Chapter 1: Big data

Q1. What are three V's of Big data? Give two examples of big data case studies. Indicate which V's are satisfied by these case studies.

The three Vs of Big data are Velocity, Volume and Variety

i. Volume

- The exponential growth in the data storage as the data is now more than text data.Bhate
- The data can be found in the format of videos, music and large images on our social media channels.
- It is very common to have Terabytes and Petabytes of the storage system for enterprises.
- As the database grows the applications and architecture built to support the data needs to be reevaluated quite often.
- Sometimes the same data is re-evaluated with multiple angles and even though the original data is the same the new found intelligence creates an explosion of the data.
- The big volume indeed represents Big Data.

ii. Velocity

- The data growth and social media explosion have changed how we look at the data.
- There was a time when we used to believe that data of yesterday is recent.
- The matter of fact newspapers are still following that logic.
- However, news channels and radios have changed how fast we receive the news.
- Today, people reply on social media to update them with the latest happening. On social media sometimes a few seconds old messages (a tweet, status updates etc.) is not something that interests users.
- They often discard old messages and pay attention to recent updates. The data movement is now almost real time and the update window has reduced to fractions of the seconds.
- This high velocity data represents Big Data.

iii. Variety

- Data can be stored in multiple formats. For example database, excel, csv, access or for the matter of the fact, it can be stored in a simple text file.

- Sometimes the data is not even in the traditional format as we assume, it may be in the form of video, SMS, pdf or something we might have not thought about. It is the need of the organization to arrange it and make it meaningful.
- It will be easy to do so if we have data in the same format, however it is not the case most of the time. The real world has data in many different formats and that is the challenge we need to overcome with Big Data. This variety of the data represent Big Data

Q2. Short note on Big Data and analytics.

The technology research firm **Gartner** defines Big Data as diverse, high-volume, high-velocity information assets that require new forms of processing to enable enhanced decision making, insight discovery, and process optimization.

Big Data Institute defines Big Data as vast data sets that:

- Exhibit variety;
- Include structured, unstructured, and semi-structured data;
- Are generated at high velocity with an uncertain pattern;
- Do not fit neatly into traditional, structured, relational databases
- Can be captured, processed, transformed, and analyzed in a reasonable amount of time only by sophisticated information systems.

Big Data generally consists of the following.

Traditional enterprise data: Examples are customer information from customer relationship management systems, transactional enterprise resource planning data, Web store transactions, operations data, and general ledger data.

Machine-generated/sensor data: Examples are smart meters; manufacturing sensors; sensors integrated into smartphones, automobiles, airplane engines, and industrial machines; equipment logs; and trading systems data.

Social data: Examples are customer feedback comments; microblogging sites such as Twitter; and social media sites such as Facebook, YouTube, and LinkedIn.

Images captured by billions of devices located throughout the world, from digital cameras and camera phones to medical scanners and security cameras.

Q3. Compare Big Data and Traditional data warehouse approach.

S.NO.	BIG DATA	DATA WAREHOUSE

1.	Big data is the data which is in enormous form on which technologies can be applied.	Data warehouse is the collection of historical data from different operations in an enterprise.
2.	Big data is a technology to store and manage large amount of data.	Data warehouse is an architecture used to organize the data.
3.	It takes structured, non-structured or semi-structured data as an input.	It only takes structured data as an input.
4.	Big data does processing by using distributed file system.	Data warehouse doesn’t use distributed file system for processing.
5.	Big data doesn’t follow any SQL queries to fetch data from databas ine.	In data warehouse we use SQL queries to fetch data from relational databases.
6.	Apache Hadoop can be used to handle enormous amount of data.	Data warehouse cannot be used to handle enormous amount of data.

7.	When new data is added, the changes in data are stored in the form of a file which is represented by a table.	When new data is added, the changes in data do not directly impact the data warehouse.
8.	Big data doesn’t require efficient management techniques as compared to data warehouse.	Data warehouse requires more efficient management techniques as the data is collected from different departments of the enterprise.

Chapter 2 and 4 : Hadoop and Map Reduce

Q1. What is Hadoop? What are the goals and Assumptions? What are Hadoop limitations?

Hadoop is an open source,Java-based programming framework that supports the processing and storage of extremely large data sets in a distributed computing environment. It is part of the Apache project sponsored by the Apache Software Foundation. In 2008 Yahoo released Hadoop as an open source project. Today it is a framework by non-profit organization (ASF) Apache Software Foundation. Hadoop is a framework that allows storing and distributed processing of big data across clusters of Commodity hardware using a single programming model.

Goals:

1. Scalable: It can scale up to 1 to thousands of machines.
2. Fault Tolerance It is designed to detect and handle failures at the application layer.
3. Economical: Based on commodity hardware.
4. Handle hardware failures:It is delivering a highly available service on the top of a cluster of computers.

Hadoop does the following tasks.

1. Massive Data storage on clusters of commodity hardware
2. Fast processing

Assumptions..

1. Huge data- Large Data Set,
2. Hardware may fail,
3. Portability Across Heterogeneous Hardware and Software Platforms
4. Streaming Data Access-(Batch Processing) so high throughput and low latency,
5. Simple Coherency Model-data is write once and read multiple.
6. Moving Computation is Cheaper than Moving Data
7. Supports tens of millions of files in a single instance.

Limitations

1. HDFS can not be directly mounted by an existing OS. Getting data in and out from existing OS is inconvenient.

2. File Access can be achieved through

1. Native Java API through C++/Ruby/Python etc

2. Command line interface

3. HDFS-UI web app over HTTP

3. Security Concerns: Hadoop security model is disabled by default. It does not provide encryption at storage and network level. So Govt. agencies and others do not prefer to keep their data in hadoop framework.

4. Vulnerable by nature: Framework is entirely written in java. A language most widely used by cyber criminals.

5. Not fit for small data: Due to high capacity design, HDFS lacks the ability to efficiently support the random reading of small files. It is not recommended for organizations with small data.

6. Potential stability issues: Open-source platform that is created by many developers. While improvements are constantly made, it has stability issues. Need to make sure that organizations are running the stable version.

7. General limitations: Google mentioned that Hadoop may not be the single solution for big data.

8. Cloud Dataflow .. Google's solution, Apache Flume has the ability to improve the efficiency and reliability of data, Collection, Integration and aggregation of data

Q2. Explain core Hadoop components.

1. Hadoop Common Libraries

The necessary Java files and scripts required to start Hadoop. Libraries and utilities needed by other hadoop modules. Hadoop uses the Hadoop Common as a kernel to provide the framework's essential libraries.

Provides filesystem and OS level abstractions .

JAR files + Scripts Needed to start Hadoop.

JRE 1.6 and Higher version in the machine : to start Hadoop.

SSH (Secure Shell) : Standard startup and shutdown scripts uses SSH to setup between the nodes in the cluster.

Java Libraries and Utilities required by other Hadoop modules like Hbase/Hive etc.

2. HDFS: It Stores data in terms of chunks (may be 64 or 128MB) across different machines.

Distributed file system (DFS) design.

• Hadoop provides a command interface to interact with HDFS.

• It runs on commodity hardware. It is designed using low-cost hardware.

• HDFS is highly fault tolerant

• HDFS stores huge data across multiple machines. (distributed storage)

• These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. Thus provides reliable and fast acccess.

• HDFS also makes applications available to parallel processing.

• The built-in servers of namenode and datanode help users to easily check the status of cluster.

• Streaming access to file system data.

• HDFS provides file permissions and authentication.

3. YARN

The YARN or Yet Another Resource Negotiator is the update to Hadoop since its second version. It is responsible for Resource management and Job Scheduling. Yarn comprises of the following components:

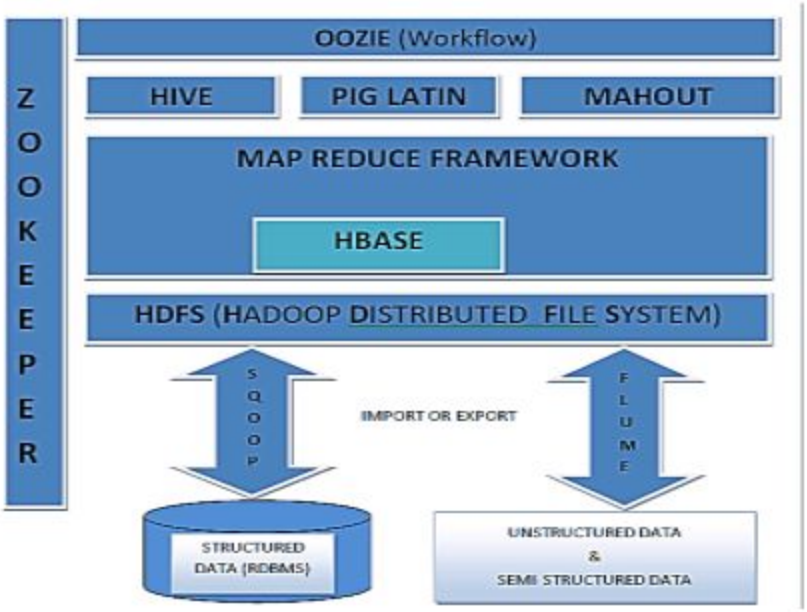
- Resource Manager: It is the core component of Yarn and is considered as the *Master,* responsible for providing generic and flexible frameworks to administer the computing resources in a *Hadoop Cluster.*
-
- Node Manager: It is the *Slave* and it serves the *ResourceManager.* Node Manager is assigned to all the Nodes in a Cluster. The main responsibility of the Node Manager is to monitor the Status of the Container and App Manager.
- App Manager: It manages data processing in the Container and requests the Container resources from the Resource Manager.
- Container: Container is where the actual data processing takes place.

4. MapReduce:

- Hadoop MapReduce is a software framework. Helps in writing applications. Helps in parallel processing of Big Data on large clusters.
- Basically comprises of three sequential processes/tasks- Map, Shuffle, Reduce
- The MapReduce framework takes care of scheduling and monitoring of tasks. It also re-executes the failed tasks with help of JobTracker.
- The MapReduce framework consists of a single master JobTracker and one or many slave TaskTracker per cluster-node
- The Map Task: This is the first task, which takes input data on which desired function is executed to convert it into a set of data, (intermediate key/value pairs). It is a parallel and share nothing processing of input.
- Shuffle Task: Shuffle phase is introduced and managed internally by the framework
- The Reduce Task: Input to Reduce is output of Map Task Grouped by key and Values combined /aggregated together as per Reduce algorithm provided by user.. Typically both the input and the output are stored in a file-system.

Q3. Draw Hadoop Ecosystem components and explain the functionality of each of them.

Hadoop is a framework which deals with Big Data but unlike any other frame work it's not a simple framework, it has its own family for processing different thing which is tied up in one umbrella called as Hadoop Ecosystem.



HDFS (Hadoop Distributed File System)

HDFS is a main component of Hadoop and a technique to store the data in distributed manner in order to compute fast. HDFS saves data in a block of 64MB(default) or 128 MB in size which is logical splitting of data in a Datanode (physical storage of data) in Hadoop cluster(formation of

several Datanode which is a collection commodity hardware connected through single network). All information about data splits in data node known as metadata is captured in Namenode which is again a part of HDFS.

MapReduce Framework

It is another main component of Hadoop and a method of programming in a distributed data stored in a HDFS. We can write Map reduce program by using any language like JAVA, C++ PIPEs, PYTHON, RUBY etc. By name only Map Reduce gives its functionality Map will do mapping of logic into data (distributed in HDFS) and once computation is over reducer will collect the result of Map to generate final output result of MapReduce. MapReduce Program can be applied to any type of data whether Structured or Unstructured stored in HDFS. Example - word count using MapReduce.

HBASE

Hadoop Database or HBASE is a non-relational (NoSQL) database that runs on top of HDFS. HBASE was created for large table which have billions of rows and millions of columns with fault tolerance capability and horizontal scalability and based on Google Big Table. Hadoop can perform only batch processing, and data will be accessed only in a sequential manner for random access of huge data HBASE is used.

Hive

Many programmers and analyst are more comfortable with Structured Query Language than Java or any other programming language for which Hive is created by Facebook and later donated to Apache foundation. Hive mainly deals with structured data which is stored in HDFS with a Query Language similar to SQL and known as HQL (Hive Query Language). Hive also run Map reduce program in a backend to process data in HDFS but here programmer has not worry about that backend MapReduce job it will look similar to SQL and result will be displayed on console.

Pig

Similar to HIVE, PIG also deals with structured data using PIG LATIN language. PIG as originally developed at Yahoo to answer similar need to HIVE. It is an alternative provided to programmer who loves scripting and don't want to use Java/Python or SQL to process data. A Pig Latin program is made up of a series of operations, or transformations, that are applied to the input data which runs MapReduce program in backend to produce output.

Mahout

Mahout is an open source machine learning library from Apache written in java. The algorithms it implements fall under the broad umbrella of machine learning or collective intelligence. This can mean many things, but at the moment for Mahout it means primarily recommender engines (collaborative filtering), clustering, and classification. Mahout aims to be the machine learning tool of choice when the collection of data to be processed is very large, perhaps far too large for a single machine. In its current incarnation, these scalable machine learning implementations in Mahout are written in Java, and some portions are built upon Apache's Hadoop distributed computation project.

Oozie

It is a workflow scheduler system to manage hadoop jobs. It is a server-based Workflow Engine specialized in running workflow jobs with actions that run Hadoop MapReduce and Pig jobs. Oozie is implemented as a Java Web-Application that runs in a Java Servlet-Container. Hadoop basically deals with bigdata and when some programmer wants to run many job in a sequential manner like output of job A will be input to Job B and similarly output of job B is input to job C and final output will be output of job C. To automate this sequence we need a workflow and to execute same we need engine for which OOZIE is used.

Zookeeper

Writing distributed applications is difficult because partial failure may occur between nodes to overcome this Apache Zookeeper has been developed by maintaining an open-source server which enables highly reliable distributed coordination. ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. In case of any partial failure clients can connect to any node and be assured that they will receive the correct, up-to-date information.

Q4. **Explain HDFS architecture with diagram (techmax page 2-5)**

The Hadoop Distributed File System (HDFS) is designed to provide a fault-tolerant file system designed to run on commodity hardware. The primary objective of HDFS is to store data reliably even in the presence of failures including Name Node failures, Data Node failures and network partitions.

HDFS uses a master/slave architecture in which one device (the master) controls one or more other devices (the slaves). The HDFS cluster consists of a single Name Node and a master server manages the file system namespace and regulates access to files.

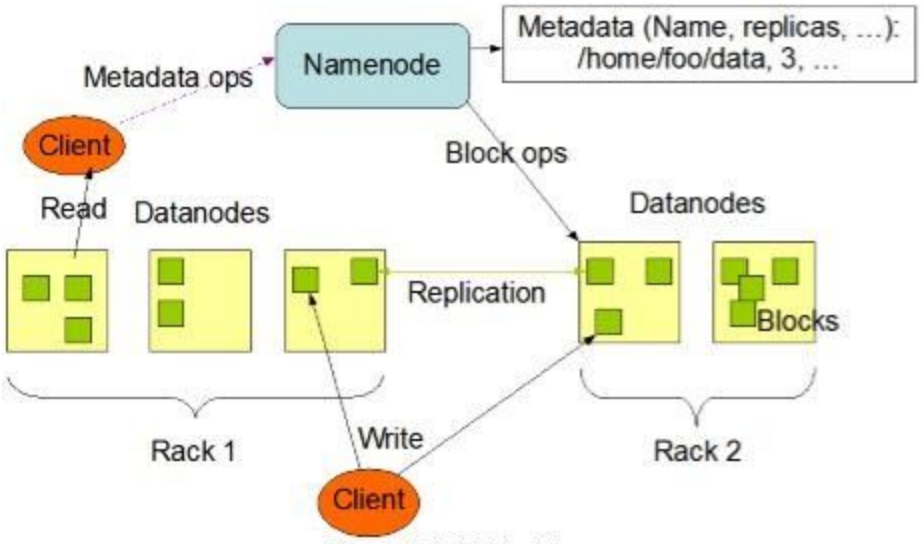


Figure: HDFS Architecture

The main components of HDFS are as described below:

NameNode and DataNodes:

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

The NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software. Usage of the highly portable Java language means that HDFS can be deployed on a wide range of machines. A typical deployment has a dedicated machine that runs only the NameNode software. Each of the other machines in the cluster runs one instance of the DataNode software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case. The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode.

The File System Namespace:

HDFS supports a traditional hierarchical file organization. A user or an application can create directories and store files inside these directories. The file system namespace hierarchy is similar to most other existing file systems; one can create and remove files, move a file from one directory to another, or rename a file. HDFS does not yet implement user quotas. HDFS does not support hard links or soft links. However, the HDFS architecture does not preclude implementing these features.

The NameNode maintains the file system namespace. Any change to the file system namespace or its properties is recorded by the NameNode. An application can specify the number of replicas of a file that should be maintained by HDFS. The number of copies of a file is called the replication factor of that file. This information is stored by the NameNode.

Data Replication:

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.

Q5. What is MapReduce? Explain how Map and Reduce Tasks work?

Hadoop MapReduce's programming model facilitates the processing of big data stored on HDFS.

By using the resources of multiple interconnected machines, MapReduce effectively handles a large amount of structured and unstructured data.

MapReduce assigns fragments of data across the nodes in a Hadoop cluster. The goal is to split a dataset into chunks and use an algorithm to process those chunks at the same time. The parallel processing on multiple machines greatly increases the speed of handling even petabytes of data.

As the name suggests, MapReduce works by processing input data in two stages – Map and Reduce. To demonstrate this, we will use a simple example with counting the number of occurrences of words in each document.

The final output we are looking for is: How many times the words Apache, Hadoop, Class, and Track appear in total in all documents.

For illustration purposes, the example environment consists of three nodes. The input contains six documents distributed across the cluster. We will keep it simple here, but in real circumstances, there is no limit. You can have thousands of servers and billions of documents.

1. First, in the map stage, the input data (the six documents) is split and distributed across the cluster (the three servers). In this case, each map task works on a split containing two documents. During mapping, there is no communication between the nodes. They perform independently.

2. Then, map tasks create a <key, value> pair for every word. These pairs show how many times a word occurs. A word is a key, and a value is its count. For example, one document contains three of four words we are looking for: Apache 7 times, Class 8 times, and Track 6 times. The key-value pairs in one map task output look like this:

<apache, 7>

<class, 8>

<track, 6>

This process is done in parallel tasks on all nodes for all documents and gives a unique output.

3. After input splitting and mapping completes, the outputs of every map task are shuffled. This is the first step of the Reduce stage. Since we are looking for the frequency of occurrence for four words, there are four parallel Reduce tasks. The reduce tasks can run on the same nodes as the map tasks, or they can run on any other node.

The shuffle step ensures the keys Apache, Hadoop, Class, and Track are sorted for the reduce step. This process groups the values by keys in the form of <key, value-list> pairs.

4. In the reduce step of the Reduce stage, each of the four tasks process a <key, value-list> to provide a final key-value pair. The reduce tasks also happen at the same time and work independently.

In our example from the diagram, the reduce tasks get the following individual results:

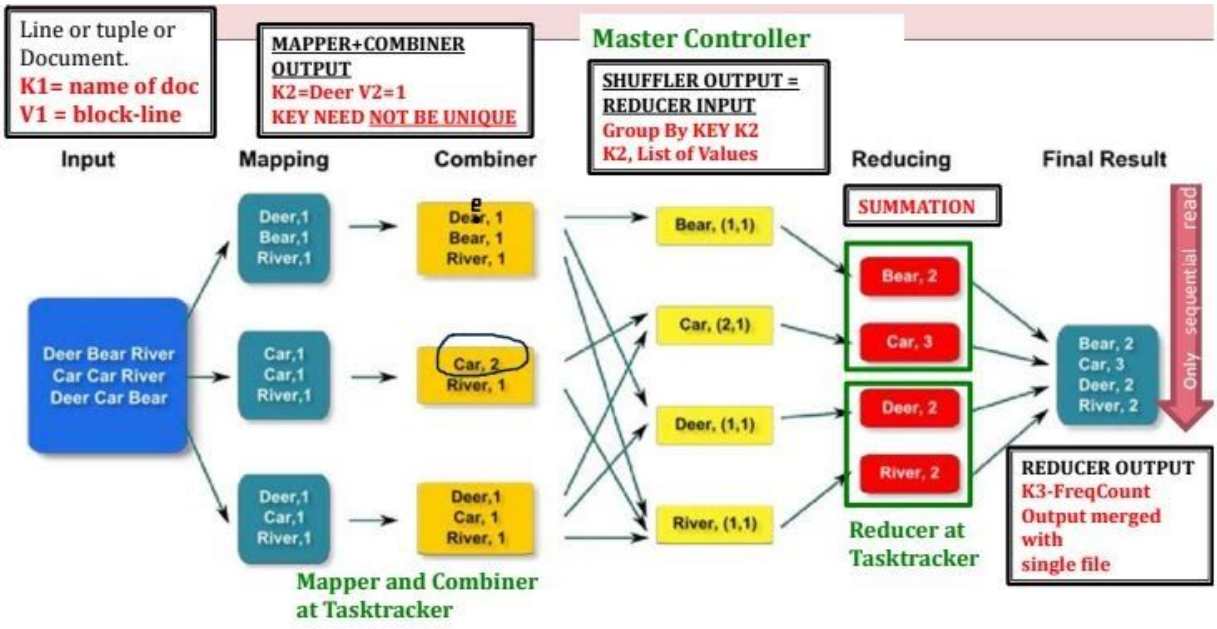
<apache, 22>

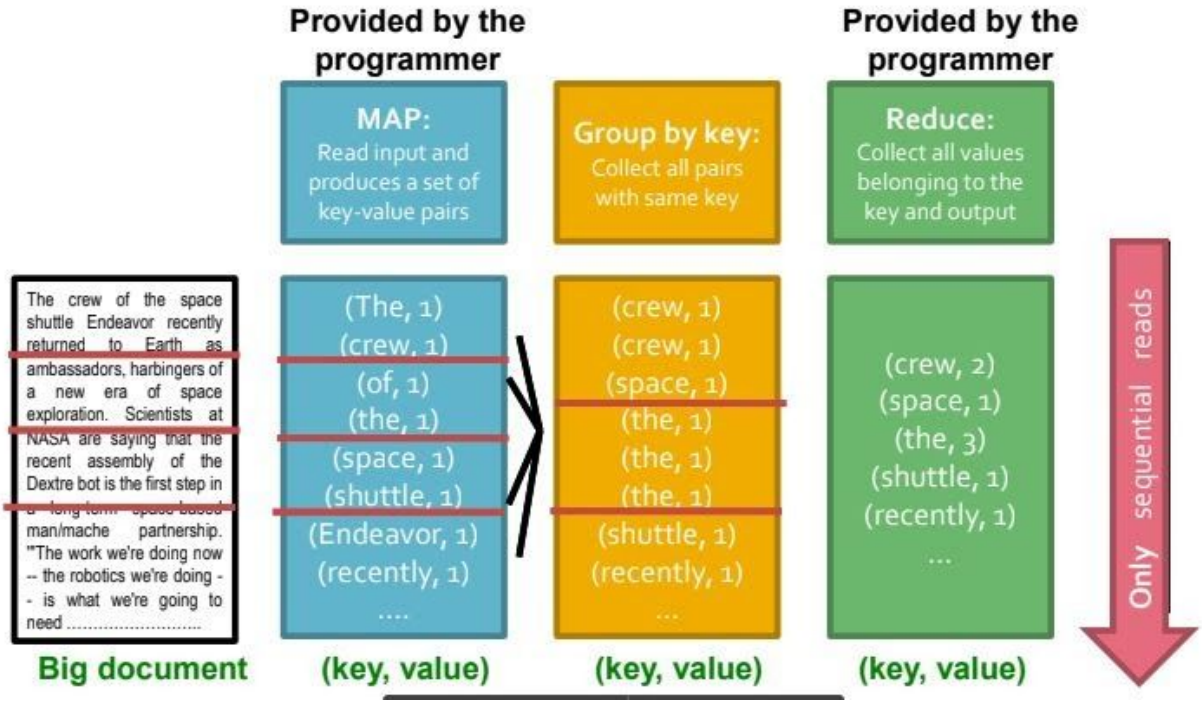
<hadoop, 20>

<class, 18>

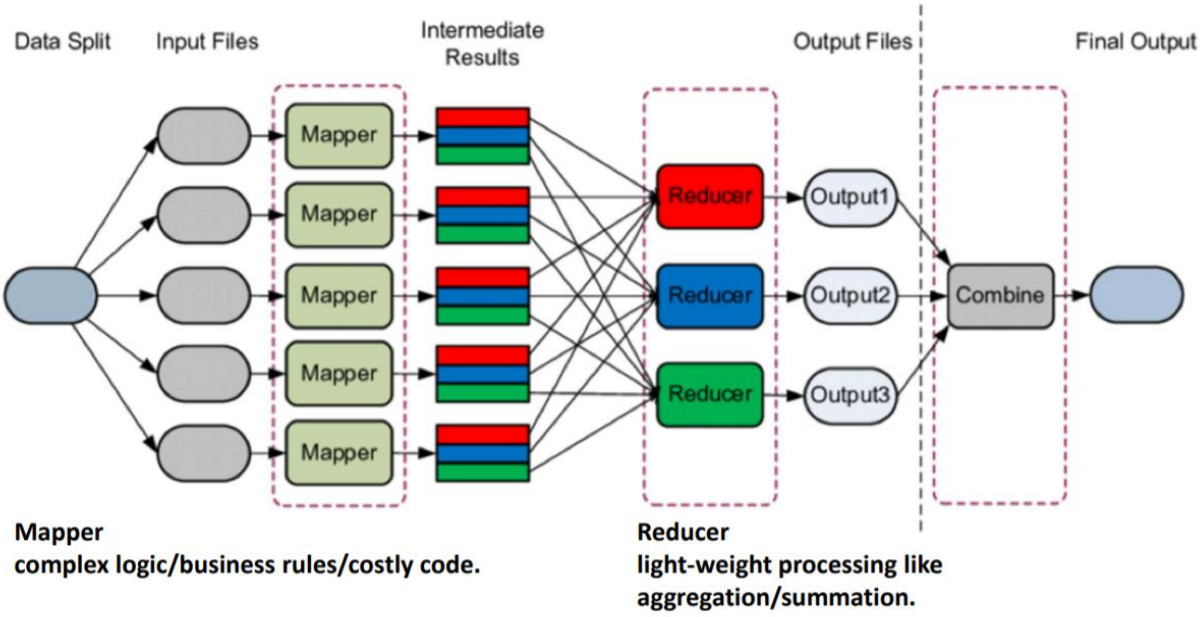
<track, 22>

Q6. Explain the process flow of the problem of word count in Hadoop with the help of Hadoop architecture diagram.

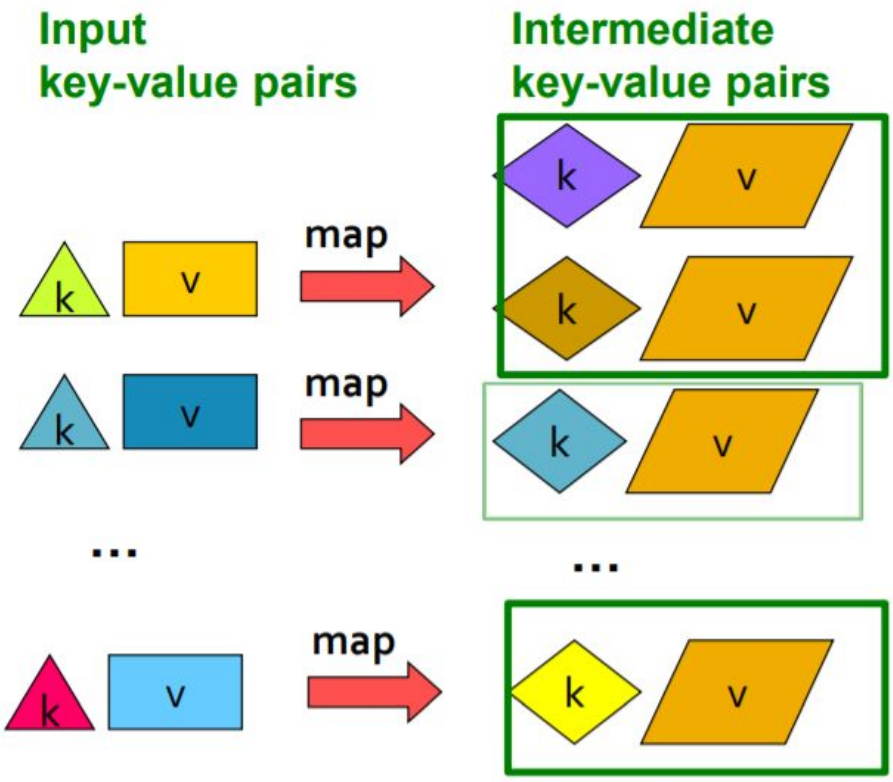




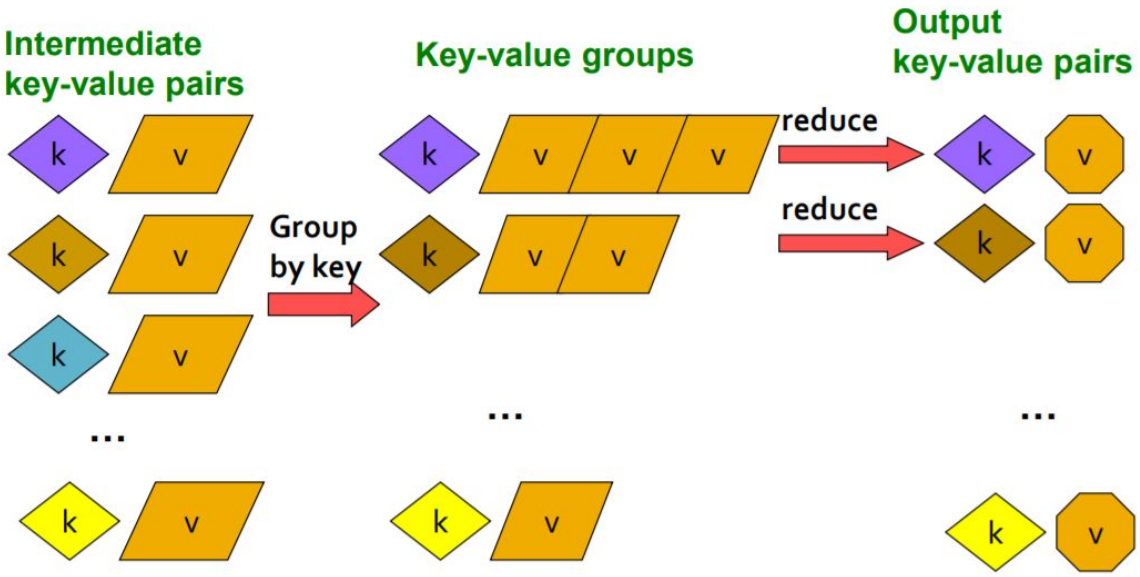
Q7. Explain with the help of diagram and psedudo code the working of word count problem solved using Map-reduce.



Mapping Step



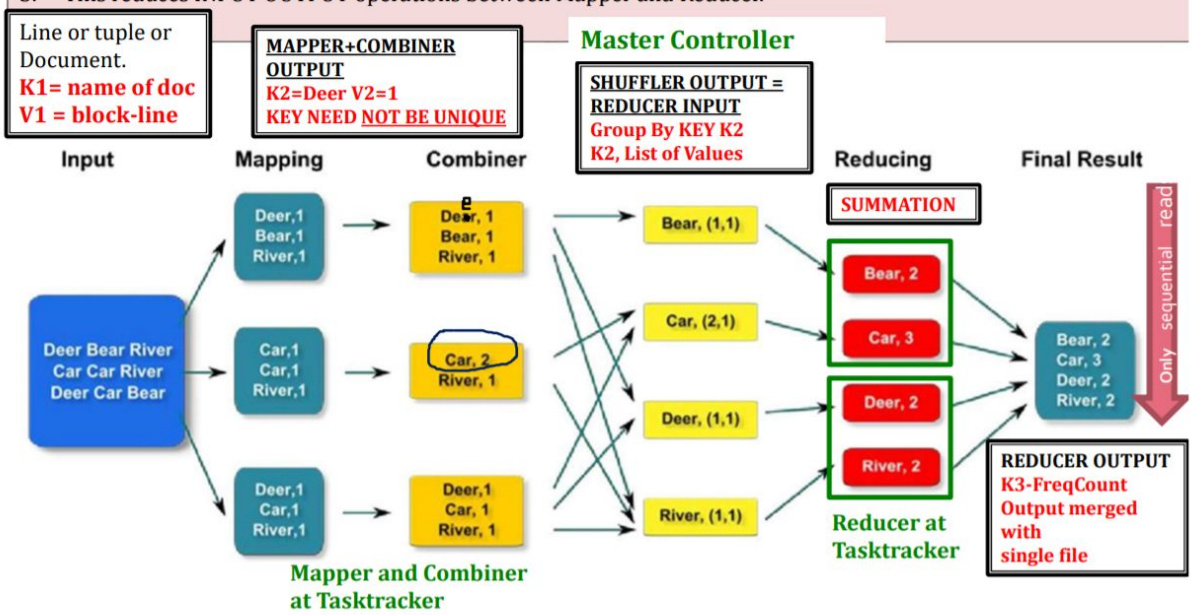
REducing Step

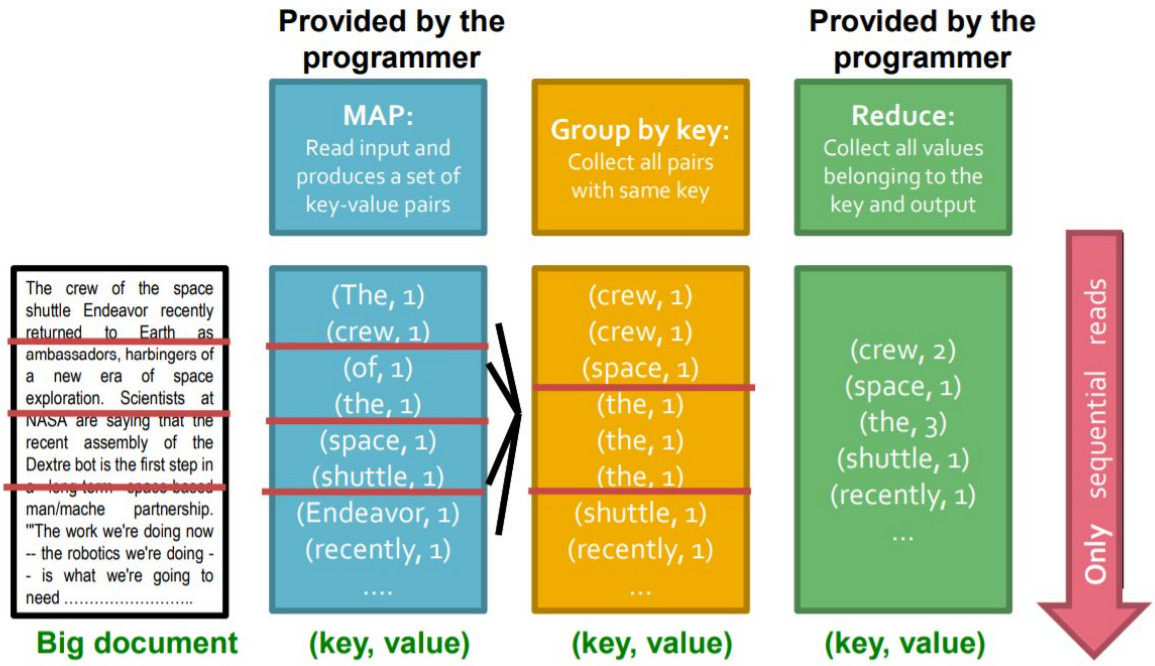


WordCount Exercise

COMBINER (Optional)

1. Some of the tasks of reducer can be assigned to combiner
2. Instead of sending all values to reducer, some value can be calculated at mapper side for same keys
3. This reduces INPUT OUTPUT operations between Mapper and Reducer.





PseudoCode:

map(key, value):

// key: document name; value: text of the document

for each word w in value:

    emit(w, 1)

reduce(key, values):

// key: a word; count: an iterator over values

result = 0

for each count v in values:

    result += v

    emit(key, result)

Q6. Describe the operations shuffle and sort in the map reduce framework? Explain with the help of one example.

Shuffling is the process by which it transfers mappers intermediate output to the reducer. Reducer gets 1 or more keys and associated values on the basis of reducers. The intermediated key value generated by mapper is sorted automatically by key. In Sort phase

merging and sorting of map output takes place. Shuffling and Sorting in Hadoop occurs simultaneously.

Shuffling in MapReduce

The process of transferring data from the mappers to reducers is shuffling. It is also the process by which the system performs the sort. Then it transfers the map output to the reducer as input. This is the reason shuffle phase is necessary for the reducers. Otherwise, they would not have any input (or input from every mapper). Since shuffling can start even before the map phase has finished. So this saves some time and completes the tasks in lesser time.

Sorting in MapReduce

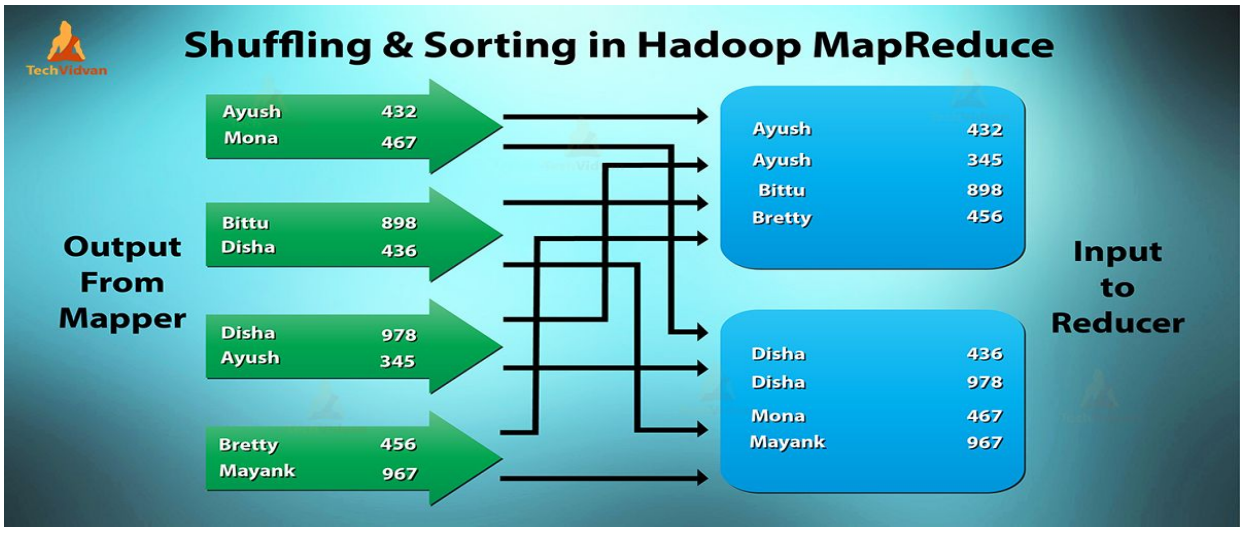
MapReduce Framework automatically sort the keys generated by the mapper. Thus, before starting of reducer, all intermediate key-value pairs get sorted by key and not by value. It does not sort values passed to each reducer. They can be in any order.

Sorting in a MapReduce job helps reducer to easily distinguish when a new reduce task should start. This saves time for the reducer. Reducer in MapReduce starts a new reduce task when the next key in the sorted input data is different than the previous. Each reduce task takes key value pairs as input and generates key-value pair as output.

The important thing to note is that shuffling and sorting in Hadoop MapReduce are will not take place at all if you specify zero reducers (setNumReduceTasks(0)). If reducer is zero, then the MapReduce job stops at the map phase. And the map phase does not include any kind of sorting (even the map phase is faster).

Secondary Sorting in MapReduce

If we want to sort reducer values, then we use a secondary sorting technique. This technique enables us to sort the values (in ascending or descending order) passed to each reducer.



6. **Explain Matrix Multiplication algorithm using Map-Reduce with example.**

Let A and B be the two matrices to be multiplied and the result be matrix C. Matrix A has dimensions L, M and matrix B has dimensions M, N. In the Map phase:

1. For each element (i,j) of I, emit $((i,k), A[i,j])$ for k in $1,\ldots, N$.
2. For each element (j,k) of B, emit $((i,k), B[j,k])$ for i in $1, \ldots, L$.

In the reduce phase, eIit

key = (i,k)

valuI = Sum_j (A[i,j] * B[j,k])

One reducer is used per output cell

Each reducer compItesSum$_j$ (A[i,j] * B[j,k])

The block diagram of MapReduce multiplication algorithm is shown in Fig. 4.3.

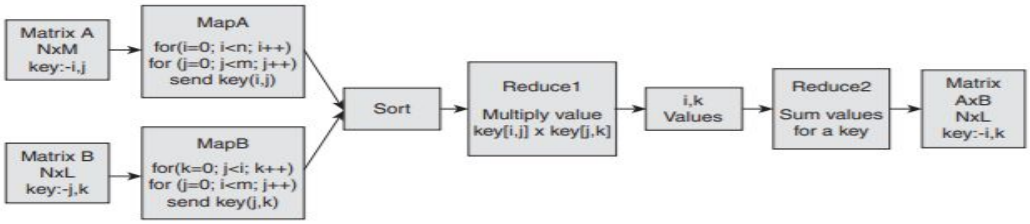


Figure 4.3 Matrix-multiplication by MapReduce.

Algorithm

Pseudo Code for Matrix-Vector Multiplication by MapReduce

```
map(key, valIe):
   for (i, j, a_ij) in value:
      emit(i, a_ij * v[j])
reduce(key, values):
   result = 0
   for value in values:
      result += value
         emit(key, result)
```

Q10. Show the Map Reduce implementation for the following two tasks using pseudocode.

I. Computing Group-by and aggregation of a -relational table.

4.3.7 Grouping and Aggression by MapReduce

The following steps show grouping and aggregation by MapReduce:

1. Grouping and aggregation can be performed in one MapReduce job.
2. Mapper extracts from each tuple values to group by and aggregate and emits them.
3. Reducer receives values to be aggregated that are already grouped and calculates an aggregation function.

Let R(A,B,C) be a relation to which apply the Irator $\gamma A,\theta(B)(R)$. Map will perform the grouping, while Reduce does the aggregation.

1. **The Map function:** For each tuple (a, b, c) produce the key–value pair (a, b).
2. **The Reduce function:** Each key "a" represents a group. Apply the aggregation operator θ to the list $[b_1, b_2, \ldots, b_n]$ of B-values associated with key "a". The output is the pair (a, x), where x is the result of applying θ to the list. For example, if θ is SUM, then $x = b_1 + b_2 + \cdots + b_n$, and I is MAX, then x is the largest of $b_1, b_2, \ldots, b_n$.

Algorithm

Pseudo Code for Grouping and Aggregation

```
map(key, value):
   for (a, b, c) in value:
      emit(a, b)

reduce(key, values):
   emit(key, theta(values))
```



```
In other words:
class Mapper
   method Map(null, tuple [value GroupBy,IlueAggregateBy, value ...])
      Emit(value GroupBy, value AggregateBy)
class Reducer
   methodRIce(value GroupBy, [v1, v2,...])
      Emit(vaIGroupBy, aggregate( [v1, v2,...] ) ) // aggregate() : sum(), max(),...
```

Note: If there are several grouping attributes, then the key is the list of the values of a tuple for all these attributes.

Mapper(M) → {(key, value)}
Sort ({(key, value)}) → group by "key"
Reducer ({"key, value_i}) → ("key, f(value_i))
Can repeat, constant number of rounds

If there is more than one aggregation, then the Reduce function applies each of them to the list of values associated with a given key and produces a tuple consisting of the key, including components for all grouping attributes if there is more than one, followed by the results of each of the aggregations.

II. Multiplication of two matrices

2.3.9 Matrix Multiplication

If M is a matrix with element m_{ij} in row i and column j, and N is a matrix with element n_{jk} in row j and column k, then the product $P = MN$ is the matrix P with element p_{ik} in row i and column k, where

$$p_{ik} = \sum_j m_{ij} n_{jk}$$

It is required that the number of columns of M equals the number of rows of N, so the sum over j makes sense.

We can think of a matrix as a relation with three attributes: the row number, the column number, and the value in that row and column. Thus, we could view matrix M as a relation $M(I, J, V)$, with tuples (i, j, m_{ij}), and we could view matrix N as a relation $N(J, K, W)$, with tuples (j, k, n_{jk}). As large matrices are often sparse (mostly 0's), and since we can omit the tuples for matrix elements that are 0, this relational representation is often a very good one for a large matrix. However, it is possible that i, j, and k are implicit in the position of a matrix element in the file that represents it, rather than written explicitly with the element itself. In that case, the Map function will have to be designed to construct the I, J, and K components of tuples from the position of the data.

The product MN is almost a natural join followed by grouping and aggregation. That is, the natural join of $M(I, J, V)$ and $N(J, K, W)$, having only attribute J in common, would produce tuples (i, j, k, v, w) from each tuple (i, j, v) in M and tuple (j, k, w) in N. This five-component tuple represents the pair of matrix elements (m_{ij}, n_{jk}). What we want instead is the product of these elements, that is, the four-component tuple $(i, j, k, v \times w)$, because that represents the product $m_{ij} n_{jk}$. Once we have this relation as the result of one MapReduce operation, we can perform grouping and aggregation, with I and K as the grouping attributes and the sum of $V \times W$ as the aggregation. That is, we can implement matrix multiplication as the cascade of two MapReduce operations, as follows. First:

The Map Function: For each matrix element m_{ij}, produce the key value pair $\big(j, (M, i, m_{ij})\big)$. Likewise, for each matrix element n_{jk}, produce the key value

pair $\big(j, (N, k, n_{jk})\big)$. Note that M and N in the values are not the matrices themselves. Rather they are names of the matrices or (as we mentioned for the similar Map function used for natural join) better, a bit indicating whether the element comes from M or N.

The Reduce Function: For each key j, examine its list of associated values. For each value that comes from M, say (M, i, m_{ij}), and each value that comes from N, say (N, k, n_{jk}), produce a key-value pair with key equal to (i, k) and value equal to the product of these elements, $m_{ij}n_{jk}$.

Now, we perform a grouping and aggregation by another MapReduce operation.

The Map Function: This function is just the identity. That is, for every input element with key (i, k) and value v, produce exactly this key-value pair.

The Reduce Function: For each key (i, k), produce the sum of the list of values associated with this key. The result is a pair $\big((i, k), v\big)$, where v is the value of the element in row i and column k of the matrix $P = MN$.

Q11. Show the Map Reduce implementation for the following two tasks using pseudocode.

III. Join of two relations with an example. (choose any answer u see fit)

4.3.6 Computing Natural Join by MapReduce

For doing Natural join, the relation R(A,B) with S(B,C), it is required to find tuples that agree on their B components, that is, the second component from tuples of R and the first component of tuples of S. Using the B-value of tuples from either relation as the key, the value will be the other component along with the name of the relation, so that the Reduce function can know where each tuple came from.

1. **The Map function:** For each tuple (a, b) of R, produce the key–value pair b, (R, a). For each tuple (b, c) of S, produce the key–value pair b, (S, c).
2. **The Reduce function:** Each key value b will be associated with a list of pairs that are either of the form (R, a) or (S, c). Construct all pairs consisting of one with first component R and the other with first component S, say (R, a) and (S, c). The output from this key and value list is a sequence of key–value pairs. The key is irrelevant. Each value is one of the triples (a, b, c) such that (R, a) and (S, c) are on the input list of values.

Algorithm

Pseudo Code for Natural Join

```
map(key, value):
  if key == R:
    for (a, b) in value:
      emit(b, (R, a))
```



```
  else:
    for (b, c) in value:
      emit(b, (S, c))

reduce(key, values):
  list_R = [a for (x, a) in values if x == R]
  list_S = [c for (x, c) in values if x == S]
  for a in list_R:
    for c in list_S:
      emit(key, (a, key, c))
```

IV. Multiplication of two matrices with one step Map Reduce.

2.3.10 Matrix Multiplication with One MapReduce Step

There often is more than one way to use MapReduce to solve a problem. You may wish to use only a single MapReduce pass to perform matrix multiplication $P = MN$. [5] It is possible to do so if we put more work into the two functions. Start by using the Map function to create the sets of matrix elements that are needed to compute each element of the answer P. Notice that an element of M or N contributes to many elements of the result, so one input element will be turned into many key-value pairs. The keys will be pairs (i, k), where i is a row of M and k is a column of N. Here is a synopsis of the Map and Reduce functions.

The Map Function: For each element m_{ij} of M, produce all the key-value pairs $\big((i,k),\ (M,j,m_{ij})\big)$ for $k = 1, 2, \ldots$, up to the number of columns of N. Similarly, for each element n_{jk} of N, produce all the key-value pairs $\big((i,k),\ (N,j,n_{jk})\big)$ for $i = 1, 2, \ldots$, up to the number of rows of M. As before, M and N are really bits to tell which of the two matrices a value comes from.

The Reduce Function: Each key (i, k) will have an associated list with all the values (M, j, m_{ij}) and (N, j, n_{jk}), for all possible values of j. The Reduce function needs to connect the two values on the list that have the same value of j, for each j. An easy way to do this step is to sort by j the values that begin with M and sort by j the values that begin with N, in separate lists. The jth values on each list must have their third components, m_{ij} and n_{jk} extracted and multiplied. Then, these products are summed and the result is paired with (i, k) in the output of the Reduce function.

You may notice that if a row of the matrix M or a column of the matrix N is so large that it will not fit in main memory, then the Reduce tasks will be forced to use an external sort to order the values associated with a given key (i, k). However, in that case, the matrices themselves are so large, perhaps 10^{20} elements, that it is unlikely we would attempt this calculation if the matrices were dense. If they are sparse, then we would expect many fewer values to be associated with any one key, and it would be feasible to do the sum of products in main memory.

Q12. What is the role of "combiner" in a Map-Reduce framework? Explain with the help of one example.

3.2.4 Combiners

Q. What are Combiners? When Should one use combiner in mapreduce job? **(May 17,**

Q. What are combiners? Explain the position and significance of combiners.

- A combiner is a type of mediator between the mapper phase and the reducer phase. The use of combiner is optional. As a combiner sits between the mapper and the reducer, it accepts the output of map phase as an input passes the key-value pairs to the reduce operation.
- Combiners are also known as semi-reducers as they reside before the reducer. A combiner is used when the function is commutative and associative. This means that the values can be combined in any order without the final result. For example, addition is an operation which is both commutative as well as associative.
- In case of a huge dataset, when a MapReduce technique is applied, then the mapper phase generates an amount of key-value pairs and these intermediate results need to be handed over to the reducer for the process. But to transfer such a large number of key-value pairs from the mapper to the reducer, sufficient communication network bandwidth is required which generally leads to network congestion.
- So to avoid such congestion we can put some of the work of reducers in to the Map tasks. For example in a single task if c number of (w, 1) key-value pairs appear having the same key, they can be combined into a pair (w, c) w is the key and c is the number of times the word w appears in the set of documents handled by this Map task.
- Fig. 3.2.2 shows position and working mechanism of combiner. Combiner reside between map phase and reduce phase which is responsible for making a group of values of same key which is received from mapper phase.

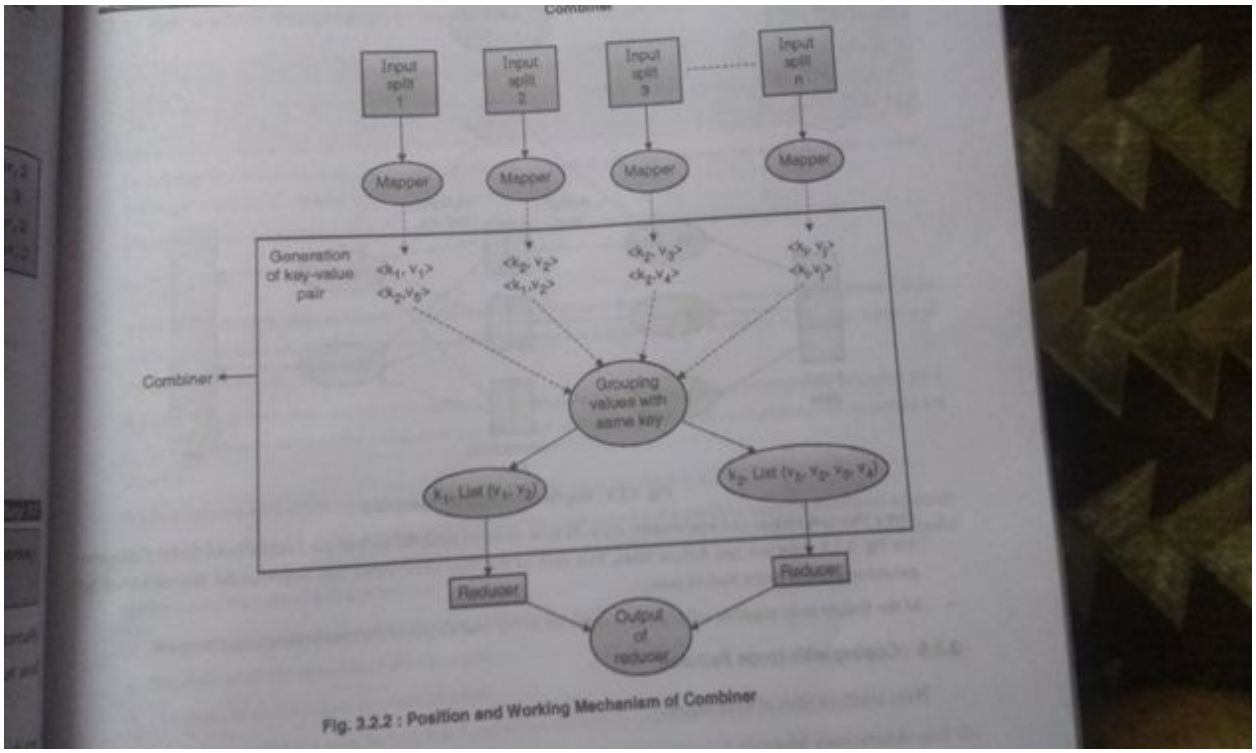


Fig. 3.2.2 : Position and Working Mechanism of Combiner

Chapter 3: NoSQL

1. What are the different architectural patterns in NoSQL? Explain "Graph data store" and Column Family Store patterns with relevant examples.

Ans. Types of NoSQL Data Stores

The following types of NoSQL architectural patterns:

1. Key−value store.

2. Column store.

3. Document store.

4. Graph store.

Graph Data Store

- Graph store is based on graph theory. These databases are designed for data whose relations are represented as a graph and have elements which are interconnected, with

an undetermined number of relations between them. Examples include: Neo4j, AllegroGraph, TeradataAster.

- Graph databases are used when a business problem has complex relationship among their objects, especially in social networks and rule-based engines. Social networking apps like Facebook, Google+, LinkedIn, Twitter, etc. and video hosting applications like YouTube, Flickr, etc. use graph database to store their data. In Google+, profile data are stored internally as nodes and nodes are connected with each other using relationships.
- Graph databases store nodes and links in an optimized way so that querying these graph stores (graph traversal) is simpler. It does not require complex JOINS or indexes to retrieve connected data from its adjacent node. Graph stores contain sequence of nodes and relations that form the graph. Both nodes and relationships contain properties like follows, friend, family, etc. So a graph store has three fields: nodes, relationships and properties. Properties are key−value pairs that represent data.
- For example, Node Name = “Employee” and (empno:2300, empname: “Tom”, deptno: 20) are a set of properties as key−value pairs. Dept can be another node. The relationship that connects them could be “works_for”. “works_for” relationship has empno: 2300 property as key−value pair. Each relationship has start/from (here “Employee”) node and end/to (“Dept”) node.
- In Neo4j relationships should be directional (unidirectional or bidirectional), else it will throw error. Neo4j does not support Sharding.

Column Family Store

Column family is different from column store. Instead of storing data in rows, these column store databases are designed to store data tables as section of column of data, rather than as rows of data. They are generally used in sparse matrix system. Wide-column store offers very high performance and a highly scalable architecture. Column stores are used in OLAP systems for their ability to rapidly aggregate column data. Column family store uses the same concept of spreadsheet where a cell is identified by row (number) and column (name) identifier. Similarly, a column family store uses a row and a column as keys. In addition to this, a column family to group similar column names together. A timestamp is also included to store multiple versions of a value over time. Therefore is < (rowed, column family, column name, timestamp), value>. Like traditional databases all the column values for each row need not be stored. Examples include: BigTable from Google, HBase, Accumulo, HyperTable, Cassandra. Figure shows the sample of tweets stored as column family; key and value are marked. Cassandra’s data model is a column-family-based one. Cassandra has a multi-dimensional data model. Table defines a high-level grouping of the data and in Cassandra it is referred to as a keyspace. There’s one keyspace for each application. The column family is the basic unit of data organization and it lies within a keyspace. A column family stores together a set of columns for a given row. In other words, in a column-oriented database, data is stored by column, with all rows grouped together. This optimizes the column design by grouping commonly queried columns together.

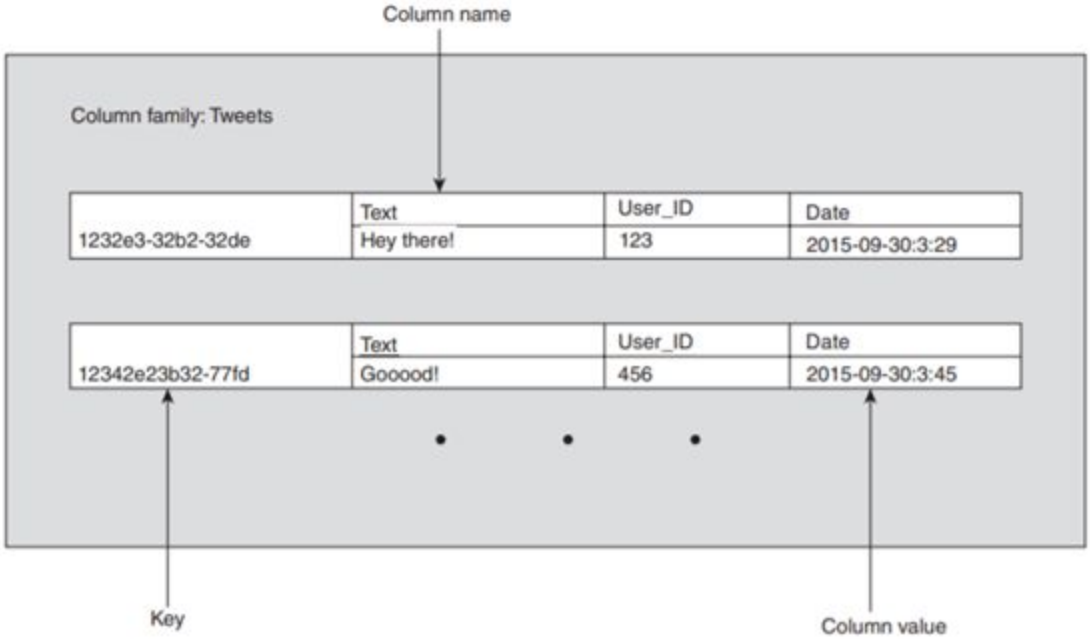


Figure 3.4 Sample column family store.

EmployeeIndia : {

address: {

city: Mumbai

pincode: 400058

},

projectDetails: {

durationDays: 100

cost: 50000

}

Here

1. The outermost key EmployeeIndia is analogues to row.

2. “address” and “projectDetails” are called column families.

3. The column-family “address” has columns “city” and “pincode”.

4. The column-family “projectDetails” has columns “durationDays” and “cost”.

5. Columns can be referenced using ColumnFamily

2. What are the different architectural patterns in NoSQL? Explain “key store” and “Document Store” patterns with relevant examples.

Ans. Types of NoSQL Data Stores

The following types of NoSQL architectural patterns:

1. Key−value store.

2. Column store.

3. Document store.

4. Graph store.

Key Store

The key−value store uses a key to access a value. The key−value store has a schema-less format. The key can be artificially generated or auto-generated while the value can be String, JSON, BLOB, etc. For example, key could be a web page, file path, REST call, image name, SQL query, etc. The key−value type uses a hash table with a unique key and a pointer to a particular item of data. A bucket is a logical group (not physical) of keys and so different buckets can have identical keys. The real key is a hash (Bucket + Key). Use of the cache mechanisms for the mappings improves the performance. The clients can read and write values using a key as shown below: 1. To fetch the value associated with the key use − Get(key). 2. To associate the value with the key use − Put(key, value). 3. To fetch the list of values associated with the list of keys use − Multi-get (key1, key2, …, keyN). 4. To remove the entry for the key from the data store use − Delete(key). In addition to Get, Put, and Delete API, key−value store has two rules: 1. Distinct keys: All keys in key−value store are unique. 2. No queries on values: No queries can be performed on the values of the table.

Document Store

Document store basically expands on the basic idea of key−value stores where “documents” are more complex, in that they contain data and each document is assigned a unique key, which is used to retrieve the document. These are designed for storing, retrieving, and managing document-oriented information, also known as semi-structured data. Hierarchical (tree) data structures can be directly stored in document database. The column family and key−value store

lack a formal structure and so cannot be indexed. Hence searching is not possible. This problem is resolved in document store. Key here is a simple id. Irrespective of the id, a query can result in getting any item (value or content) out of a document store. This is made possible in a document store because everything inside a document is automatically indexed. So if you want to search for a particular property, all the documents with the same property can be quickly found. The difference in key−value store and document store is that the key−value store stores into memory the entire document in the value portion whereas document store extracts subsections of all documents. Document store uses a tree structure. “Document path” is used like a key to access the leaf values of a document tree structure. So the search item’s exact location can also be found. For example, if the root is Employee, the path can be Employee[id=‘2300’]/Address/street/BuildingName/text() Though the document store tree structure is complex the search API is simple. Document structure uses JSON (JavaScript Object Notation) format for deep nesting of tree structures associated with serialized objects. But JSON does not support document attributes such as bold, hyperlinks, etc. Examples include: MongoDB, CouchBase, and CouchDB. MongoDB uses a document model. Document can be thought of as a row or tuple in RDBMS. Document (object) map to data type in programming language. A MongoDB database holds a collection of documents. Embedded documents and arrays reduce need for JOINs, which is a key factor for improving performance and speed. MongoDB focuses on speed, power, ease of use and flexibility. MongoDB became a popular NoSQL product because of its Ad serving. This service of MongoDB will quickly send at the same time a banner Ad to millions of users who are interested in the Ad product or service, when they browse a web page. The Ad servers have to be up and available 24 × 7. They should apply complex business rules to select the most appropriate Ad and place it on the user’s webpage when it loads. MongoDB also services content management, real-time sentiment analysis, product management, to store user data in video games and similar domains. MongoDB stores document in Binary JSON (BSON) format. Documents with similar structure are organized as collections, similar to table in RDBMS; documents are similar to rows and fields are similar to columns. All the data is spread across many tables after normalization in RDBMS, whereas in MongoDB all data related to a single record reside in a single document. In a document, multiple arrays of sub-documents can be embedded. Single read statement can get the entire document. Data in MongoDB is localized; hence there is no need for any JOIN operations. Documents are self-describing, so schema can vary from document to document.

3. Explain Key-value store architectural pattern with example. (Amazon DynamoDB)

Ans. The key−value store uses a key to access a value. The key−value store has a schema-less format. The key can be artificially generated or auto-generated while the value can be String, JSON, BLOB, etc. For example, key could be a web page, file path, REST call, image name, SQL query, etc. The key−value type uses a hash table with a unique key and a pointer to a particular item of data. A bucket is a logical group (not physical) of keys and so different buckets can have identical keys. The real key is a hash (Bucket + Key). Use of the cache

mechanisms for the mappings improves the performance. The clients can read and write values using a key as shown below:

1. To fetch the value associated with the key use − Get(key).

2. To associate the value with the key use − Put(key, value).

3. To fetch the list of values associated with the list of keys use − Multi-get (key1, key2, …, keyN).

4. To remove the entry for the key from the data store use − Delete(key). In addition to Get, Put, and Delete API, key−value store has two rules:

1. Distinct keys: All keys in key−value store are unique.

2. No queries on values: No queries can be performed on the values of the table.

Amazon.com has one of the largest e-commerce operations in the world. Customers from all around the world shop all hours of the day. So the site has to be up 24 × 7. Initially Amazon used RDBMS system for shopping cart and checkout system. Amazon DynamoDB, a NoSQL store brought a turning point. DynamoDB addresses performance, scalability and reliability, the core problems of RDBMS when it comes to growing data. Developers can store unlimited amount of data by creating a database table and DynamoDB automatically saves it at multiple servers specified by the customer and also replicates them across multiple “Available” Zones. It can handle the data and traffic while maintaining consistent, fast performance. The cart data and session data are stored in the key−value store and the final (completed) order is saved in the RDBMS.

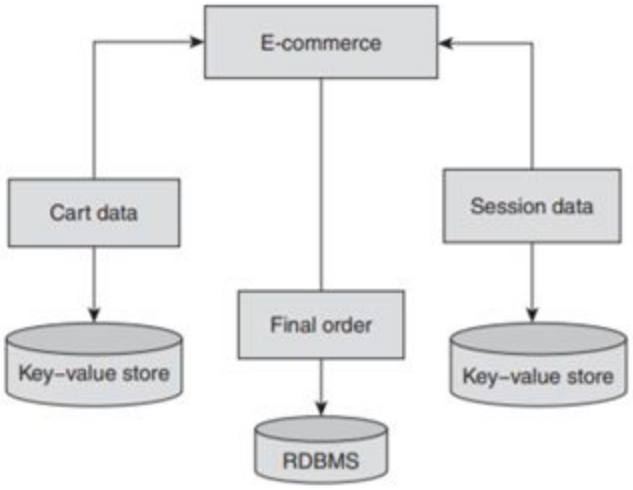


Figure 3.3 E-commerce shopping cart uses key–value store.

The salient features of key−value store are as follows: 1. Scalable: If application’s requirements change, using the AWS Management Console or the DynamoDB APIs table throughput

capacity can be updated. 2. Automated storage scaling: Using the DynamoDB write APIs, more storage can be obtained, whenever additional storage is required. 3. Distributed horizontally shared nothing: Seamlessly scales a single table over hundreds of commodity servers. 4. Built-in fault tolerance: DynamoDB automatically replicates data across multiple available zones in synchronous manner. 5. Flexible: DynamoDB has schema-free format. Multiple data types (strings, numbers, binary and sets) can be used and each data item can have different number of attributes. 6. Efficient indexing: Every item is identified by a primary key. It allows secondary indexes on non-key attributes, and query data using an alternate key. 7. Strong consistency, Atomic counters: DynamoDB ensures strong consistency on reads (only the latest values are read). DynamoDB service also supports atomic counters to atomically increment or decrement numerical attributes with a single API call. 8. Secure: DynamoDB uses cryptography to authenticate users. Access control for users within organization can be secured by integrating with AWS Identity and Access Management. 9. Resource consumption monitoring: AWS Management Console displays request throughput and latency for each DynamoDB table, since it is integrated with CloudWatch. 10. Data warehouse − Business intelligence facility: Amazon Redshift Integration − data from DynamoDB tables can be loaded into Amazon Redshift (data warehouse service). 11. MapReduce integration: DynamoDB is also integrated with Amazon Elastic MapReduce to perform complex analytics (hosted pay-as-you-go Hadoop framework on AWS).

4. Explain Column family store architectural pattern with example. (Big Table/Hbase)

Ans.

On the contrary to row-oriented database, column-oriented database divides the tables by columns. Figure 2.2 shows the physical layout of the row-oriented database. Usually, in the database, many of the data are irrelevant to the query. Column store database can solve this problem. Each time when we need to find out a certain value, we just need to go through a certain column, it could avoid the waste of searching for useless columns. Column store database has four basic elements: Column Family, Column, Row Key, Timestamp. Column family is a collection of columns. Physically, columns which belong to one family will store together on the file system. So if the columns have the similar access pattern, it’s advised to have these columns to be in the same column family. HBase suggests to have no more than three column families in the database sever. What’ more, HBase suggests to have same cardinality of the rows in different column families. HBase will split the data rows and store them into different region servers. For instance column family A only has a few rows, but the second column family B has huge amount of the rows. The data will still be split based on number of rows of column family B. It will lead to inefficiency of scanning the the column family A.

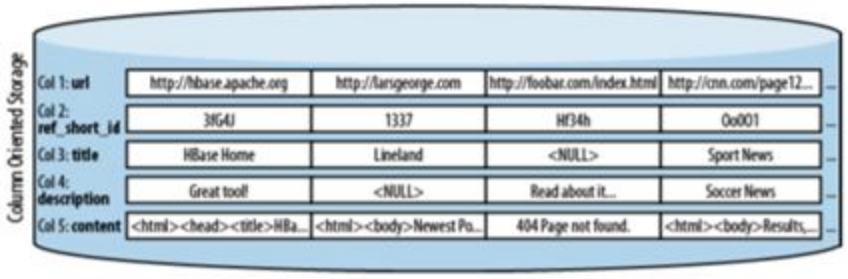


Figure 2.2: Column-oriented database[5]

Google's motivation for developing BigTable is driven by its need for massive scalability, better performance characteristics, and ability run on commodity hardware. Each time when a new service or increase in load happens, its solution BigTable would result in only a small incremental cost. Volume of Google's data generally is in petabytes and is distributed over 100,000 nodes. BigTable is built on top of Google's other services that have been in active use since 2005, namely, Google File System, Scheduler, MapReduce and Chubby Lock Service. BigTable is a column-family database which is designed to store data tables (sparse matrix of row, column values) as section of column of data. It is distributed (high volume of data), persistent, multi-dimensional sorted map. The map is indexed by a row key, column key and a timestamp (int64).

In the HBase data model columns are grouped into *column families*, which must be defined up front during table creation. Column families are stored together on disk, which is why HBase is referred to as a column-oriented data store.

Logical View of Customer Contact Information in HBase

Row Key	Column Family: {Column Qualifier:Version:Value}
00001	CustomerName: {'FN': 1383859182496:'John', 'LN': 1383859182858:'Smith', 'MN': 1383859183001:'Timothy', 'MN': 1383859182915:'T'} ContactInfo: {'EA': 1383859183030:'John.Smith@xyz.com', 'SA': 1383859183073:'1 Hadoop Lane, NY

	11111’}
00002	CustomerName: {‘FN’: 1383859183103:‘Jane’, ‘LN’: 1383859183163:‘Doe’, ContactInfo: { ’SA’: 1383859185577:’7 HBase Ave, CA 22222’}

The table shows two column families: CustomerName and ContactInfo. When creating a table in HBase, the developer or administrator is required to define one or more column families using printable characters.

Generally, column families remain fixed throughout the lifetime of an HBase table but new column families can be added by using administrative commands. The official recommendation for the number of column families per table is three or less.

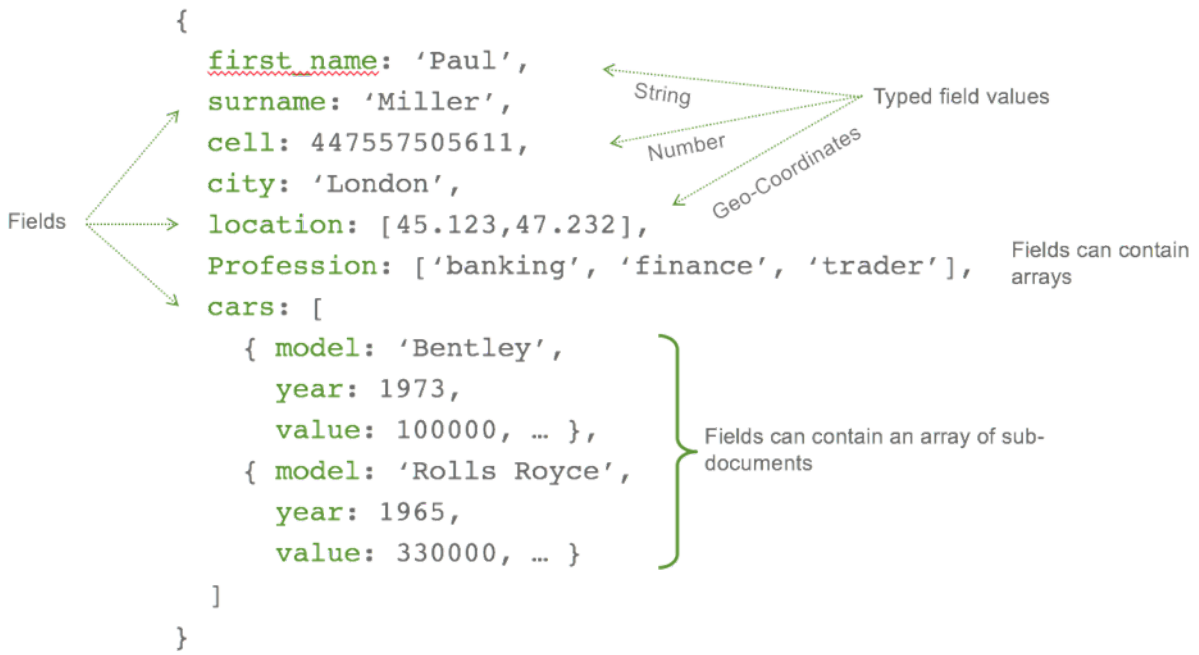
In addition, you should store data with similar access patterns in the same column family — you wouldn’t want a customer’s middle name stored in a separate column family from the first or last name because you generally access all name data at the same time.

5. Explain Document store architectural pattern with example. (MongoDB)

Ans. Document store basically expands on the basic idea of key−value stores where “documents” are more complex, in that they contain data and each document is assigned a unique key, which is used to retrieve the document. These are designed for storing, retrieving, and managing document-oriented information, also known as semi-structured data. Hierarchical (tree) data structures can be directly stored in document database. The column family and key−value store lack a formal structure and so cannot be indexed. Hence searching is not possible. This problem is resolved in document store. Key here is a simple id. Irrespective of the id, a query can result in getting any item (value or content) out of a document store. This is made possible in a document store because everything inside a document is automatically indexed. So if you want to search for a particular property, all the documents with the same property can be quickly found. The difference in key−value store and document store is that the key−value store stores into memory the entire document in the value portion whereas document store extracts subsections of all documents. Document store uses a tree structure. “Document path” is used like a key to access the leaf values of a document tree structure. So the search item’s exact location can also be found. For example, if the root is Employee, the path can be Employee[id=‘2300’]/Address/street/BuildingName/text() Though the document store tree

structure is complex the search API is simple. Document structure uses JSON (JavaScript Object Notation) format for deep nesting of tree structures associated with serialized objects. But JSON does not support document attributes such as bold, hyperlinks, etc. Examples include: MongoDB, CouchBase, and CouchDB. MongoDB uses a document model. Document can be thought of as a row or tuple in RDBMS. Document (object) map to data type in programming language. A MongoDB database holds a collection of documents. Embedded documents and arrays reduce need for JOINs, which is a key factor for improving performance and speed. MongoDB focuses on speed, power, ease of use and flexibility. MongoDB became a popular NoSQL product because of its Ad serving. This service of MongoDB will quickly send at the same time a banner Ad to millions of users who are interested in the Ad product or service, when they browse a web page. The Ad servers have to be up and available 24 × 7. They should apply complex business rules to select the most appropriate Ad and place it on the user's webpage when it loads. MongoDB also services content management, real-time sentiment analysis, product management, to store user data in video games and similar domains. MongoDB stores document in Binary JSON (BSON) format. Documents with similar structure are organized as collections, similar to table in RDBMS; documents are similar to rows and fields are similar to columns. All the data is spread across many tables after normalization in RDBMS, whereas in MongoDB all data related to a single record reside in a single document. In a document, multiple arrays of sub-documents can be embedded. Single read statement can get the entire document. Data in MongoDB is localized; hence there is no need for any JOIN operations. Documents are self-describing, so schema can vary from document to document.

Example of MongoDB document:-



6. Explain Graph store architectural pattern with example. (Neo4j)

Ans. Graph store is based on graph theory. These databases are designed for data whose relations are represented as a graph and have elements which are interconnected, with an

undetermined number of relations between them. Examples include: Neo4j, AllegroGraph, TeradataAster. Graph databases are used when a business problem has complex relationship among their objects, especially in social networks and rule-based engines. Social networking apps like Facebook, Google+, LinkedIn, Twitter, etc. and video hosting applications like YouTube, Flickr, etc. use graph database to store their data. In Google+, profile data are stored internally as nodes and nodes are connected with each other using relationships. Graph databases store nodes and links in an optimized way so that querying these graph stores (graph traversal) is simpler. It does not require complex JOINS or indexes to retrieve connected data from its adjacent node. Graph stores contain sequence of nodes and relations that form the graph. Both nodes and relationships contain properties like follows, friend, family, etc. So a graph store has three fields: nodes, relationships and properties. Properties are key−value pairs that represent data. For example, Node Name = “Employee” and (empno:2300, empname: “Tom”, deptno: 20) are a set of properties as key−value pairs. Dept can be another node. The relationship that connects them could be “works_for”. “works_for” relationship has empno: 2300 property as key−value pair. Each relationship has start/from (here “Employee”) node and end/to (“Dept”) node. In Neo4j relationships should be directional (unidirectional or bidirectional), else it will throw error. Neo4j does not support Sharding.

Neo4j is an open-source (source code is available in github) sponsored by Neo Technology. Its NoSQL graph database is implemented in Java and Scala. Its development started in 2003; it was made publicly available since 2007. Neo4j is used today by hundreds to thousands of enterprises. To name a few: scientific research, routing, matchmaking, network management, recommendations, social networks, software analytics, organizational and project management.

Neo4j implements the Property Graph Model efficiently down to the storage level. In contrast to graph processing or in-memory libraries, Neo4j supports ACID transaction compliance and runtime failover, making it suitable for production scenarios.

Some of Neo4j features are listed below:

1. Neo4j has CQL, Cypher query language much like SQL.

2. Neo4j supports Indexes by using Apache Lucence.

3. It supports UNIQUE constraints.

4. Neo4j Data Browser is the UI to execute CQL Commands.

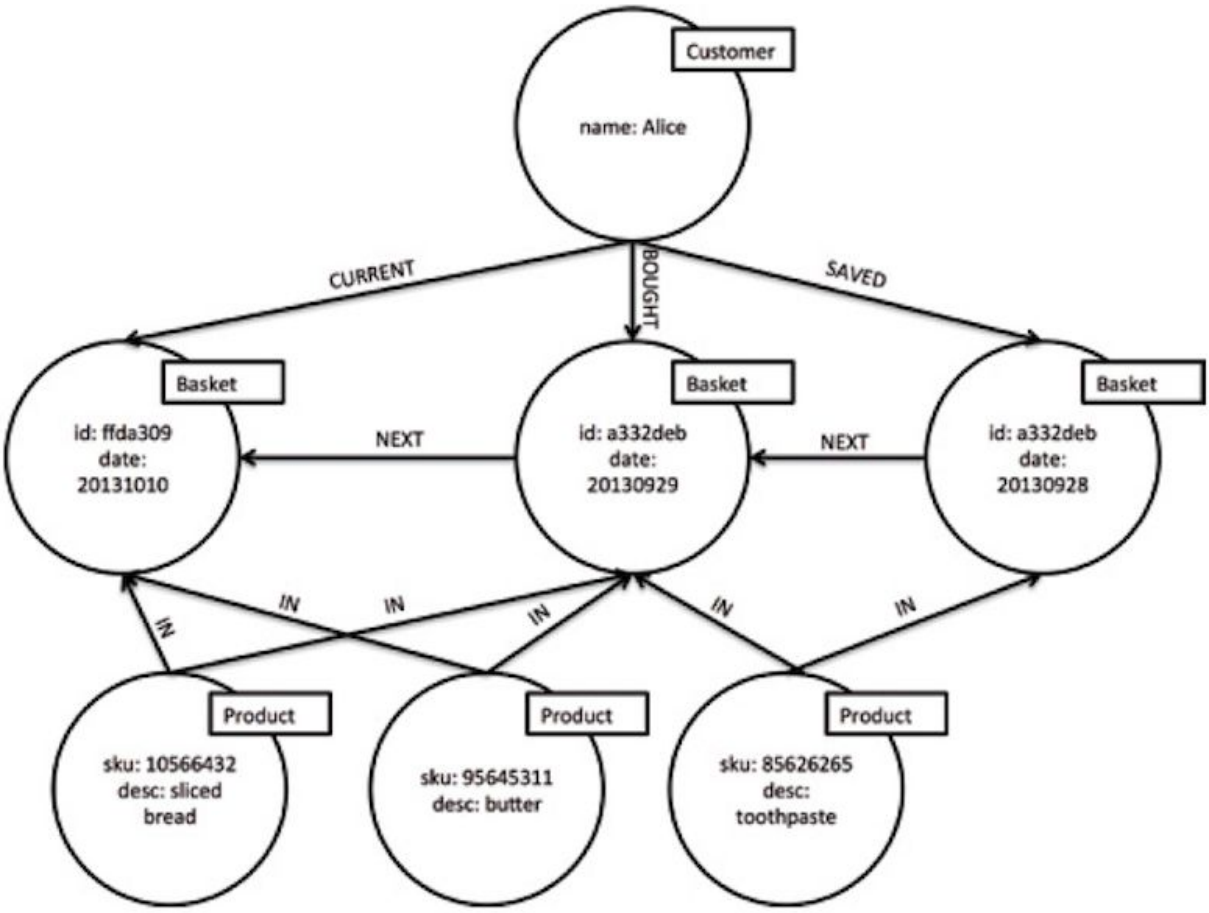
5. It supports ACID properties of RDBMS.

6. It uses Native Graph Processing Engine to store graphs.

7. It can export query data to JSON and XLS format.

8. It provides REST API to Java, Scala, etc.

Example of Graph database:-



7. What are NoSQL Business Drivers? What features of NoSQL drive business?

Ans. Enterprises today need highly reliable, scalable and available data storage across a configurable set of systems that act as storage nodes. The needs of organizations are changing rapidly. Many organizations operating with single CPU and Relational database management systems (RDBMS) were not able to cope up with the speed in which information needs to be extracted. Businesses have to capture and analyze a large amount of variable data, and make immediate changes in their business based on their findings.

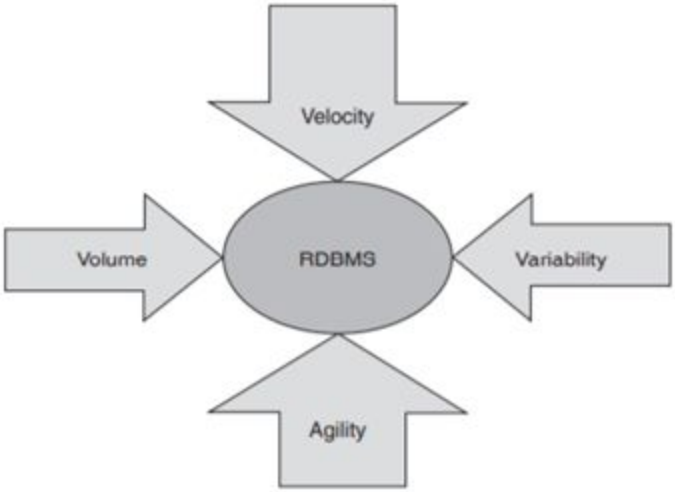


Figure 3.1 NoSQL business drivers.

Volume

There are two ways to look into data processing to improve performance. If the key factor is only speed, a faster processor could be used. If the processing involves complex (heavy) computation, Graphic Processing Unit (GPU) could be used along with the CPU. But the volume of data is limited to on-board GPU memory. The main reason for organizations to look at an alternative to their current RDBMSs is the need to query big data. The need to horizontal scaling made organizations to move from serial to distributed parallel processing where big data is fragmented and processed using clusters of commodity machines. This is made possible by the development of technologies like Apache Hadoop, HDFS, MapR, HBase, etc.

Velocity

Velocity becomes the key factor when frequency in which online queries to the database made by social networking and e-commerce web sites have to be read and written in real time. Many single CPU, RDBMS systems are unable to cope up with the demands of real-time inserts. RDBMS systems frequently index on many columns that decrease the system performance. For example, when online shopping sites introduce great discount schemes, the random bursts in web traffic will slow down the response for every user and tuning these systems as demand increases can be costly when both high read and write is required.

Variability

Organizations that need to capture and report on certain uncommon data, struggle when attempting to use RDBMS fixed schema. For example, if a business process wants to store a few special attributes for a few set of customers, then it needs to alter its schema definition. If a change is made to the schema, all customer rows within the database will also have this column. If there is no value related to this for most of the customers, then the row column representation will have sparse matrix. In addition to this, new columns to an RDBMS require to

execute ALTER TABLE command. This cannot be done on the fly since the present executing transaction has to complete and database has to be closed, and then schema can be altered. This process affects system availability, which means losing business.

Agility

The process of storing and retrieving data for complex queries in RDBMS is quite cumbersome. If it is a nested query, data will have nested and repeated subgroups of data structures that are included in an object-relational mapping layer. This layer is responsible to generate the exact combination of SELECT, INSERT, DELETE and UPDATE SQL statements to move the object data from and to the backend RDBMS layer. This process is not simple and requires experienced developers with the knowledge of object-relational frameworks such as Java Hibernate. Even then, these change requests can cause slowdowns in implementation and testing.

Analytics and business intelligence

A key business strategic driver that suggests the implementation of a NoSQL database environment is the need to mine the data that is being collected in order to derive insights to gain competitive advantage. Traditional relational database system poses great difficulty in extracting meaningful business intelligence from very high volumes of data. NoSQL database systems not only provide storage and management of big data but also deliver integrated data analytics that provides instant understanding of complex datasets and facilitate various options for easy decision-making.

Q8. What are the five ways that NoSQL system handles big data problems?

1. Moving queries to the data, not data to the queries

With the exception of large graph databases, most NoSQL systems use commodity processors that each hold a subset of the data on their local shared-nothing drives. When a client wants to send a general query to all nodes that hold data, it’s more efficient to send the query to each node than it is to transfer large datasets to a central processor. This simple rule helps you understand how NoSQL databases can have dramatic performance advantages over systems that weren’t designed to distribute queries to the data nodes. Keeping all the data within each data node in the form of logical documents means that only the query itself and the final result need to be moved over a network. This keeps your big data queries fast.

2. Using hash rings to evenly distribute data on a cluster:

One of the most challenging problems with distributed databases is figuring out a consistent way of assigning a document to a processing node. Using a hash ring technique to evenly distribute big data loads over many servers with a randomly generated 40-character key is a good way to evenly distribute a network load.

Hash rings are common in big data solutions because they consistently determine how to assign a piece of data to a specific processor. Hash rings take the leading bits of a document’s hash value and use this to determine which node the document should be assigned. This allows

any node in a cluster to know what node the data lives on and how to adapt to new assignment methods as your data grows. Partitioning keys into ranges and assigning different key ranges to specific nodes is known as keyspace management.

3. Using replication to scale reads:

Databases use replication to make backup copies of data in real time. Using replication allows you to horizontally scale read requests. Figure shows this structure.

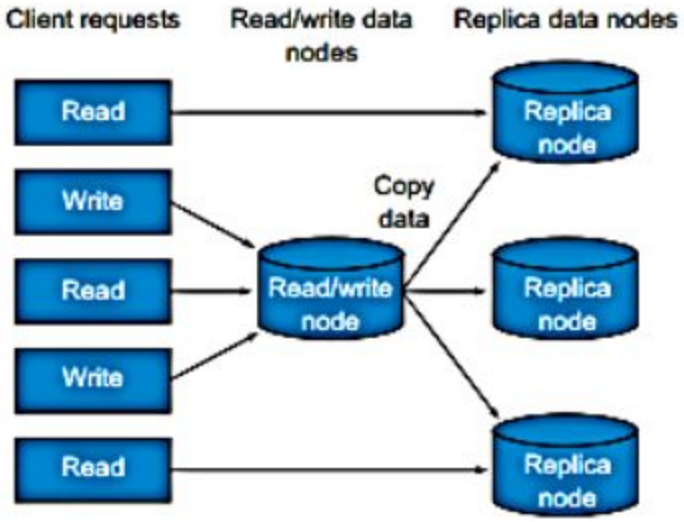


Figure: How you can replicate data to speed read performance in NoSQL systems.

This replication strategy works well in most cases. There are only a few times when you must be concerned about the lag time between a write to the read/write node and a client reading that same record from a replica. One of the most common operations after a write is a read of that same record. If a client does a write and then an immediate read from that same node, there's no problem. The problem occurs if a read occurs from a replica node before the update happens. This is an example of an inconsistent read. The best way to avoid this type of problem is to only allow reads to the same write node after a write has been done. This logic can be added to a session or state management system at the application layer. Almost all distributed databases relax data eeeconsistency rules when a large number of nodes permit writes. If your application needs fast read/write consistency, you must deal with it at the application layer.

4. Letting the database distribute queries evenly to data nodes:

In order to get high performance from queries that span multiple nodes, it's important to separate the concerns of query evaluation from query execution. Figure shows this structure:

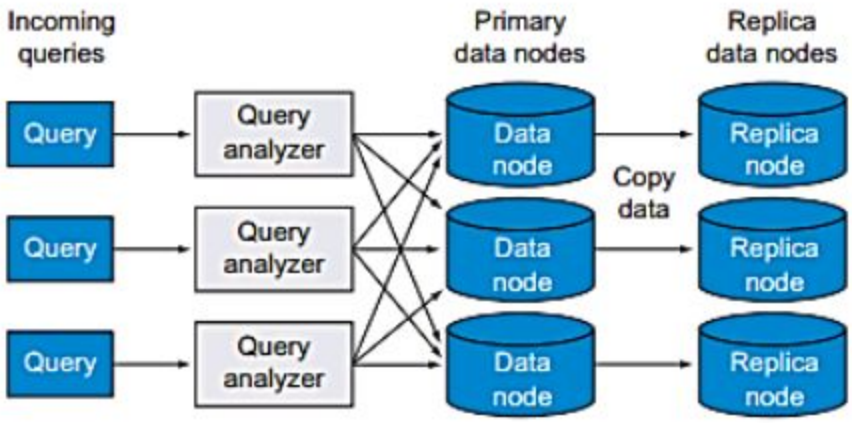


Figure NoSQL systems move the query to a data node, but don't move data to a query node. In this example, all incoming queries arrive at query analyzer nodes. These nodes then forward the queries to each data node. If they have matches, the documents are returned to the query node. The query won't return until all data nodes (or a response from a replica) have responded to the original query request. If the data node is down, a query can be redirected to a replica of the data node.

This approach is somewhat similar to the concept of federated search. Federated search takes a single query and distributes it to distinct servers and then combines the results together to give the user the impression they're searching a single system.