Big data is the term for a collection of data sets so large and complex that it become difficult to process using on-hand data base management tools or traditional data processing applications.

Big data is a collection of data from many different sources and is often describe by five characteristics

Volume, Variety, Velocity, Veracity, Value

1. **Volume**: the size and amounts of big data that companies manage and analyze

Sources: People, machine, organization, Ubiquitous computing.

* 1. Typical internal data sources Data reside inside organization firewall.

File system, SQL(RDBMS- Oracle, MS SQL server, My SQL, No SQL, Cassandra etc. and so on.

Archives Archives of scanned documents, paper archives, customer correspondence record, student admission record, student assessment records and so on.

2. External data sources: Data reside outside an organization's firewall.

public web: Wikipedia, weather, regulatory, census etc.

3. Both(internal external sources)

Sensor data, machine log data, social media, Business apps, Media, DOCS.

1. **Variety**: It deals with the wide range of data types and sources of data.

1. Structured schema/structure)-data from (predefined traditional transaction processing system and RDBMS etc.

2. Semi Structured data(uses tags to segregate elements) for ex. HTML, XML, JSON etc.

3. Unstructured data(does not confirm to any predefined data model) for ex. Unstructured text document, audios, videos, email, photos, social media etc.

1. **Value**: the most important "V" from the perspective of the business, the value of big data usually comes from insight discovery and pattern recognition that lead to more effective operations, stronger customer relationships and other clear and quantifiable business benefits.
2. **Velocity**: the speed at which companies receive, store and manage data e.g., the specific number of social media posts or search queries received within a day, hour or other unit of time.

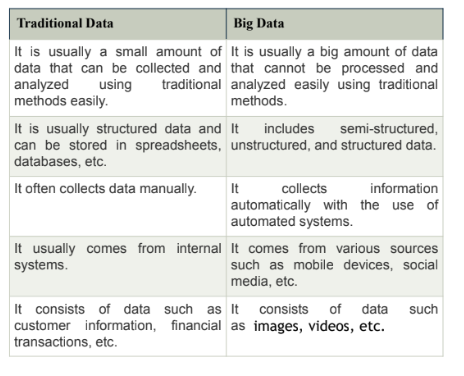
We moved from the days of batch processing to real time processing.

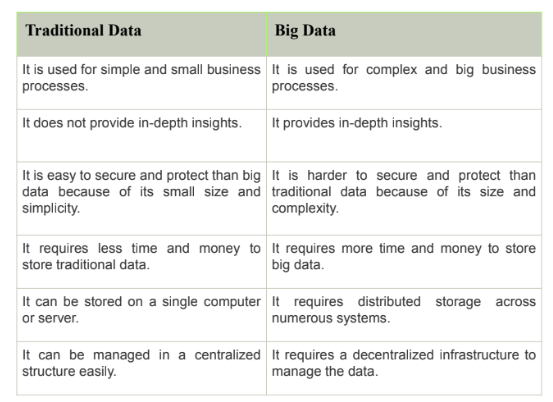
1. **Veracity**: the "truth" or accuracy of data and information assets, which often determines executive-level confidence.

Hadoop is an open-source software framework for storing and processing big data in a distributed way on large clusters of commodity hardware. Basically, it accomplishes the following two tasks:

* 1. Massive data storage.
  2. Faster processing

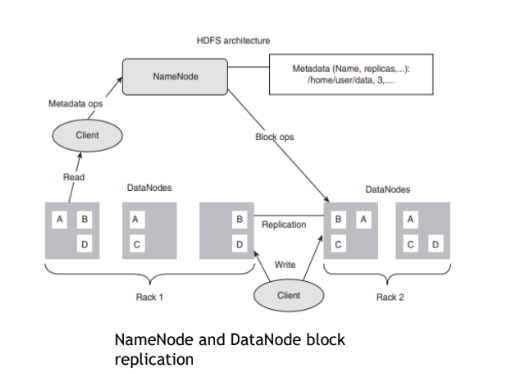
**Traditional data vs big data diff**

****

****

**Core components of hadoop**

**Hdfs Arch diagram**

****

**Hadoop Common:** This package provides file system and OS level abstractions. It contains libraries and utilities required by other Hadoop modules.

This consists of necessary Java archive (JAR) files and scripts needed to start Hadoop. Hadoop requires Java Runtime Environment (JRE) 1.6 or higher version. The standard start-up and shut-down scripts need Secure Shell (Ssh) to be setup between the nodes in the cluster.

**Hadoop Distributed File System (HDFS):** HDFS is a distributed file system that provides a limited interface for managing the file system.

HDFS creates multiple replicas of each data block and distributes them on computers throughout a cluster to enable reliable and rapid access.

When a file is loaded into HDFS, it is replicated and fragmented into “blocks” of data, which are stored across the cluster nodes; the cluster nodes are also called the DataNodes.

The NameNode is responsible for storage and management of metadata, so that when MapReduce or another execution framework calls for the data, the NameNode informs it where the data that is needed resides.

**NameNode** is the master that contains the metadata. In general, it maintains the directories and files and manages the blocks which are present on the DataNode

**DataNodes** are the slaves which provide the actual storage and are deployed on each machine. They are responsible for processing read and write requests for the clients

**Hadoop Map Reduce:** Map Reduce is the key algorithm that the Hadoop Map Reduce engine uses to distribute work around a cluster.

The MapReduce algorithm aids in parallel processing and basically comprises sequential phases: map and reduce. Two

1. In the **map phase,** a set of key-value pairs forms the input and over each key-value pair, the desired function is executed so as to generate a set of intermediate key-value pairs.
2. In the **reduce phase**, the intermediate key-value pairs are grouped by key and the values are combined together according to the reduce algorithm provided by the user. Sometimes no reduce phase is required, given the type of operation coded by the user

**Main Components of MapReduce** The main components of MapReduce are listed below:

1. JobTrackers: JobTracker is the master which manages the jobs and resources in the cluster. The Job Tracker tries to schedule each map on the Task Tracker which is running on the same DataNode as the underlying block.

2. Task Trackers: TaskTrackers are slaves which are deployed on each machine in the cluster. They are responsible for running the map and reduce tasks as instructed by the JobTracker.

3. JobHistoryServer: JobHistoryServer is a daemon that saves historical information about completed tasks/applications

**Hadoop Yet Another Resource Negotiator (YARN) (Map Reduce 2.0)**: It is a resource management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications.

It splits up the two major functionalities "resource management" and "job scheduling and monitoring" of the JobTracker into two separate daemons. One acts as a "global Resource Manager (RM)" and the other as a "ApplicationMaster (AM)" per application. Thus, instead of having a single node to handle both scheduling and resource management for the entire cluster, YARN distributes this responsibility across the cluster.

**HDFS ECOSYSTEM**

1. Apache H Base: Columnar (Non-relational) database

2. Apache Hive: Data access and query

3. Apache H Catalog: Metadata services

4. Apache Pig: Scripting platform.

5. Apache Mahout: Machine learning libraries for Data Mining

6. Apache Oozie: Workflow and scheduling services.

7. Apache Zookeeper: Cluster coordination.

8. Apache Sqoop: Data integration services.

**Word count using MapReduce algorithm**

map (String key, String value)

// key: document name

// value: document contents

for each word w in value

EmitIntermediate(w, "1")

reduce(String key, Iterator values):

// key: word

// values: a list of counts

for each v in values:

result += ParseInt(v);

Emit(AsString(result));

**NoSQL** is database management system that provides mechanism for storage and retrieval of massive amount of unstructured data in a distributed environment on virtual servers with the focus to provide high scalability, performance, availability and agility

NoSQL database is also referred as Not only SQL

1 NoSQL is next generation database which is completely different from the traditional database.

2 NoSQL stands for Not only SQL. SQL as well as other query languages can be used with NoSQL dataAPI.11l

3NoSQL is non-relational database, and it is schema-free.

4 NoSQL is free of JOINS.

5 NoSQL uses distributed architecture and works on multiple processors to give high performance.

6 NoSQL databases are horizontally scalable.

7 Many open-source NoSQL databases are available.

8 Data file can be easily replicated.

9 NoSQL uses simple API.

10. NoSQL can manage huge amount of data.

11 NoSQL can be implemented on commodity Hardware which has separate RAM and disk (shared nothing concept).

**WHY NOSQL?**

NoSQL is known for its high performance with high availability, rich query language, and easy scalability which fits the need.

**CAP THEOREM**

Consistency:

All nodes should see the same data at the same time

Availability:

System is always on, no downtime.

Partition-tolerance:

The system continues to operate despite network partitions

A distributed system can satisfy any two of these guarantees at the same time but not all three

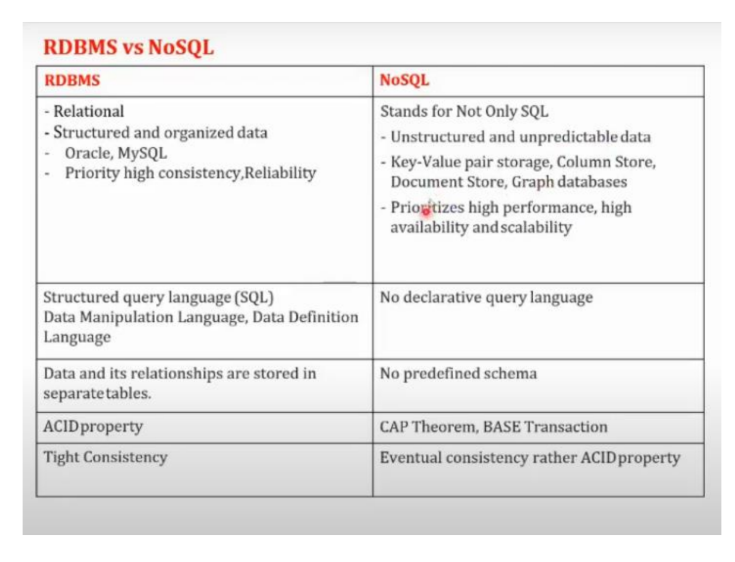
**BASE**: BASICALLY AVAILABLE, SOFT STATE, EVENTUAL CONSISTENCY

Basically, available means DB is available all the time as per CAP theorem

Soft state means even without an input; the system state may change

Eventual consistency means that the system will become consistent over time

**Sql no sql diff**

****

There are two ways of scaling horizontal and vertical scaling:

**Vertical scaling**

To scale vertically (or scale up) means to add resources within the same logical unit to increase capacity

For example to add CPUs to an existing server, increase memory in the system or expanding storage by adding hard drive.

**Horizontal scaling**

To scale horizontally (or scale out) means to add more nodes to a system, such as adding a new computer to a distributed software application.

In NoSQL system, data store can be much faster as it takes advantage of “scaling out which means to add more nodes to a system and distribute the load over those nodes

How the **business drivers** volume, velocity, variability, and agility apply pressure to the single CPU system, resulting in the cracks. **Volume and velocity** refer to the ability to handle large datasets that arrive quickly. **Variability** refers to how diverse data types don’t fit into structured tables, and **agility** refers to how quickly an organization responds to business change.

Read from the ppt👆

**Types of NoSql**

**Key-Value**

Simplest

Everything is Key = Attribute name + Value

Amazon DynamoDB

**Eg**. Oracle NoSQL, Redis Server, Scalaris

**Document Based**

Key with complex Data structure = doc

Doc have key-value pairs Nested documents

MongoDB

**Eg**. MongoDB, CouchDB, OrientDB, RavenDB

**Column based**

Optimized for queries.

Store columns of data together.

Hbase

**Eg**.BigTable, Cassandra, HBase, Hypertable

**Graph Store**

Node-social connections

Optimized for graph based algorithms.

Neo4j

**Eg**. Neo4j, InfoGrid, Infinite Graph, FlockDB

A **data stream** is a real-time, continuous and ordered sequence of items.

We do not know entire data set in advance.

**Ex**.. How many people are involved in searching in Google.

Google queries

Facebook or twitter status update

A **Data Stream Management System (DSMS)** is a software system designed to manage and process continuous, real-time streams of data, unlike traditional DBMSs that work with static, stored datasets.

It is widely used in applications such as IoT monitoring, stock trading, network traffic analysis, and social media analytics. Components:

**Stream Source**: Entry point of real-time data (e.g., IoT devices, sensors).

**Stream Buffer**: Temporarily stores incoming data to ensure smooth flow.

**Query Processor:** Core unit that executes continuous queries on streaming data.

**Query Optimizer**: Improves query performance by selecting efficient execution plans.

**Catalog Manager:** Stores metadata about streams, queries, and schemas.

**User Interface**: Allows users to define and monitor continuous queries using CQL.

**Output/Result Sink**: Sends processed results to dashboards, databases, or alert systems.

DSMS continuously collects, processes, and delivers real-time insights from streaming data.

**1. One-time Query**

\* Executed \*\*once\*\* on a \*\*static snapshot\*\* of data.

\* Returns a \*\*single, final result\*\* to the user.

\* Common in \*\*traditional DBMS\*\*.

✅ \*Example:\*

SELECT AVG(salary) FROM employees;

(Calculates the average salary once.)

**2. Continuous Query**

\* Runs \*\*continuously\*\* as new data arrives in a \*\*data stream\*\*.

\* Continuously updates results in \*\*real-time\*\*.

\* Used in \*\*Data Stream Management Systems (DSMS)\*\*.

✅ \*Example:\*

Continuously monitor average temperature from live sensor data.

**3. Ad-hoc Query**

\* A \*\*custom, on-demand\*\* query defined by the user \*\*at runtime\*\*.

\* Not pre-defined or stored; used for \*\*exploratory analysis\*\*.

✅ \*Example:\*

SELECT \* FROM sales WHERE region = ‘Asia’ AND amount > 5000;

(User runs it spontaneously to answer a specific question.)

| Query Type | Description | Execution | Example Use |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

| \*\*One-time\*\* | Runs once on stored data | Single result | Traditional DBMS |

| \*\*Continuous\*\* | Runs continuously on data streams | Ongoing result | DSMS / IoT |

| \*\*Ad-hoc\*\* | User-defined on demand | One-time or occasional | Exploratory analysis |

**Issues in Data Stream Query Processing**

1. Unbounded Memory Requirements

\* Data streams are \*\*continuous and infinite\*\*, so storing all data is impossible.

\* DSMS must process data \*\*on the fly\*\* using limited memory.

\* Solutions: \*\*Windowing\*\*, \*\*sampling\*\*, or \*\*synopsis structures\*\*.

2. Approximate Query Answering

\* Since exact answers need large memory and time, DSMS often produces \*\*approximate results\*\*.

\* Uses \*\*probabilistic techniques\*\* or \*\*summaries\*\* to give quick, near-accurate answers.

3. Sliding Windows

\* Instead of processing all past data, queries focus on a \*\*recent subset\*\* of data (e.g., last 10 minutes).

\* Reduces memory use and keeps results relevant to recent trends.

\* Types: \*\*Tumbling\*\*, \*\*Sliding\*\*, and \*\*Session windows\*\*.

4. Batch Processing, Sampling, and Synopses

\* Data is processed in \*\*small batches\*\* or \*\*sampled\*\* to reduce load.

\* \*\*Synopses\*\* (like histograms, sketches, or summaries) store compact representations of data streams.

\* Helps manage speed and memory efficiency.

5. Blocking Operators

\* Some operators (like \*\*sort\*\* or \*\*aggregate\*\*) require \*\*all data\*\* before producing output.

\* This is impractical for continuous streams.

\* Solution: Use \*\*incremental or non-blocking operators\*\* that can output partial results.

Data stream query processing faces challenges like \*\*infinite data, limited memory, need for approximation\*\*, and \*\*non-blocking real-time processing\*\* — handled through \*\*windowing, sampling, and summarization techniques\*\*.

**Batch Processing vs Real-Time Processing**

1. Batch Processing

\* Data is \*\*collected over time\*\* and processed \*\*in large batches\*\*.

\* Suitable for \*\*non-time-sensitive\*\* tasks.

\* Typically used in \*\*data warehouses\*\* and \*\*ETL processes\*\*.

\* \*\*High latency\*\*, but efficient for large volumes.

Example:\* Generating monthly sales reports or payroll processing.

2. Real-Time Processing

\* Data is processed \*\*immediately as it arrives\*\*.

\* Used for \*\*time-critical applications\*\* where instant results are needed.

\* Requires \*\*continuous input, low latency\*\*, and often \*\*stream processing systems\*\* like Spark Streaming or Flink.

Example:\* Fraud detection, stock market monitoring, live sensor analytics.

| Feature | \*\*Batch Processing\*\* | \*\*Real-Time Processing\*\* |

|  |  |  |
| --- | --- | --- |
|  |  |  |

| \*\*Data Input\*\* | Large, accumulated data | Continuous data stream |

| \*\*Processing Frequency\*\* | Periodic (scheduled) | Instant (continuous) |

| \*\*Latency\*\* | High | Very low |

| \*\*Use Case\*\* | Reports, backups, ETL | Monitoring, alerts, IoT |

| \*\*Examples\*\* | Hadoop MapReduce | Apache Spark Streaming, Flink |

Batch processing handles large datasets at intervals, while real-time processing analyzes data instantly as it arrives.

**FILTERING STREAMS:**

“Filtering” tries to observe an infinite stream of data, look at each of its items and quantify whether the item is of interest and should be stored for further evaluation

**THE BLOOM FILTER**

A bloom filter is a probabilistic search filter, a way to describe a desired pattern without specifying it exactly. Bloom filters offer an efficient way to express a search pattern while protecting privacy

**FLAJOLET MARTIN ALGORITHM:**

Flajolet Martin Algorithm, also known as FM algorithm, is used to approximate the number of unique elements in a data stream or database in one pass. The highlight of this algorithm is that it uses less memory space while executing.

**DGIM algorithm** is used for approximate counting of 1’s in a sliding window over a binary data stream.

It helps track the number of recent events (1’s) efficiently using limited memory.

In streaming data, we often need to count how many “1” bits have appeared in the last N bits (e.g., last 1000 sensor readings).

Storing all bits would take O(N) memory.

DGIM gives an approximate count using O(log² N) memory.

**Key Idea:**

Instead of storing every bit, DGIM stores buckets that summarize groups of 1’s:

Each bucket represents a group of 1s and stores:

Timestamp of the most recent 1 in the bucket

Size (number of 1s it represents — always a power of 2)

**Rules of the DGIM Algorithm:**

Each bucket size = 2^k for some integer 𝑘.

At most two buckets of the same size are allowed.

When a new 1 arrives:

* Create a new bucket of size 1.
* If there are more than two buckets of the same size, merge the two oldest into one larger bucket (double size).

Buckets that fall outside the sliding window are deleted.

**Estimating Count of 1’s:**

To count 1’s in the last N bits:

Add the sizes of all buckets within the window.

For the oldest bucket, count half its size (since it may be only partially within the window).

This gives an approximate count.

**Types of Filtering in Big Data Analysis**

Filtering in Big Data refers to \*\*selecting relevant data\*\* from massive datasets to reduce noise and improve processing efficiency.

There are several main types:

1. Data Filtering

\* Removes \*\*irrelevant, duplicate, or corrupted data\*\* before analysis.

\* Ensures only \*\*useful and clean data\*\* is processed.

✅ \*Example:\* Removing null or duplicate entries from a log file.

2. Content-Based Filtering

\* Used mainly in \*\*recommendation systems\*\*.

\* Recommends items similar to what a user liked before, based on item attributes.

✅ \*Example:\* Netflix recommending movies of the same genre you watched.

3. Collaborative Filtering

\* Makes recommendations based on \*\*user behavior patterns\*\*.

\* Two types:

\* \*\*User-based:\*\* Users with similar interests.

\* \*\*Item-based:\*\* Items liked by similar users.

✅ \*Example:\* Amazon showing “Users who bought this also bought…”

4. Bloom Filter

\* A \*\*probabilistic data structure\*\* used to test \*\*whether an element is present\*\* in a dataset.

\* Very \*\*memory efficient\*\*, but may produce \*\*false positives\*\*.

✅ \*Example:\* Used in web caching or detecting spam emails.

5. Stream Filtering

\* Filters \*\*real-time streaming data\*\* as it arrives.

\* Used in \*\*IoT\*\*, \*\*network monitoring\*\*, and \*\*social media analytics\*\*.

✅ \*Example:\* Filtering only error logs from a live server data stream.

| Type | Purpose | Example |

|  |  |  |
| --- | --- | --- |
|  |  |  |

| \*\*Data Filtering\*\* | Clean and reduce data | Remove duplicates |

| \*\*Content-Based Filtering\*\* | Recommend similar items | Movie suggestions |

| \*\*Collaborative Filtering\*\* | Recommend using user behavior | Amazon/Spotify |

| \*\*Bloom Filter\*\* | Fast membership checking | Web cache lookup |

| \*\*Stream Filtering\*\* | Real-time data selection | IoT or log monitoring |