

Hadoop Ecosystem (Continue)

Data Access Component

Hive

Hive is an ETL and data warehousing tool developed by Apache. Hive makes it easier to query and manage massive datasets that are stored in multiple locations. Hive supports the summarization, querying and analysis of unstructured and semi-structured data in the Hadoop Ecosystem. The read, write and update operations on massive datasets in a distributed environment are performed through Hive Query Language (HQL) that is similar to Structured Query Language (SQL) and translates queries into MapReduce jobs automatically.

Components of Hive

- **Hive command line interface:** To execute HQL commands Hive provides a command line interface.
- **JDBC/ODBC driver:** For database connectivity, Java Database Connectivity (JDBC) and Object Database Connectivity (ODBC) drivers are used.

Hive Features

Some of the key features of Hive are:

- High scalability
- Real-time processing as well as batch query processing, i.e. interactive query processing
- Compatibility with each and every SQL primitive data type
- Support for user-defined functions (UDF) along with pre-set functions
- Automatic translation of queries into map and reduce functions through HQL
- Efficient extensive sequential scans

Limitations of Hive

- Not effective where fast response times are crucial because queries may encounter unusually high delays (up to several minutes).
- Hive is a read-based system and not suited for jobs typically involving many write operations.

Apache Pig

Pig is a platform for analyzing large datasets; it is comprised of a high-level language for describing data analysis programs and infrastructure for evaluating these programs.

Pig consists of two parts: Pig Latin, a programming language, and the Pig runtime, an environment in which Pig Latin programs can be executed. The ability of Pig programming to readily manage parallel processes gives it the advantage of being useful for managing very large amounts of data.

Pig supports the following scalar data types: int, float, double, long, chararray and bytearray. Pig's advanced data types are Map, Tuple and Bag.

Pig Latin is a straightforward scripting language that is similar in structure to SQL. Users find Apache Pig very comfortable as 200 lines of MapReduce Java code can be written in around 10 lines of Pig Latin. However, at a deeper level, the compiler translates Pig Latin into MapReduce. Pig Latin can be thought of as a black box which can easily generate a sequence of MapReduce jobs in an abstract way.

Apache Drill

Apache Drill is an open-source application that can perform analysis on massive datasets while working in a distributed context. Its main strength is to “drill” into the variety of NoSQL databases and file systems with only a single query. It offers a remarkable scalability factor, allowing it to support millions of users and serve their query requests for large-scale data in petabytes and exabytes.

Apache Sqoop

Apache Sqoop is a popular tool for the bidirectional data transfer of large datasets between Hadoop and relational database servers. It is a data migration tool based upon a connector architecture to connect with different databases. Sqoop uses JDBC driver which supports plugins to provide connectivity to a variety of external systems. Sqoop provides parallel data transfer which makes it fast and cost-effective. Sqoop can be used to import/export data from a wide variety of data stores, including but not limited to the following:

- 
- HDFS
 - Hive
 - HBase
 - Hcatalog
 - Accumulo
 - Oracle
 - MySql
 - PostgreSQL
 - SQL Server
 - DB2

Apache Avro

Developed by Doug Cutting, Apache Avro is an open-source remote procedure call and data serialization framework. Avro enables the interchange of large datasets between programs written in any language.

For ease of understanding, Avro uses the JSON format for representing data definition and binary format to store data itself making it compact and efficient. Avro files include markers for splitting data into subsets suitable for MapReduce.

Features of Avro

The key features of the Apache Avro framework are (Apache Avro™ 1.11.1 Documentation):

- Rich data structures
- A compact, fast, binary data format
- Robust support for dynamically changing data schemas
- A container file, to store persistent data
- Simple integration with dynamic languages including compiled and scripting languages

Apache Mahout

Apache Mahout is a framework to facilitate the free implementation of distributed and scalable machine learning algorithms focused primarily on linear algebra. For the concise and clear expression of algorithms, instead of traditional Scala-like syntax, Apache Mahout provides mathematically expressive Scala Domain Specific Language (DSL) which is similar to R syntax (Mahout, 2014).

Mahout offers data scientists a ready-to-use framework for performing the fast and effective analysis of large volumes of data using features like:

- 
- Classification (using methods like distributed naive Bayes and complementary naive Bayes classification)
 - Clustering (methods like k-means, fuzzy k-means, Canopy, Dirichlet and Mean-Shift)
 - Recommendation
 - Distributed fitness function capabilities for evolutionary programming
 - Matrix and vector libraries

Data Management Component

ZooKeeper

Developed by the Apache Software Foundation, ZooKeeper is an open-source, centralized service that is responsible for coordinating numerous distributed services like managing configuration information, naming, distributed synchronization and delivering group services.

Before the introduction of ZooKeeper, coordinating amongst the various Hadoop Ecosystem services was an extremely challenging and time-consuming endeavor. Common configuration errors during data synchronization are just one example of the problems that have arisen in the past



due to poor service interactions. Even if the services are configured, any modifications made to the configurations of the services will make it tough to manage. Additionally, item identification and categorization were time-consuming. ZooKeeper was developed as a solution to the issues mentioned. Just like any other distributed file system, ZooKeeper has a hierarchal name space with the only difference being that it allows a file to also be a directory (ZooKeeper Project, 2022).

Working of Apache ZooKeeper

A straightforward client server architecture serves as the foundation for ZooKeeper's overall design. The clients are the nodes that make service requests to the server, and the server is the node that fulfills those service requests on behalf of the clients. It's possible to have more than one ZooKeeper server. When the services are first begun, a chief administrator is chosen from within this community of servers. Every client establishes a connection with exactly one server, and that server alone is responsible for handling all of the read operations. Once a server receives a request to "write," it is sent on to the leader, who will then poll the quorum to see if it agrees with the action. A decision on this request needs to be approved by a quorum, which is defined as an absolute majority of the nodes that are part of the ensemble. If a favorable response is received from the quorum, the "write" request is judged to have been successful. This means that ZooKeeper is best suited for distributed systems with fewer "write" requests and more "read" ones, as the former take longer to process.

ZNode

The nodes in a ZooKeeper tree are referred to as ZNode. The responsibility of ZNode is to keep track of the statistics of the structure as well as the version numbers for any data updates. Because ZooKeeper does not support partial read and write operations, the transactions are atomic.

Types of ZNodes

- **Persistent ZNodes:** This is the default ZNode. Only by making a call to Zookeeper's delete feature can it be removed from the system.
- **Ephemeral ZNodes:** These are the ZNodes with no child and get deleted along with the data stored when a session is ended or a connection is severed by the client.

Oozie

Apache Oozie is a scalable, reliable and extensible workflow scheduler to manage Apache Hadoop jobs (Apache Oozie, 2021). As part of its architecture, Oozie uses a web server and a database to keep track of its various jobs. The default web server used in Oozie is the open-source software Apache Tomcat. The Oozie server is a stateless web application, which means that it does not retain any information in memory relating to the user or the job. When Oozie fetches the task state from the database in order to process the request, all of this information is first saved in the SQL database. The Oozie client can interact with the server in three distinct ways: via the command line tool, the Java Client API or the HTTP REST API.

Ambari

Apache Ambari provides an intuitive, easy-to-use Hadoop management web UI backed by its RESTful APIs. The highly interactive dashboard of Ambari facilitates keeping track of the running applications and their status in the easiest way. The goal of Ambari is to provision, manage and monitor Apache Hadoop clusters in the most effective way (Ambari, 2021).

Features of Ambari

- Easy step-by-step guide for installation and user-friendly configuration.
- The Ambari Metrics System provides immediate insight into the health of Hadoop clusters.
- The Ambari Alert Framework provides emergency alerts/notifications when a node goes down, disk space is low, etc.
- Central management for starting, stopping and reconfiguring Hadoop services across the entire cluster.
- Interactive dashboard for monitoring and visualizing the progress and status of every application running over the Hadoop cluster.
- Authentication, authorization and auditing by installing Kerberos-based Hadoop clusters.
- Flexible and adaptive technology.

Apache Flume

Flume is a service that efficiently collects, aggregates and moves enormous amounts of log data. It is a distributed system that is dependable and always available. Flumes provides a reliable and tolerant big data handling system with a number of failover and recovery techniques. It employs a straightforward data model that is capable of being extended, which makes online analytical applications possible (Apache Flume™, 2022).

Advantages of Apache Flume

- Flume facilitates a highly scalable, very dependable and highly fault-tolerant service.
- Flume ensures that there is a consistent flow of data transfer; for example, if the pace of reading data rises, then the speed of writing data will likewise increase.
- Because of its capacity to interface with a wide variety of applications, including Kafka, HDFS and Thrift, it provides a flexible option to users for data ingestion.

Apache Spark

Apache Spark is an alternative framework to Hadoop. Apache Spark is a software framework that is used to process data in a distributed manner and is replacing MapReduce in many use cases.

As a data processing engine, Apache Spark is known for its speed and ability to temporarily store its findings in memory space. It may be utilized in a broad variety of settings. Spark has numerous different deployment options. Apache Spark supports the programming languages Java, Python, Scala and R, and there is native assistance for SQL, real-time data (streaming data), machine learning (ML) and graph processing (GP) (Apache Spark™, 2014).

EXAMPLE

Consider a jigsaw Puzzle Ravensburger Beneath the Sea (5000 pieces). Children above 14 years of age will assemble the pieces in order to solve the puzzle. What will be the effect on time intervals for solution in three situations, when 4, 100 and 200 children simultaneously attempt the solution.

SOLUTION

Let the time taken by a single child to solve the puzzle be T . Assume 4 children sit together and solve the puzzle by dividing the tasks. Each child assembles one-fourth part of the picture for which they pick the pieces from a common basket (Distributed computing and centralized data model).

Alternatively, each child assembles one-fourth part of the picture for which the pieces are distributed in four baskets. The child in case does not find a piece in his/her basket, then searches for it in another basket (Distributed databases and distributed computing tasks with data sharing model).

Partitioning of assembling jobs into four has an issue. A child may complete his/her part much later than the remaining children. Beneath-the-sea portion is too complex, while upper-depth-sea portion is just plain. The children combine all four parts and finally complete the puzzle. Each one has to look into the other three parts to find a match and complete the task. Time taken to solve the puzzle is $[T/4 + T_I(4) + T_c(4)]$, where $T_I(4)$ is the time taken in seeking from others the pieces not available to a child during intermediate phases, and $T_c(4)$ in combining the results of the four children. Scaling factor is slightly less than 4. The proposed distributed model works well.

Assume a second situation in which 100 children assemble their parts of 50 pieces each, and finally combine all 100 parts and complete the puzzle. Each child must seek a piece, not available with her/him during the intermediate phase. Combining also becomes difficult and a time-consuming exercise compared to the four children case because each child now matches the results with the remaining 99 counterparts to arrive at the final solution. The time taken to solve the puzzle is $[T/100 + T_I(100) + T_c(100)]$,

where $T_I(100)$ and $T_c(100)$ are the time taken in seeking pieces not available with the child and combining results of 100 children, respectively. Scaling is by factor less than 100. The distributed model has issues like sharing pieces, seeking pieces not available and combining issues. Issues are at the intermediate as well as at the end stages.

If 200 children attempt to solve the puzzle simultaneously at the same time then finally combining all 200 portions of the Beneath the Sea, the integration of 200 portions will be tedious and will be a far more time-consuming exercise than with 4 or 100. The time taken to solve the puzzle is $[T/200 + T_I(200) + T_c(200)]$, where $T_I(200)$ and $T_c(200)$ is the time taken in seeking the pieces not available and combining, respectively. Scaling up is by factor much less than 200 and may even be less than even 100. The distributed model with pieces sharing between the children is unsatisfactory because $T_I(200) + T_c(200) < T/200$.



Problem of inter-children interactions exponentially grows with the number of children in the proposed distributed model with seeking pieces in intermediate phases. Time T_I becomes significantly high.

Alternatively, the picture parts and corresponding pieces of each part distribute to each participating child distinctly (Distributed computing model with no data sharing). Time T_I taken in seeking a piece not available with him/her is zero. The time taken in joining the assembled picture portions is only at the end. Problem of inter-children interactions during solving the puzzle does not exist.

EXAMPLE

Consider a data storage for University students. Each student data, *stuData* which is in a file of size less than 64 MB ($1 \text{ MB} = 2^{20}\text{B}$). A data block stores the full file data for a student of *stuData_idN*, where $N = 1$ to 500.

- (i) How the files of each student will be distributed at a Hadoop cluster? How many student data can be stored at one cluster? Assume that each rack has two DataNodes for processing each of 64 GB ($1 \text{ GB} = 2^{30}\text{B}$) memory. Assume that cluster consists of 120 racks, and thus 240 DataNodes.
- (ii) What is the total memory capacity of the cluster in TB ($1 \text{ TB} = 2^{40}\text{B}$) and DataNodes in each rack?
- (iii) Show the distributed blocks for students with ID= 96 and 1025. Assume default replication in the DataNodes = 3.
- (iv) What shall be the changes when a *stuData* file size is 128 MB?

SOLUTION

(i) Data block default size is 64 MB. Each student's file size is less than 64MB. Therefore, for each student file one data block suffices. A data block is in a DataNode. Assume, for simplicity, each rack has two nodes each of memory capacity = 64 GB. Each node can thus store $64 \text{ GB} / 64 \text{ MB} = 1024$ data blocks = 1024 student files. Each rack can thus store $2 \times 64 \text{ GB} / 64 \text{ MB} = 2048$ data blocks = 2048 student files. Each data block default replicates three times in the DataNodes. Therefore, the number of students whose data can be stored in the cluster = number of racks multiplied by number of files divided by 3 = $120 \times 2048 / 3 = 81920$. Therefore, the maximum number of 81920 stuData_IDNfiles can be distributed per cluster, with $N = 1$ to 81920.

- (ii) Total memory capacity of the cluster = $120 \times 128 \text{ MB} = 15360 \text{ GB} = 15 \text{ TB}$. Total memory capacity of each DataNode in each rack = $1024 \times 64 \text{ MB} = 64 \text{ GB}$.
- (iii) Figure 2.3 shows a Hadoop cluster example, and the replication of data blocks in racks for two students of IDs 96 and 1025. Each stuData file stores at two data blocks, of capacity 64 MB each.
- (iv) Changes will be that each node will have half the number of data blocks.

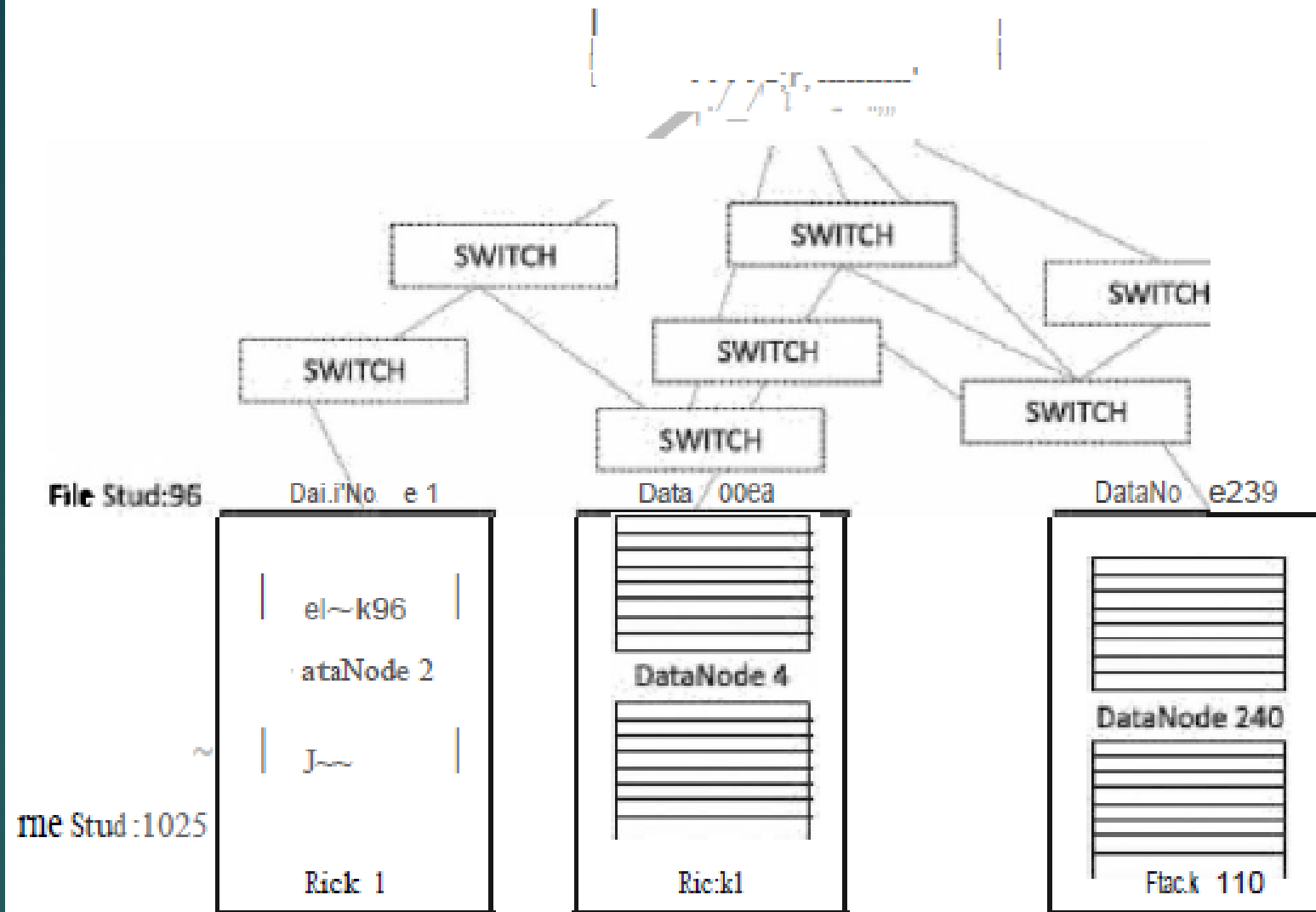


Figure 2.3 A Hadoop cluster example, and the replication of data blocks in racks for two students of IDs 96 and 1025

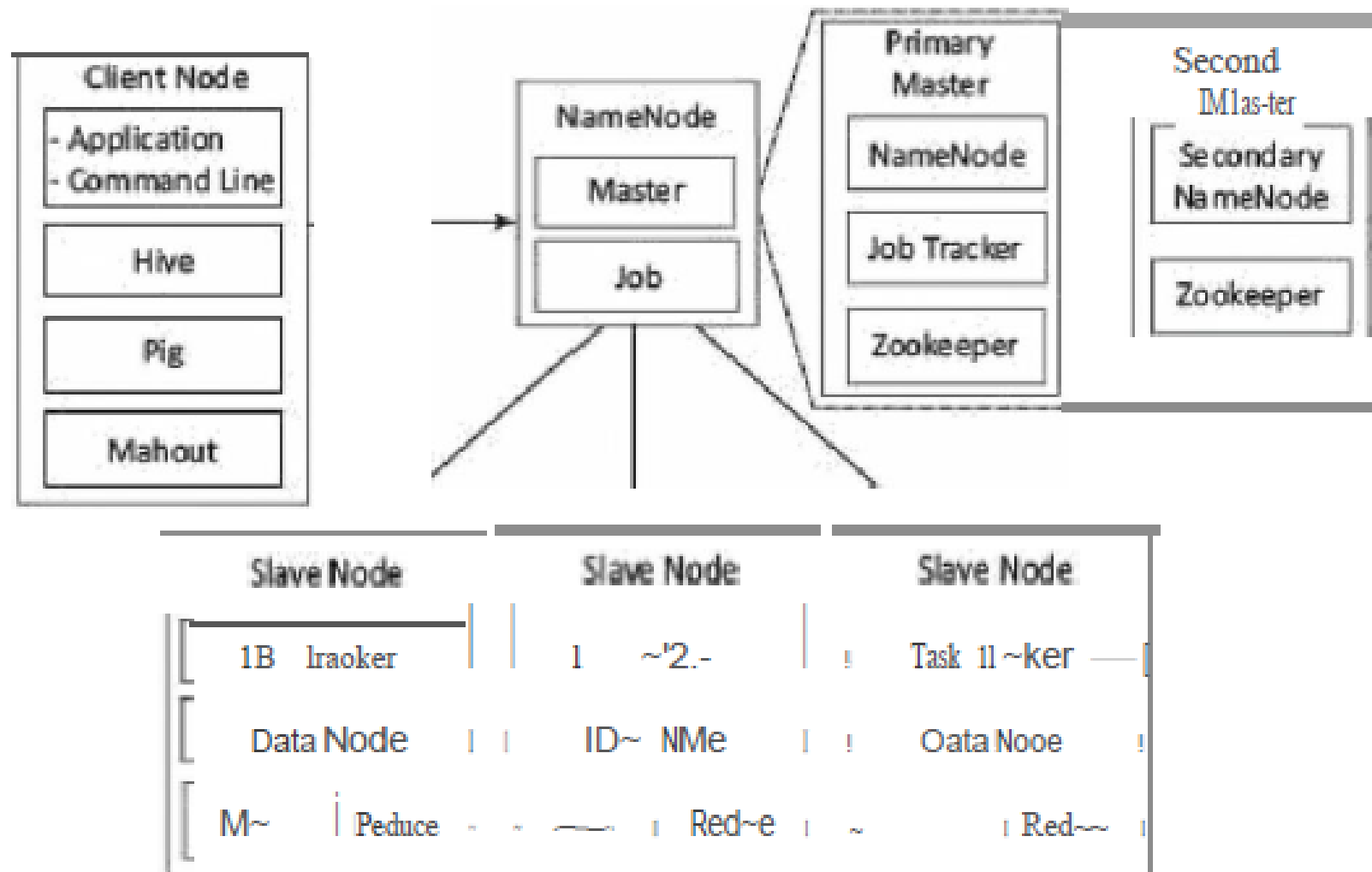


Figure 2.4 The client, master NameNode, MasterNodes and slave nodes

NoSQL Databases

Introduction

NoSQL databases, referred to as “No SQL” or “Not only SQL,” are non-tabular databases and store data differently than relational tables (IBM Cloud Education, 2019). The standard RDBMS system stores and retrieves information using SQL syntax for deeper analysis. This is done so that better choices can be made. Instead, data in a NoSQL database system can be stored in a wide variety of non-relational formats, such as structured, semi-structured, unstructured and polymorphic data, across several databases. A NoSQL database system is able to accommodate the storage of a variety of data. There is no requirement for a NoSQL database, also known as a non-relational data management system, to have a specified schema. NoSQL provides high scalability and availability, and because of this it is frequently utilized in environments dealing with large amounts of data as well as web applications that operate in real time, for example Twitter, Facebook and Google.

Features of NoSQL

RDBMS are the most widely used database management systems. However, with the exponential growth in the volume of data to be stored and managed, the reaction time of traditional RDBMS systems gradually slows down. It is possible that we will “scale up” our systems by adding additional hardware on top of what we already have in place in order to find a solution to this problem. This method comes at a very high financial cost.



Because of the massive volumes of data that Internet giants like Google, Facebook and Amazon, among others, are responsible for managing, the concept of NoSQL databases has received widespread acceptance. NoSQL supports the concept of “scaling out” where the burden of the database is distributed among several different servers when needed.

Some of the key features of NoSQL are as follows.

Non-Relational

NoSQL databases never adhere to the relational model. They don't produce tables with records that have a flat format and fixed column count. Data to be stored in NoSQL databases need not be normalized. Also, NoSQL doesn't support features like self-contained aggregates, BLOBs, referential integrity, specific query languages, query planners and BLOBs.

Schema Free

NoSQL databases can be defined as either having “schema-free” or “relaxed” schemas. It does not require any data structure explanation. NoSQL offers multiple file types for exchanging data that are all related to the same subject area.

Simple API

The most popular form of the NoSQL query language does not adhere to any particular industry standards databases that are enabled for the web and function as services that may be accessed via the Internet. Simple NoSQL APIs offer users storage and querying options for the data that has been provided, together with user interfaces that are basic and easy to understand and use. These APIs make it possible to do low-level activities such as processing data and selecting items from a list. Protocols are based on text and are frequently used in conjunction with HTTP, REST and JSON.

Distributed

The administration of large numbers of NoSQL databases can be performed in a decentralized approach which provides capabilities such as auto-scaling and failover. NoSQL is based on the “shared nothing” concept, i.e. the memory and disk spaces are not shared. As a direct consequence of this, there is a lower level of coordination and a greater degree of resource dispersion.

Difference between NoSQL and SQL

<i>Feature</i>	<i>SQL Databases</i>	<i>NoSQL Databases</i>
Concept	SQL databases are relational databases used for structured data	NoSQL databases are non-relational databases used for storing unstructured data
Schema	SQL databases have a predefined schema and use structured query language	NoSQL databases have dynamic schemas suitable for unstructured data
Scalability	SQL databases are vertically scalable, i.e. handling the load by scale-up with a larger server	NoSQL databases are horizontally scalable, i.e. scale-out across commodity servers
Structure	SQL databases are table-based with fixed rows and columns	NoSQL databases are document, key-value, graph or wide-column stores

Preferable use	SQL databases are better for multi-row transactions	NoSQL is better for unstructured data like documents or JSON
Multi-record ACID transactions	Supported	Most NoSQL databases do not support multi-record ACID transactions. However, some – like MongoDB – do
Performance	Slow query processing compared to NoSQL	Generally, queries in NoSQL databases are faster than SQL databases
Joins	Required to perform various query operations	Joins are typically not required in NoSQL databases

Types of NoSQL Databases

Key-value pair databases, column-oriented databases, graph-based databases and document-oriented databases are the primary varieties of NoSQL databases. Every category has its own specific traits and requirements to fulfill. None of the databases presented here is preferable to the others when it comes to resolving all issues. Users need to select a database in accordance with the requirements of the product.

Types of NoSQL Databases

- Key-value pair based
- Column-oriented databases
- Document-oriented databases
- Graph-based

Key-Value Pair Based Storage

Data is kept as pairs of keys and values. It is designed to handle large amounts of data and heavy loads. Key-value pair storage databases store data as a hash table in which each key is unique and the value can be a JSON object, BLOB, text, etc. In a sense, a key-value store is like a relational database with only two columns: the key or attribute name (such as “P-Id”) and the value (such as “1023”) as shown in Table 5.2.

Table 5.2 Example of Key-Value Store

<i>Key</i>	<i>Value</i>
P-Id	1023
P-name	'abc'
Weight	153 kg
Height	143 cm



This kind of NoSQL database can be used to store various data types, including collections, dictionaries, associative arrays and more. With the assistance of key-value stores, programmers can easily store data that does not have a schema. The contents of shopping carts are the primary beneficiaries of their utilization.

Key-value stores are primarily used for handling large amounts of data with simple lookup queries.

Column-Oriented Databases

The structure of column-oriented databases is based on Google's BigTable concept. In contrast to relational databases where data is stored in rows and accessed in a row-by-row fashion, a column-oriented database is organized as a set of columns. Just like key-value stores, column-oriented databases have some basic NoSQL structure while also preserving a lot of flexibility. In a single table column names and formatting can differ from row to row and each column is stored separately on disk. This means that when you want to run analytics on a small number of columns, you can read those columns directly without consuming memory with the unwanted data. The information contained in each column is processed individually and hence column-oriented databases can quickly aggregate the value of a given column. Columns are often of the same type and benefit from more efficient compression, making reads even faster. These characteristics of column-oriented databases make it very useful for analyzing large amounts of data with predictable query patterns.

Table 5.3 Column-Oriented Database Structure

	<i>Column Family A</i>		<i>Column Family B</i>		
Row key 1	Col a	Col b	Col c	Col d	Col e
	Val 1a	Val 1b	Val 1c	Val 1d	Val 1e
	Timestamp	Timestamp	Timestamp	Timestamp	timestamp
Row key 2	Col a	Col b	Col c		Col f
	Val 2a	Val 2b	Val 2c		Val 2f
	Timestamp	Timestamp	Timestamp		timestamp

- Row: Each row contains a unique row identifier to identify the column data. Every row can have a different number of columns from other rows.
- Column: Each column is contained in its row. A column consists of a name value pair along with a timestamp of its creation date.
- Column family: A column family consists of multiple rows.

Table 5.4 Example of Column-Oriented Database

	<i>Orders</i>		<i>Customers</i>		
Row key 1	Order Id	Order value	Name	Email	Age
	1002	680/ Rs	Bob	bob@123. com	22
	Timestamp	Timestamp	Timestamp	Timestamp	timestamp
Row key 2	Order Id	Order value	Name		Contact
	2043	1285/ Rs	Alice		982345615
	Timestamp	Timestamp	Timestamp		timestamp

Document-Oriented Databases

Document-oriented databases are used to store semi-structured data and descriptions of that data in document format. A document-oriented database stores data in JavaScript Object Notation (JSON), Binary JavaScript Object Notation (BSON) or XML. JSON and BSON support the embedding of documents and arrays within other documents and arrays.

In a document-oriented database, documents can be nested and specific elements can be indexed for faster querying. Document-oriented databases are widespread among developers because they have the flexibility to restructure their documents as per their business needs. This flexibility speeds development because, in effect, data becomes like code and is under the control of developers whereas, in SQL databases, to change the structure of a database, database administrators' intervention may be required. Figure 5.1 shows an example of a document-oriented database.

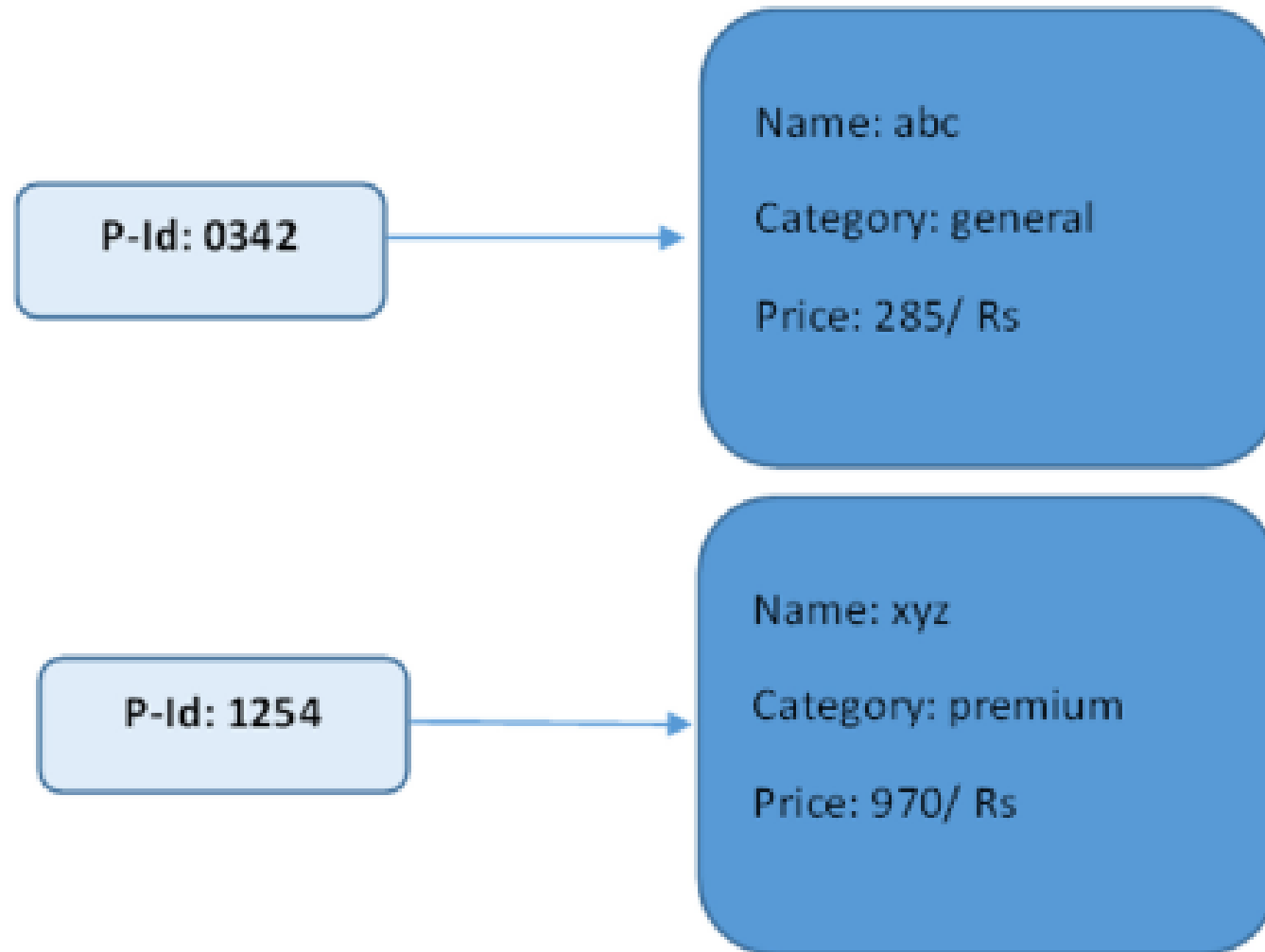


Figure 5.1 Example of document-oriented database.

Graph-Based Databases

A graph-based database uses graph structures to store data elements and to define the relationships between stored data elements. Graph databases are useful for identifying patterns in unstructured and semi-structured data content. Each data element to be stored is depicted as a node in the graph, and the connections that exist between it and other elements are represented as edges as shown in Figure 5.2. A unique identifier has been assigned to every node and edge in the network. A graph-oriented database is optimized to describe



and explore the connections between data elements, overcoming the overhead associated with joining multiple tables in SQL databases. Because graph-oriented databases store the relationships between data elements, they can support richer representations of data relationships. Also, graph-oriented data models can be augmented easily as they don't follow strict schemas unlike relational models.

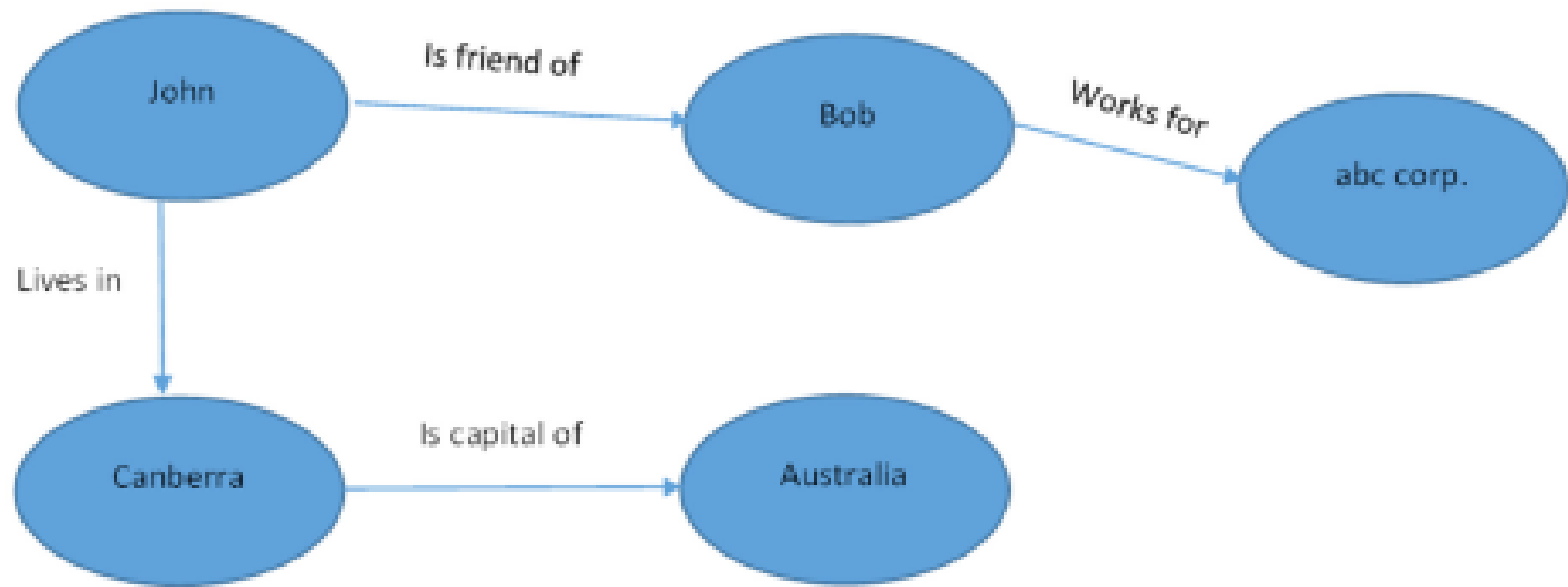


Figure 5.2 Graph-oriented data store.

Summary of NoSQL Databases

The NoSQL databases provide an efficient way of handling large volumes of unstructured data in a cloud or distributed environment. These databases don't follow strict schemas like relational databases and provide much flexibility to store data and analyze it as per the business needs. The NoSQL databases can be broadly classified into four types which are summarized in Table 5.5.

Table 5.5 Summary of Different Types of NoSQL Databases

<i>Parameter</i>	<i>Key-Value Pair Based Databases</i>	<i>Column-Oriented Databases</i>	<i>Document-Oriented Databases</i>	<i>Graphs-Based Databases</i>
Structure	Data is stored in a simple structure of a unique key and its associated value	Data is stored in tables where a row key can link to a large number of columns	Stores data in document-like structures which can be encoded in JASON-like formats	Data elements are stored as a node in a graph and the relations between different data elements are represented through edges of the graph
Use case	Session management, application logs, managing shopping carts, recording click stream data	Recommendation engines, catalogs, fraud detection, event-logging	Effective in content management and monitoring mobile and web applications, web analytics, e-commerce	Social media platforms, reservation systems or customer relationship management
Popular databases	Aerospike, DynamoDB, Redis, Riak	Accumulo, Amazon SimpleDB, Cassandra, HBase, Hypertable	Couchbase Server, CouchDB, MarkLogic, MongoDB	AllegroGraph, IBM Graph, Amazon Neptune, ArangoDB, Neo4j

BASE Model of NoSQL

Although some of the NoSQL databases like Apache's CouchDB or IBM's Db2 follow traditional atomicity, consistency, isolation and durability (ACID) compliance at a certain level, most of the NoSQL databases are designed not to strictly follow the ACID rules and focus on availability even in the case of frequent failures.