically. You can concentrate on application development without having to worry about infrastructure provisioning, high availability, software maintenance, schema, and index management, or performance optimization using Amazon SimpleDB.

This SimpleDB service provides streamlined access to data storage and query tools, allowing users to quickly upload data and easily retrieve or change that data. Customers who have relatively simple data needs, such as data storage, should utilize SimpleDB. A firm, for example, may use cookies to track visits to its website. The company may read the cookies to determine the visitor's identity and the feeds they are interested in. Amazon SimpleDB allows users to store tens of attributes for a million customers, but not thousands of attributes for a single customer.

Benefits of Amazon Simple DB

- **Low touch**: The service enables you to devote your complete attention to value-added application development rather than time-consuming database administration. Amazon SimpleDB supports infrastructure provisioning, hardware and software maintenance, data replication and indexing, and performance tweaking automatically.
- **Highly available:** Amazon SimpleDB produces numerous globally scattered copies of each data item you save automatically. This ensures high availability and durability; if one replica fails, Amazon SimpleDB may fail over to another replica in the system.
- **Flexible**: As your company or application grows, you can quickly represent these changes in Amazon SimpleDB without having to worry about breaking a strict schema or refactoring code — simply add another characteristic to your Amazon SimpleDB data set as needed. You may also pick between consistent and eventually consistent read requests, giving you the freedom to tailor read performance (latency and throughput) and consistency requirements to the demands of your application, or even divergent sections of your application.
- **Simple to use**: Amazon SimpleDB simplifies access to the store and query tasks that are normally performed by a relational database cluster — while excluding additionally complicated, often unneeded database activities. Through a simple set of API methods, you may rapidly add data and simply retrieve or change that data.
- **Designed for use with other Amazon Web Services**: Amazon SimpleDB is intended to be readily integrated with other AWS services such as Amazon S3 and EC2, providing the framework for developing web-scale applications. Developers, for example, can run their programs on Amazon EC2 and store their data objects on Amazon S3. Amazon SimpleDB may then be used from within the application in Amazon EC2 to query the object metadata and return references to the objects stored in Amazon S3. Developers may also utilize Amazon SimpleDB in conjunction with Amazon RDS for applications that require both relational and non-relational databases. Data transfers between Amazon SimpleDB and other Amazon Web Services within the same Region are free.
- **Secure**: Amazon SimpleDB offers an HTTPS endpoint for a secure, encrypted connection between your application or client and your domain. Furthermore, by integrating with AWS Identity and Access Management, you may provide user or group-level access to specified SimpleDB domains and activities.
- **Inexpensive**: Amazon SimpleDB passes on the financial benefits of Amazon's scalability to you. You just pay for the resources you use. This implies that with Amazon SimpleDB, data store reads, and writes are paid by the computing resources spent by each operation, and you are not billed for computing resources when you are not actively utilizing them (i.e., making requests).

MULTI-TENANT CLOUD ✓ 5 unit ma xa syllabus ma.

A multi-tenant cloud is a cloud computing architecture that enables clients to share computer resources in either the public or private cloud. Each tenant's data is segregated and hidden from other residents. Users in a multi-tenant cloud system have their area to store their projects and data. Each segment of a multi-tenant cloud network comprises sophisticated permissions to provide each user access to just their stored information while also protecting them from other cloud tenants. Each tenant's data is unavailable to all other tenants inside the cloud architecture and may only be accessed with the cloud provider's rights.

Customers, or tenants, in a private cloud, might be various individuals or groups inside a single firm, but in a public cloud, completely separate enterprises can securely share their server space. The multi-tenancy approach is used by the majority of public cloud providers. It enables them to run servers with single instances, which saves money and streamlines upgrades.

Benefits of Amazon Simple DB

- **Low touch**: The service enables you to devote your complete attention to value-added application development rather than time-consuming database administration. Amazon SimpleDB supports infrastructure provisioning, hardware and software maintenance, data replication and indexing, and performance tweaking automatically.
- **Highly available:** Amazon SimpleDB produces numerous globally scattered copies of each data item you save automatically. This ensures high availability and durability; if one replica fails, Amazon SimpleDB may fail over to another replica in the system.
- **Flexible**: As your company or application grows, you can quickly represent these changes in Amazon SimpleDB without having to worry about breaking a strict schema or refactoring code – simply add another characteristic to your Amazon SimpleDB data set as needed. You may also pick between consistent and eventually consistent read requests, giving you the freedom to tailor read performance (latency and throughput) and consistency requirements to the demands of your application, or even divergent sections of your application.
- **Simple to use**: Amazon SimpleDB simplifies access to the store and query tasks that are normally performed by a relational database cluster – while excluding additionally complicated, often unneeded database activities. Through a simple set of API methods, you may rapidly add data and simply retrieve or change that data.
- **Designed for use with other Amazon Web Services**: Amazon SimpleDB is intended to be readily integrated with other AWS services such as Amazon S3 and EC2, providing the framework for developing web-scale applications. Developers, for example, can run their programs on Amazon EC2 and store their data objects on Amazon S3. Amazon SimpleDB may then be used from within the application in Amazon EC2 to query the object metadata and return references to the objects stored in Amazon S3. Developers may also utilize Amazon SimpleDB in conjunction with Amazon RDS for applications that require both relational and non-relational databases. Data transfers between Amazon SimpleDB and other Amazon Web Services within the same Region are free.
- **Secure**: Amazon SimpleDB offers an HTTPS endpoint for a secure, encrypted connection between your application or client and your domain. Furthermore, by integrating with AWS Identity and Access Management, you may provide user or group-level access to specified SimpleDB domains and activities.
- **Inexpensive**: Amazon SimpleDB passes on the financial benefits of Amazon's scalability to you. You just pay for the resources you use. This implies that with Amazon SimpleDB, data store reads, and writes are paid by the computing resources spent by each operation, and you are not billed for computing resources when you are not actively utilizing them (i.e., making requests).

MULTI-TENANT CLOUD

A multi-tenant cloud is a cloud computing architecture that enables clients to share computer resources in either the public or private cloud. Each tenant's data is segregated and hidden from other residents. Users in a multi-tenant cloud system have their area to store their projects and data. Each segment of a multi-tenant cloud network comprises sophisticated permissions to provide each user access to just their stored information while also protecting them from other cloud tenants. Each tenant's data is unavailable to all other tenants inside the cloud architecture and may only be accessed with the cloud provider's rights.

Customers, or tenants, in a private cloud, might be various individuals or groups inside a single firm, but in a public cloud, completely separate enterprises can securely share their server space. The multi-tenancy approach is used by the majority of public cloud providers. It enables them to run servers with single instances, which saves money and streamlines upgrades.

SINGLE-TENANT CLOUD

A single customer is hosted on a server and has access to it in a single-tenant cloud. Because multi-tenancy systems host several customers on the same servers, it is critical to thoroughly understand the security and performance provided by the supplier. Customers have more control over data, storage, security, and performance in single-tenant clouds.

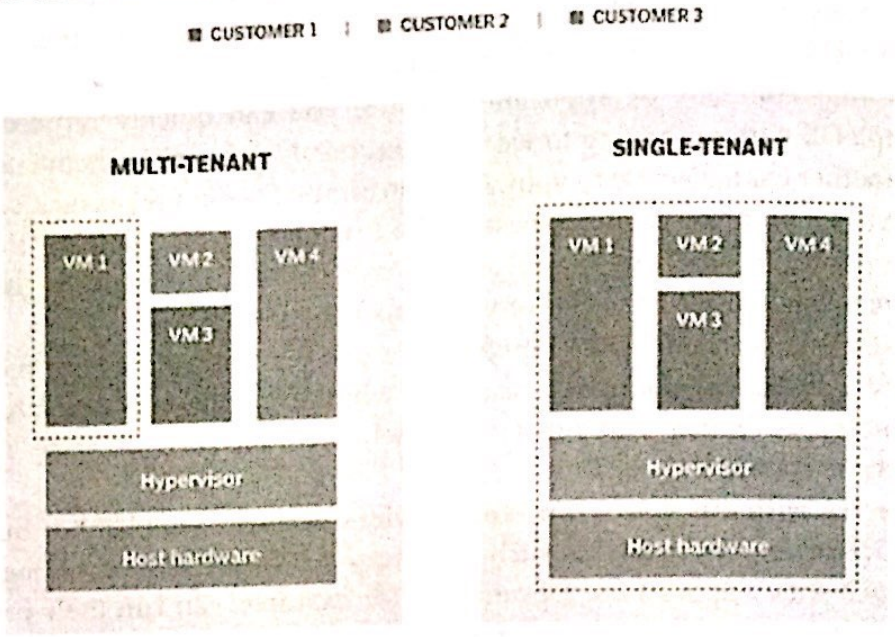

Figure 7.5: Multi-tenant and single-tenant mechanism

BENEFITS OF MULTI-TENANT CLOUD

Multi-tenant cloud networks offer more storage and better access than single-tenancy clouds, which have restricted access and security settings. Cloud computing multi-tenancy makes a bigger pool of resources available to a bigger number of individuals without losing privacy or security or slowing down applications. The virtualization of storage locations in cloud computing enables flexibility and simplicity of access from virtually any device or location.

Example of Multi-tenancy

The structure of multi-tenant clouds can be compared to that of an apartment building. Each tenant has access to their apartment within the framework of the building's agreement, and only authorized persons are permitted to enter the specified apartments. However, utilities such as water, power, and common rooms are shared by the entire building.

PARALLEL COMPUTING

Parallel computing is a sort of computer architecture in which many processors simultaneously execute or process an application or calculation. Parallel computing aids in the performance of big calculations by splitting the workload across several processors, all of which work on the computation at the same time. The majority of supercomputers run using parallel computing methods. Parallel processing is another name for parallel computing.

Parallel processing is typically used in operating environments/scenarios that need large computing or processing capability. Parallel computing's primary goal is to enhance available compute power for quicker application processing or job resolution. Parallel computing infrastructure is often hosted in a single facility where multiple processors are deployed in a server rack or independent servers are linked together.

The application server provides a calculation or processing request that is broken down into little pieces or components that are processed concurrently on each processor/server. Parallel processing can be divided into bit-level, instructional-level, data-level, and task-level parallelism.

PARALLEL COMPUTING AND SERIAL COMPUTING

Serial Computing

Traditionally, the software has been written for serial computation:

- A problem is broken into a discrete series of instructions
- Instructions are executed sequentially one after another
- Executed on a single processor
- Only one instruction may execute at any moment in time

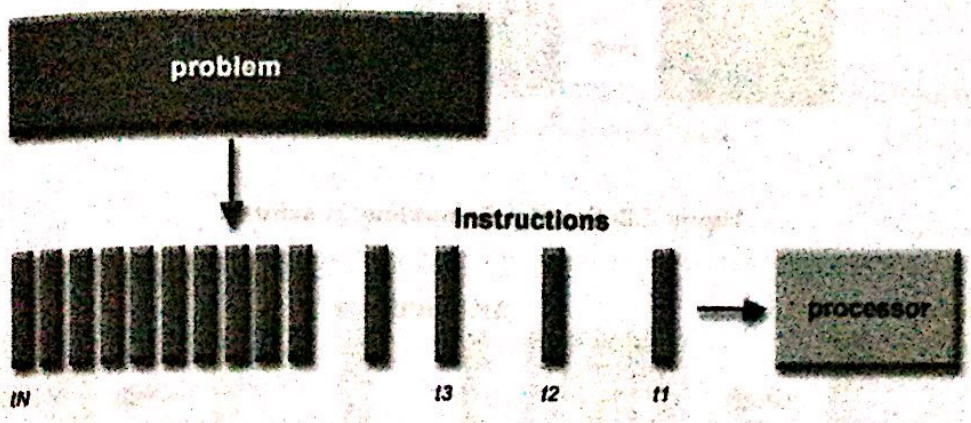

Figure 7.6: Serial Computing

For example:

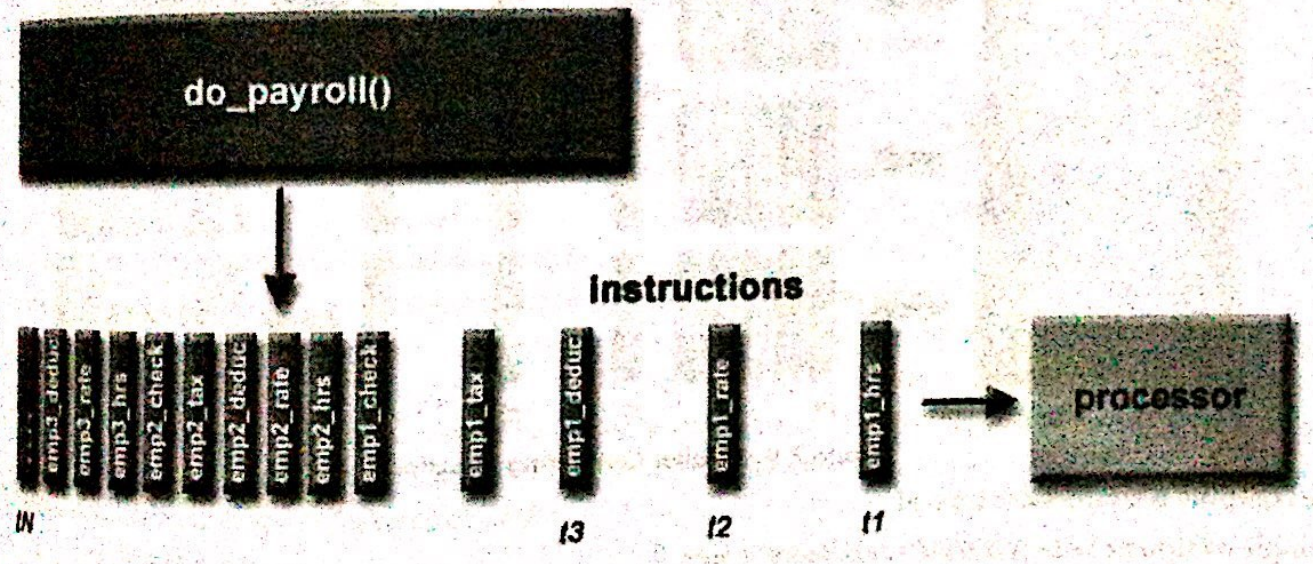

Figure 7.7: Serial Computing in action

PARALLEL COMPUTING

Parallel computing, in its most basic form, is the utilization of many computes resources to solve a computational problem at the same time:

- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down into a series of instructions
- Instructions from each part execute simultaneously on different processors
- An overall control/coordination mechanism is employed

The computational problem should be able to:

- Be broken apart into discrete pieces of work that can be solved simultaneously.
- Execute multiple program instructions at any moment in time.
- Be solved in less time with multiple compute resources than with a single compute resource.

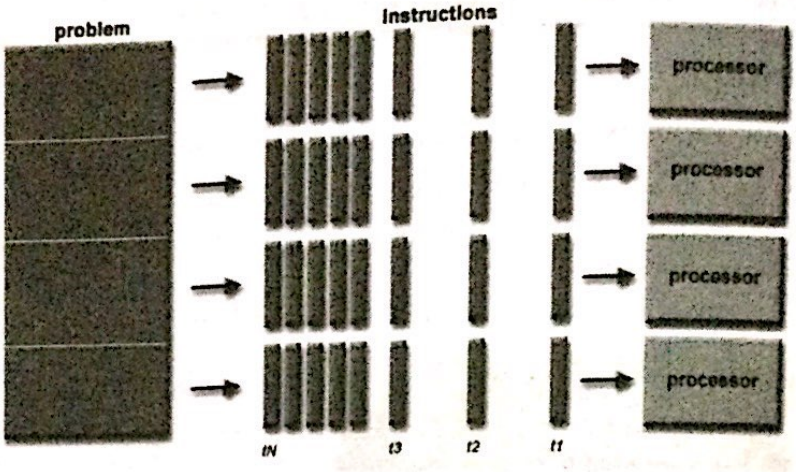

Figure 7.8: Parallel Computing in action

For example:

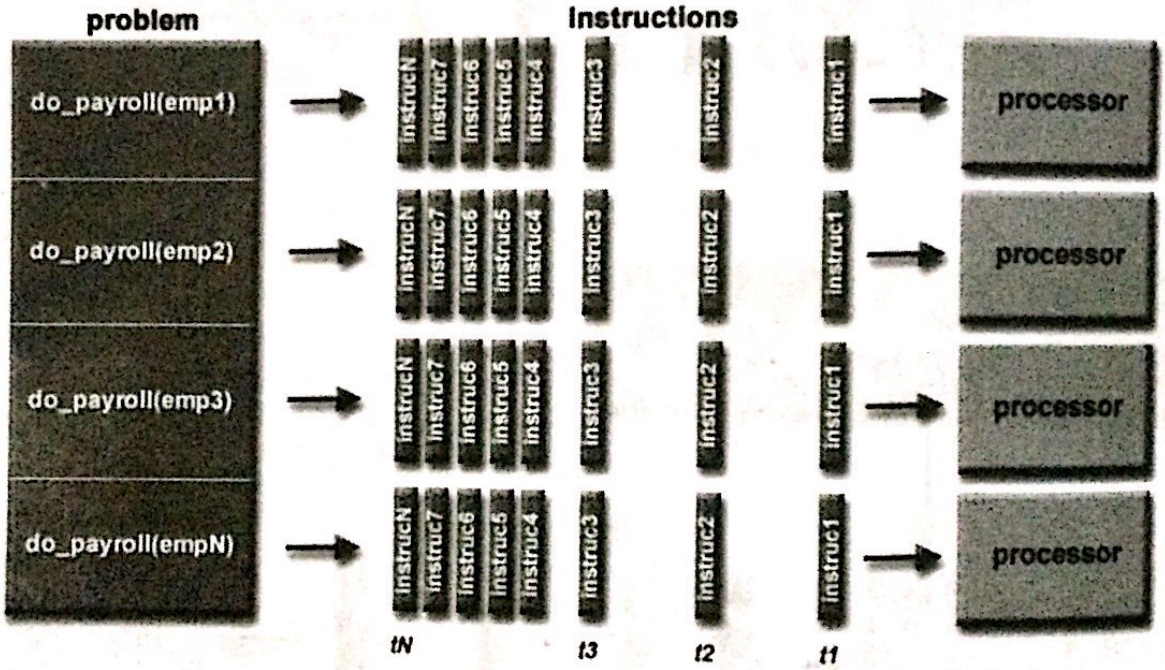

Figure 7.9: Parallel Computing in action

The compute resources are typical:

A single computer with multiple processors/cores

An arbitrary number of such computers connected by a network

Advantages

- Parallel computing saves time by allowing programs to be executed in less time.
- Larger problems must be solved in a shorter period.
- Parallel computing is far superior to serial computing for modeling, simulating, and comprehending complicated real-world phenomena.
- Many real-world problems are so huge and/or complicated that solving them on a single computer is impractical or impossible, especially given limited computer capacity.
- A lot of things can be performed at the same time if you use parallel computing resources.

It can store a large amount of data and perform speedy data calculations.

Disadvantages

Programming to target Parallel architecture is a bit more challenging and needs thorough study and practice. The usage of parallel computing allows you to address computationally and data-intensive problems utilizing multicore processors, but sometimes the control algorithms may not provide excellent results which can affect the system convergence.

The additional cost due to increased execution time is related to data transfers, synchronization, communication, thread creation/destruction, and so on. These costs can be rather high at times and may even outweigh the benefits of parallelization.

To boost speed, various code tweaks must be made for various target architectures.

In the case of clusters, better cooling systems are necessary.

Multi-core designs consume a lot of power.

Parallel solutions are more difficult to build, debug, and prove right, and they frequently perform worse than serial versions due to communication and coordination overhead.

OBJECTIVE QUESTIONS

1. **The relational model is concerned with ?**
   a. Data structure and Data integrity b. Data Manipulation
   c. Both A and B d. None of these
2. **Logical design of database, is known to be**
   a. Database relation b. Database instance
   c. Database entity d. Database Schema
3. **A_____ in a table represents a relationship among a set of values.**
   a. Column b. B. Key c. C. Row d. D Entry
4. **For each attribute of a relation, there is a set of permitted values, called the _____ of that attribute.**
   a. Domain b. B. Relation c. C. Set d. D. Schema
5. **Course (course-id, course-name, course-length)**
   Here the course-id,course-name and course-length are _____ and course is a _____.
   a. Relations, Attribute b. B. Attributes, Relation
   c. Tuples, Relation d. D. Tuples, Attribute
6. **Scalable distributed file system designed for big data-intensive distributed applications**
   a. GFS b MySQL c. Relational DB d. AZURE
7. **Which of the following are features of GFS?**
   a. Fault tolerance b. Critical data replication
   c. High aggregate throughput d. All of the above
8. **Which is not the component of HDFS**
   a. Namenode b. Datanode c. Block d. Replication Node
9. **The default block size of HDFS is**
   a. 64MB b. 32MB c. 128MB d. 256MB
10. **The goals of HDFS includes**
    a. Fault detection and recovery b. Huge datasets
    c. Hardware at data d. All of the above
11. **BigTable is the product of**
    a. Google b. Amazon c. Microsoft d. Manjarasoft

12. The scale of BigTable is

a. Petabyte b. TeraByte c. GigaByte d. ExaByte

13. HBase is

a. column-oriented distributed
b. object-oriented database
c. relational database
d. row-oriented database

14. In an HBase:

a. The table is a collection of rows.
b. he row is a collection of column families.
c. Column family is a collection of columns.
d. The column is a collection of key-value pairs.

15. A column-oriented database is suitable for

a. OLTP b. OLAP c. Join d. None of the above

16. NoSQL database service that supports key-value and document data formats

a. DynamoDB b. HBase c. MySQL d. BigTable

17. A cloud computing architecture that enables clients to share computer resources in either the public or private cloud refers to

a. Multi-tenant cloud
b. Single-tenant cloud
c. Both A and B
d. None of the above

18. The computer system of a parallel computer is capable of

a. Decentralized computing
b. Parallel computing
c. Centralized computing
d. Decentralized computing

19. In which application system Distributed systems can run well?

a. HPC b. HTC c. HRC d. Both A and B

20. In which systems desire HPC and HTC.

a. Adaptivity b. Transparency c. Dependency d. Secretive

21. Utilization of many computes resources to solve a computational problem at the same time is referred to as:

a. Serial Computing
b. Grid Computing
c. Parallel Computing
d. one of the above

QUESTION

1. What is a relational database? Explain the advantages and disadvantages of the Relational database model in context to the cloud.
2. Explain ACID properties with examples.
3. What is GFS? Explain the features of GFS.
4. Explain Hadoop File System along with its Architecture.
5. How is Google Bigtable suitable for the applications that need high scalability? Explain.
6. Describe Apache HBase. Differentiate HDFS with HBase.
7. Explain the storage mechanisms of HBase. Differentiate HBase with RDBMS.
8. What is Amazon Dynamo? Describe its benefits.
9. What is Google Cloud Datastore? Features of Google Cloud Datastore.
10. Explain Amazon SimpleDB along with its features.
11. What do you mean by multi-tenant cloud? Differentiate it with the single-tenant mechanism.
12. What is Parallel Computing? Explain the applications of parallel computing.
13. Describe the 'Self-Driving Database'. How can it impact the future?
14. **Write short notes:**

a. GFS and HDFS
b. Google Cloud Datastore
c. Multi-tenant cloud
d. Parallel computing

❑❑❑