Introduction to Big Data

Understanding Big Data

What is Data?

Data refers to information in a form that can be stored, processed, or transmitted by computers. It can take various forms, including text, numbers, images, audio, and more. Data can be used for analysis, decision-making, and various other purposes.

Here's an example to illustrate different types of data:

Text Data:

Example: A paragraph of text

"The quick brown fox jumps over the lazy dog."

Numeric Data:

Example: Stock prices over the past week

[100.25, 101.50, 99.75, 102.00, 98.50]

Image Data:

Example: A photograph of a sunset

(Binary data representing pixels in an image)

Audio Data:

Example: A recording of a music track

(Digital representation of sound waves)

Video Data:

Example: A video clip from a movie

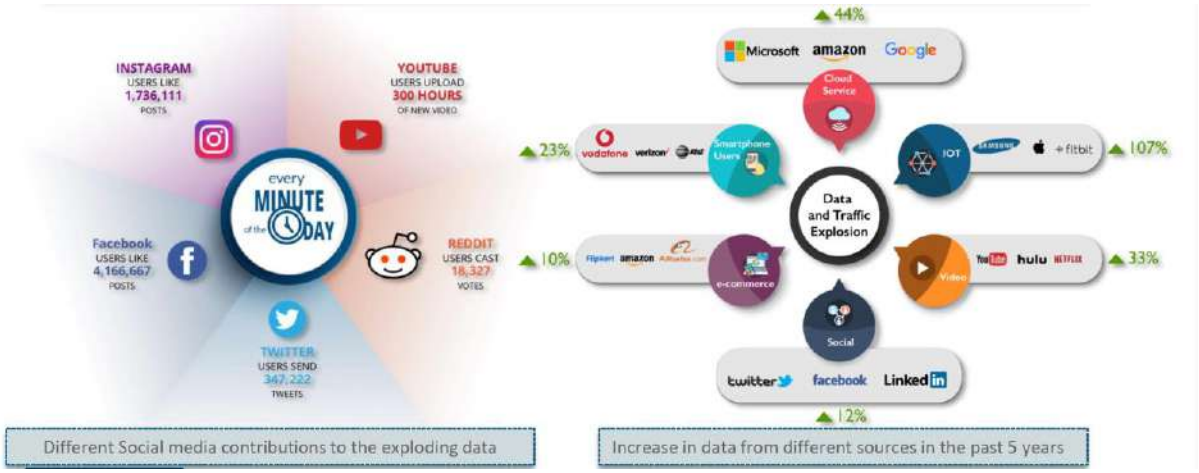
(Series of image frames and audio data)

Data is the foundation for various applications, including machine learning, analytics, business intelligence, and more. Properly organizing, analyzing, and interpreting data can provide valuable insights and drive informed decision-making.

What is Big Data?

Big data refers to extremely **large** and **complex** datasets that cannot be easily managed, processed, or analyzed using traditional data processing tools and methods.

Big data is a term which refers high-volume, high-velocity and high-variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision-making.



How big is the big data?

(NOTE: The statistics shown about big data are taken from the reports of the year 2019.)

The amount of data we produce every day is truly mind-boggling. There are 2.5 quintillion bytes of data created each day at our current pace, but that pace is only accelerating with the growth of the Internet of Things (IoT). Over the last two years alone 90 percent of the data in the world was generated.

For example: 1,000,000,000,000,000,000

The cardinal number of one quintillion is 1,000,000,000,000,000,000. We can say that a quintillion is a million million millions or a billion billions or a million trillions or a thousand quadrillions.

Evolution of Big Data

The evolution of big data has been marked by advancements in technology, increased data generation, and the growing importance of data-driven insights. Here's a general overview of the key stages in the evolution of big data:

Pre-2000s: Data Warehousing and Business Intelligence:

Organizations started adopting data warehousing and business intelligence solutions to store and analyze structured data from internal sources like transaction systems. Data was primarily stored on relational databases, and analytics focused on generating reports and dashboards for decision-making.

Early 2000s: Emergence of Unstructured Data:

With the rise of the internet and digital content, unstructured data in the form of text, images, and multimedia started to gain prominence. Organizations realized the potential value in analyzing and extracting insights from unstructured data sources.

Mid-2000s: Web 2.0 and social media:

The advent of Web 2.0 led to increased user-generated content on social media platforms, generating massive amounts of data. Organizations recognized the need to harness this data for understanding consumer behaviour, sentiment analysis, and targeted marketing.

Late 2000s: Emergence of Hadoop and NoSQL:

Apache Hadoop, an open-source framework, gained popularity for its ability to process and store large volumes of data across distributed clusters. NoSQL databases emerged to handle various types of unstructured and semi-structured data more efficiently than traditional relational databases.

Early 2010s: Big Data Analytics and Machine Learning:

Organizations began adopting advanced analytics techniques, including machine learning and predictive analytics, to extract actionable insights from big data. Cloud computing platforms made it easier to store, process, and analyze massive datasets without significant upfront investments in infrastructure.

Mid-2010s: Internet of Things (IoT) and Real-Time Analytics:

The proliferation of IoT devices, equipped with sensors and connected to the internet, generated real-time streams of data.

Real-time analytics gained importance for processing and reacting to data as it's generated, enabling applications like predictive maintenance and smart cities.

Late 2010s: Data Privacy and Governance:

Growing concerns over data privacy and security led to the development of regulations like the General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA). Organizations had to balance data-driven insights with compliance to ensure responsible data handling.

2020s and Beyond: Big Data and AI Integration:

The integration of big data with artificial intelligence (AI) and machine learning (ML) technologies enabled more sophisticated analysis and decision-making.

AI-driven insights are being used in areas like healthcare diagnostics, autonomous vehicles, personalized marketing, and more.

Ongoing Trends: Edge Computing and Quantum Computing:

Edge computing allows data processing to occur closer to the data source, reducing latency and enabling real-time analysis in remote locations.

Quantum computing holds the potential to process and analyze vast datasets even faster, opening new possibilities in data-intensive research and problem-solving.

Throughout this evolution, the volume, velocity, and variety of data have expanded significantly. As technology continues to advance, big data will likely continue to shape industries, drive innovation, and contribute to the development of new insights and applications.

Failure of Traditional Database in Handling Big Data

"Big data" encompass a wide range of the tremendous data generated from various sources such as mobile devices, digital repositories, and enterprise applications. The data can be structured as well as unstructured. It ranges from terabytes—10^12 bytes—to even exabytes—10^18 bytes. Working with "big data" is complex because of the five v's associated with "big data." Facebook (FB) gets ~10 million new photos uploaded every hour and Google processes over 24 petabytes of data every day. Twitter tweets ~400 million tweets per day. All this shows the magnificent volume, variety, value, and velocity of "big data."

RDBMS for data storage

The relational database management system (or RDBMS) had been the one solution for all database needs. Oracle, IBM (IBM), and Microsoft are the leading players of RDBMS. RDBMS uses structured query language (or SQL) to define, query, and update the database. However, the volume and velocity of business data has changed dramatically in the last couple of years.

Limitations of RDBMS to support "big data"

1. The data size has increased tremendously to the range of petabytes—one petabyte = 1,024 terabytes. RDBMS finds it challenging to handle such huge data volumes. To address this, RDBMS added more central processing units (or CPUs) or more memory to the database management system to scale up vertically.

2. The majority of the data comes in a semi-structured or unstructured format from social media, audio, video, texts, and emails. RDBMS can't categorize unstructured data. They're designed and structured to accommodate structured data such as weblog sensor and financial data.

3. Also, "big data" is generated at a very high velocity. RDBMS lacks in high velocity because it's designed for steady data retention rather than rapid growth. Even if RDBMS is used to handle and store "big data," it will turn out to be very expensive. As a result, the inability of relational databases to handle "big data" led to the emergence of new technologies.

3 Vs of Big Data

The "Three Vs" of big data—Volume, Velocity, and Variety—provide a framework for understanding the key characteristics of large and complex datasets. Here are examples of each of these "Vs" in the context of big data:

Volume: Volume refers to the sheer size or scale of data. Big data involves massive amounts of data that exceed the capacity of traditional data processing systems.

Example: Social media platforms like Facebook generate enormous volumes of data daily. For instance, Facebook users upload over 300 million photos per day, resulting in petabytes of image data.

Velocity: Velocity represents the speed at which data is generated, collected, and processed in real-time or near-real-time.

Example: Stock market trading involves high-velocity data. Thousands of transactions occur every second, with stock prices and trading data changing rapidly. Analyzing this data in real-time is crucial for making informed trading decisions.

Variety: Variety encompasses the different types and formats of data, including structured, semi-structured, and unstructured data from various sources.

Example: Healthcare data is a prime example of data variety. It includes structured data like patient records and lab results, semi-structured data like medical reports, and unstructured data like doctor's notes and medical images (X-rays, MRIs).

In addition to the "Three Vs," some definitions of big data include two more "Vs" to describe its characteristics:

Value: Value emphasizes the ultimate goal of extracting meaningful insights and value from big data. This involves applying analytics and data processing techniques to make informed decisions.

Example: Retail companies analyze large volumes of customer transaction data to identify purchasing patterns and optimize their inventory management, leading to increased sales and cost savings.

Veracity: Veracity relates to the accuracy and trustworthiness of the data. Big data often contains noisy, incomplete, or inconsistent information.

Example: Customer reviews on e-commerce platforms may contain biased or fake reviews, affecting the veracity of the data. Detecting and managing such unreliable information is crucial for accurate analysis.

These "Vs" collectively highlights the challenges and opportunities presented by big data. To harness the potential of big data, organizations need to address issues related to data volume, velocity, variety, veracity, and ultimately extract value by using advanced analytics, machine learning, and other data-driven techniques.

Sources of Big Data

Big data originates from various sources, including digital interactions, sensors, social media, and more. The term "big data" refers to datasets that are too large, complex, and fast-moving for traditional data processing tools to handle. Here are some common sources of big data:

- **Social Media Data:** Platforms like Facebook, Twitter, Instagram, and LinkedIn generate vast amounts of data in the form of posts, comments, likes, shares, and interactions, providing insights into user behaviors, preferences, and trends.
- **Sensor Data:** Internet of Things (IoT) devices, such as smart sensors, wearables, and connected appliances, generate real-time data about environmental conditions, health metrics, and more.
- **Machine and Sensor Logs:** Machines, servers, and devices generate logs that record activities, errors, and events. These logs provide insights into system performance, usage patterns, and potential issues.
- **Web and App Logs:** Websites and applications generate logs that capture user activities, clicks, navigation paths, and other interactions, which can be used for analyzing user behavior and improving user experiences.
-

- **Transaction Data:** Financial transactions, online purchases, and banking activities generate massive amounts of data that can be used for fraud detection, customer insights, and financial analysis.
- **Text and Multimedia Data:** Textual data from documents, emails, and customer feedback, as well as multimedia data such as images and videos, contribute to understanding sentiment, content analysis, and multimedia processing.
- **Geospatial Data:** GPS data, satellite imagery, and location-based services generate geospatial data that can be used for mapping, navigation, urban planning, and more.
- **Genomic Data:** DNA sequencing technologies produce vast datasets in genomics research, contributing to advancements in personalized medicine and genetic studies.
- **Streaming Data:** Real-time data streams, such as stock market data, social media feeds, and sensor data, are processed in real-time to enable rapid decision-making.
- **Public Data:** Government agencies, research organizations, and institutions provide open datasets on topics like demographics, economics, climate, and more, fostering data-driven research and innovation.
- **Healthcare Data:** Electronic health records (EHRs), medical imaging, and patient data contribute to research, diagnostics, and treatment strategies in the healthcare sector.

- **Customer Data:** Customer interactions, purchase histories, loyalty programs, and feedback contribute to understanding consumer behavior and preferences.
- **Search Data:** Search engines record user search queries, providing insights into current trends, interests, and information needs.
- **Scientific Data:** Research in fields like physics, astronomy, and climate science generates massive datasets from experiments and simulations.
- **Surveillance Data:** Security cameras, traffic cameras, and other surveillance systems generate data used for security, traffic management, and law enforcement.

These sources illustrate the diverse nature of big data and the extensive range of applications that can benefit from its analysis. Organizations leverage advanced analytics techniques, machine learning, and artificial intelligence to extract valuable insights and make informed decisions from these datasets.

Different Types of Data

Data may be machine generated or human generated. Human-generated data refers to the data generated as an outcome of interactions of humans with the machines. E-mails, documents, Facebook posts are some of the human-generated data. Machine-generated data refers to the data generated by computer applications or hardware devices without active human intervention. Data from sensors, disaster warning systems, weather forecasting systems, and satellite data are some of the machine-generated data. The data generated by a human in various social media, e-mails sent, and pictures that were taken by them and machine data generated by the satellite.

The machine-generated and human-generated data can be represented by the following primitive types of big data:

●Structured data

●Unstructured data

●Semi-structured data

Structured Data: Data that can be stored in a relational database in table format with rows and columns is called structured data. Structured data often generated by business enterprises exhibits a high degree of organization and can easily be processed using data mining tools and can be queried and retrieved using the primary key field.

Examples of Structured Data

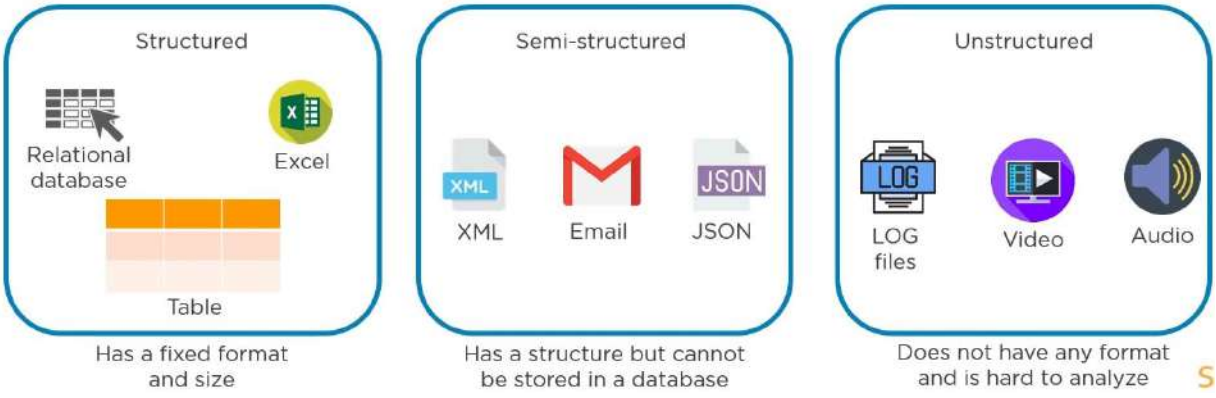
This type of data is generated by both humans and machines. There are numerous examples of structured data from machines, such as POS data like quantity, barcodes, and weblog statistics. Similarly, anyone who works on data would have used spreadsheets once in their lifetime, which is a classic case of structured data generated by humans. Due to the organization of structured data, it is easier to analyze than both semi-structured and unstructured data.

Semi-Structured Data: Semi-structured data are those that have a structure but do not fit into the relational database. Semi-structured data are organized, which makes it easier to analyze when compared to unstructured data. JSON and XML are examples of semi-structured data.

Examples of Semi-Structured Data

An example of data in a semi-structured format is delimited files. It contains elements that can break down the data into separate hierarchies. Similarly, in digital photographs, the image does not have a pre-defined structure itself but has certain structural attributes making them semi-structured. For instance, if you take a photo from a smartphone, it would have some structured attributes like geotag, device ID, and DateTime stamp. After you save them, you can assign tags to images such as ‘pet’ or ‘dog’ to provide a structure.

Unstructured Data: Data that are raw, unorganized, and do not fit into the relational database systems are called unstructured data. Nearly 80% of the data generated are unstructured. Examples of unstructured data include video, audio, images, log files, and social media posts. Unstructured data usually reside on either text files or binary files. Data that reside in binary files do not have any identifiable internal structure, for example, audio, video, and images. Data that reside in text files are e-mails, social media posts, pdf files, and word processing documents.



The following points highlight the differences between structured data vs. unstructured data vs. semi-structured data:

Organization: Structured data is well organized. Therefore, it has the highest level of organization. Semi-structured data is partially organized; hence the level of organizing is lesser than structured data but higher than that of unstructured data. Lastly, the latter category is not organized at all.

Flexibility and Scalability: Structured data is relational database or schema dependent, therefore less flexible and difficult to scale, while semi-structured data is more flexible and simpler to scale than structured data. However, unstructured data doesn't have a schema that makes it the most flexible and scalable out of the other two.

Versioning: Since structured data is based on a relational database, versioning is performed over tuples, rows, and tables. On the other hand, in semi-structured data, tuples or graphs are possible as only a partial database is supported. Lastly, in unstructured data, versioning is likely as a whole data as there's no database support.

Transaction Management: In structured data, data concurrency is available and, therefore, usually preferred for the multitasking process. In semi-structured data, the transaction gets adapted from DBMS, but still, data concurrency isn't available. Lastly, in structured data, neither transaction management nor data concurrency is present.

Big Data Infrastructure and Technology

Big Data Infrastructure

The core components of big data technologies are the tools and technologies that provide the capacity to store, process, and analyze the data. The method of storing the data in tables was no longer supportive with the evolution of data with 3 Vs, namely volume, velocity, and variety. The robust RBDMS was no longer cost effective. The scaling of RDBMS to store and process huge amount of data became expensive. This led to the emergence of new technology, which was highly scalable at very low cost.

The key technologies include

●Hadoop ●HDFS ●MapReduce

Hadoop – Apache Hadoop, written in Java, is open-source framework that supports processing of large data sets. It can store a large volume of structured, semi-structured, and unstructured data in a distributed file system and process them in parallel. It is a highly scalable and cost-effective storage platform. Scalability of Hadoop refers to its capability to sustain its performance even under highly increasing loads by adding more nodes. Hadoop files are written once and read many times. The contents of the files cannot be changed. A large number of computers interconnected working together as a single system is called a cluster. Hadoop clusters are designed to store and analyze the massive amount of disparate data in distributed computing environments in a cost-effective manner.

Hadoop Distributed File System (HDFS) – HDFS is designed to store large data sets with streaming access pattern running on low-cost commodity hardware. It does not require highly reliable, expensive hardware. The data set is generated from multiple sources, stored in an HDFS file system in a write-once, read-many-times pattern, and analyses are performed on the data set to extract knowledge from it.

MapReduce – MapReduce is the batch-processing programming model for the Hadoop framework, which adopts a divide-and-conquer principle. It is highly scalable, reliable, and fault tolerant, capable of processing input data with any format in parallel and distributed computing environments supporting only batch workloads. Its performance reduces the processing time significantly compared to the traditional batch-processing paradigm, as the traditional approach was to move the data from the storage platform to the processing platform,

whereas the MapReduce processing paradigm resides in the framework where the data actually resides.

Big Data Technology

With the advancement in technology, the ways the data are generated, captured, processed, and analyzed are changing. The efficiency in processing and analyzing the data has improved with the advancement in technology. Thus, technology plays a great role in the entire process of gathering the data to analyzing them and extracting the key insights from the data.

Apache Hadoop is an open-source platform that is one of the most important technologies of big data. Hadoop is a framework for storing and processing the data. Hadoop was originally created by Doug Cutting and Mike Cafarella, a graduate student from the University of Washington. They jointly worked with the goal of indexing the entire web, and the project is called "Nutch." The concept of MapReduce and GFS were integrated into Nutch, which led to the evolution of Hadoop. The word "Hadoop" is the name of the toy elephant of Doug's son. The core components of Hadoop are HDFS, Hadoop common, which is a collection of common utilities that support other Hadoop modules, and MapReduce.

Apache Hadoop is an open-source framework for distributed storage and for processing large data sets. Hadoop can store petabytes of structured, semi-structured, or unstructured data at low cost. The low cost is due to the cluster of commodity hardware on which Hadoop runs.

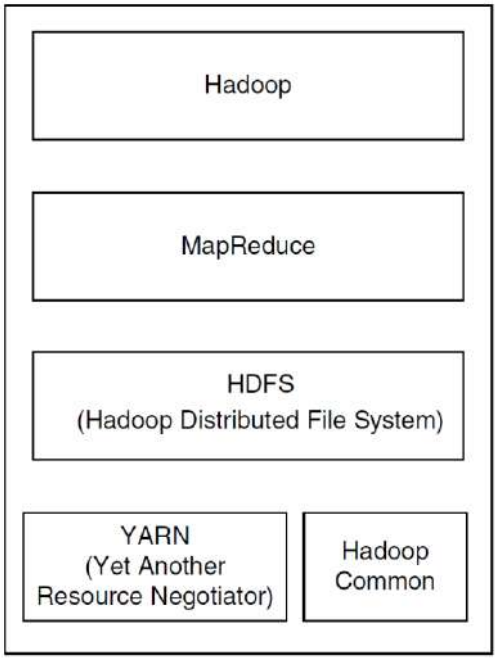


Figure 1.12 Hadoop core components.

For Hadoop, Hadoop Distributed File System (HDFS) and MapReduce refer above.

YARN – YARN is the acronym for Yet Another Resource Negotiator and is an open-source framework for distributed processing. It is the key feature of Hadoop version 2.0 of the Apache software foundation. In Hadoop 1.0 MapReduce was the only component to process the data in distributed environments. Limitations of classical MapReduce have led to the evolution of YARN. The cluster resource management of MapReduce in Hadoop 1.0 was taken over by YARN in Hadoop 2.0. This has lightened up the task of MapReduce and enables it to focus on the data processing part. YARN enables Hadoop to run jobs other than MapReduce jobs as well.

Hadoop common – Hadoop common is a collection of common utilities, which supports other Hadoop modules. It is considered as the core module of Hadoop as it offers essential services. Hadoop common has the scripts and Java Archive (JAR) files that are required to start Hadoop.

Challenges Faced by Big Data Technology

Indeed, we are facing a lot of challenges when it comes to dealing with the data. Some data are structured that could be stored in traditional databases, while some are videos, pictures, and documents, which may be unstructured or semi-structured, generated by sensors, social media, satellite, business transactions, and much more. Though these data can be managed independently, the real challenge is how to make sense by integrating disparate data from diversified sources.

- Heterogeneity and incompleteness
- Volume and velocity of the data
- Data storage
- Data privacy

Heterogeneity and Incompleteness: The data types of big data are heterogeneous in nature as the data is integrated from multiple sources and hence has to be carefully structured and presented as homogenous data before big data analysis. The data gathered may be incomplete, making the analysis much more complicated. Consider an example of a patient online health record with his name, occupation, birth data, medical ailment, laboratory test results, and previous medical history. If one or more of the above details are missing in multiple records, the analysis cannot be performed as it may not turn out to be valuable. In some scenarios a

NULL value may be inserted in the place of missing values, and the analysis may be performed if that particular value does not have a great impact on the analysis and if the rest of the available values are sufficient to produce a valuable outcome.

Volume and Velocity of the Data: Managing the massive and ever-increasing volume of big data is the biggest concern in the big data era. In the past, the increase in the data volume was handled by appending additional memory units and computer resources. But the data volume was increasing exponentially, which could not be handled by traditional existing database storage models. The larger the volume of data, the longer the time consumed for processing and analysis.

The challenge faced with velocity does not only mean rate at which data arrives from multiple sources but also the rate at which data has to be processed and analyzed in the case of real-time analysis. For example, in the case of credit card transactions, if fraudulent activity is suspected, the transaction has to be declined in real time.

Data Storage: The volume of data contributed by social media, mobile Internet, online retailers, and so forth, is massive and was beyond the handling capacity of traditional databases. This requires a storage mechanism that is highly scalable to meet the increasing demand. The storage mechanism should be capable of accommodating the growing data, which is complex in nature. When the data volume is previously known, the storage capacity required is predetermined. But in case of streaming data, the required storage capacity is not predetermined. Hence, a storage mechanism capable of accommodating this streaming data is required. Data storage should be reliable and fault tolerant as well.

Data stored has to be retrieved at a later point in time. This data may be purchasing history of a customer, previous releases of a magazine, employee details of a company, twitter feeds, images captured by a satellite, patient records in a hospital, financial transactions of a bank customer, and so forth. When a business analyst has to evaluate the improvement of sales of a company, she has to compare the sales of the current year with the previous year. Hence, data has to be stored and retrieved to perform the analysis.

Data Privacy: Privacy of the data is yet another concern growing with the increase in data volume. Inappropriate access to personal data, EHRs, and financial transactions is a social problem affecting the privacy of the users to a great extent. The data has to be shared limiting the extent of data disclosure and ensuring that the data shared is sufficient to extract business knowledge from it. Whom access to the data should be granted to, limit of access to the data,

and when the data can be accessed should be predetermined to ensure that the data is protected. Hence, there should be a deliberate access control to the data in various stages of the big data life cycle, namely data collection, storage, and management and analysis. The research on big data cannot be performed without the actual data, and consequently the issue of data openness and sharing is crucial. Data sharing is tightly coupled with data privacy and security. Big data service providers hand over huge data to the professionals for analysis, which may affect data privacy. Financial transactions contain the details of business processes and credit card details. Such kind of sensitive information should be protected well before delivering the data for analysis.

Big Data Life Cycle

Big data yields big benefits, starting from innovative business ideas to unconventional ways to treat diseases, overcoming the challenges. The challenges arise because so much of the data is collected by the technology today. Big data technologies are capable of capturing and analyzing them effectively. Big data infrastructure involves new computing models with the capability to process both distributed and parallel computations with highly scalable storage and performance.

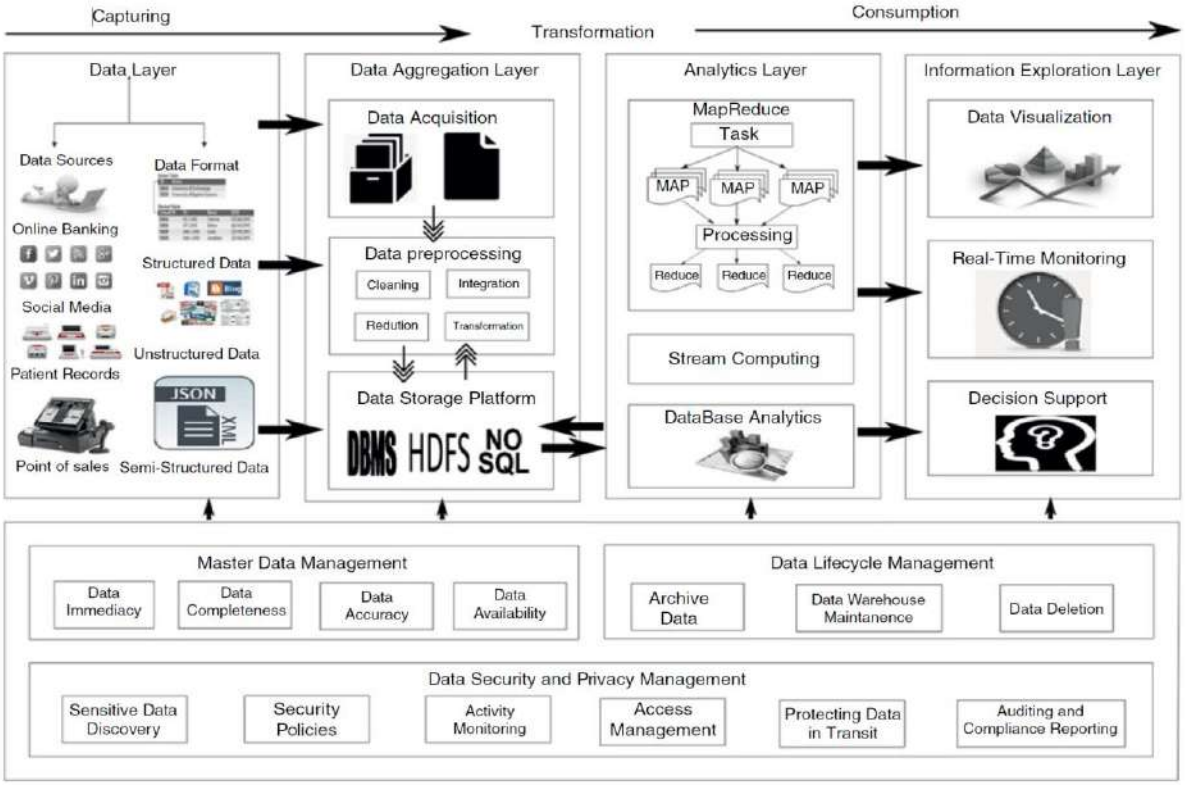


Figure 1.10 Big data life cycle.

Some of the big data components include Hadoop (framework), HDFS (storage), and MapReduce (processing). Data arriving at high velocity from multiple sources with different data formats are captured. The captured data is stored in a storage platform such as HDFS and NoSQL and then preprocessed to make the data suitable for analysis. The preprocessed data stored in the storage platform is then passed to the analytics layer, where the data is processed using big data tools such as MapReduce and YARN and analysis is performed on the processed data to uncover hidden knowledge from it. Analytics and machine learning are important concepts in the life cycle of big data. Text analytics is a type of analysis performed on unstructured textual data. With the growth of social media and e-mail transactions, the importance of text analytics has surged up. Predictive analysis on consumer behavior and

consumer interest analysis are all performed on the text data extracted from various online sources such as social media, online retailing websites, and much more. Machine learning has made text analytics possible.

The analyzed data is visually represented by visualization tools such as Tableau to make it easily understandable by the end user to make decisions.

Big Data Generation

The first phase of the life cycle of big data is the data generation. The scale of data generated from diversified sources is gradually expanding.

Data Aggregation

The data aggregation phase of the big data life cycle involves collecting the raw data, transmitting the data to the storage platform, and preprocessing them.

Data Acquisition

Data acquisition in the big data world means acquiring the high-volume data arriving at an ever-increasing pace. Data acquisition refers to the process of collecting, measuring, and recording data from various sources or sensors for the purpose of analysis, monitoring, control, or further processing. The raw data thus collected is transmitted to a proper storage infrastructure to support processing and various analytical applications.

Data Preprocessing

Data preprocessing is an important process performed on raw data to transform it into an understandable format and provide access to consistent and accurate data. The data generated from multiple sources are erroneous, incomplete, and inconsistent because of their massive volume and heterogeneous sources, and it is meaningless to store useless and dirty data. Additionally, some analytical applications have a crucial requirement for quality data. Hence, for effective, efficient, and accurate data analysis, systematic data preprocessing is essential. The quality of the source data is affected by various factors. There are several steps involved in data preprocessing:

1) Data integration
2) Data cleaning
3) Data reduction
4) Data transformation

1) Data Integration

Data integration involves combining data from different sources to give the end users a unified data view. Several challenges are faced while integrating data; as an example, while extracting data from the profile of a person, the first name and family name may be interchanged in a certain culture, so in such cases integration may happen incorrectly.

2) Data Cleaning

The data-cleaning process fills in the missing values, corrects the errors and inconsistencies, and removes redundancy in the data to improve the data quality. The larger the heterogeneity of the data sources, the higher the degree of dirtiness. Consequently, more cleaning steps may be involved.

Data cleaning involves several steps such as spotting or identifying the error, correcting the error or deleting the erroneous data, and documenting the error type. To detect the type of error and inconsistency present in the data, a detailed analysis of the data is required.

Data redundancy is the data repetition, which increases storage cost and transmission expenses and decreases data accuracy and reliability. The various techniques involved in handling data redundancy are redundancy detection and data compression.

Missing values can be filled in manually, but it is tedious, time-consuming, and not appropriate for the massive volume of data. A global constant can be used to fill in all the missing values, but this method creates issues while integrating the data; hence, it is not a foolproof method.

3) Data Reduction

Data processing on massive data volume may take a long time, making data analysis either infeasible or impractical. Data reduction is the concept of reducing the volume of data or reducing the dimension of the data, that is, the number of attributes. Data reduction techniques are adopted to analyze the data in reduced format without losing the integrity of the actual data and yet yield quality outputs.

Data reduction techniques include data compression, dimensionality reduction.

Data compression techniques are applied to obtain the compressed or reduced representation of the actual data. If the original data is retrieved back from the data that is being compressed without any loss of information, then it is called lossless data reduction. On the other hand, if the data retrieval is only partial, then it is called lossy data reduction.

Dimensionality reduction is the reduction of a number of attributes, and the techniques include wavelet transforms where the original data is projected into a smaller space and attribute subset selection, a method which involves removal of irrelevant or redundant attributes.

4) Data Transformation

Data transformation refers to transforming or consolidating the data into an appropriate format and converting them into logical and meaningful information for data management and analysis. The real challenge in data transformation comes into the picture when fields in one system do not match the fields in another system. Before data transformation, data cleaning and manipulation takes place. Organizations are collecting a massive amount of data, and the volume of the data is increasing rapidly. The data captured are transformed using ETL tools.

Data transformation involves the following strategies:

Smoothing: which removes noise from the data by incorporating binning, clustering, and regression techniques.

Aggregation: which applies summary or aggregation on the data to give a consolidated data.

Generalization: which is normally viewed as climbing up the hierarchy where the attributes are generalized to a higher level overlooking the attributes at a lower level.

Discretization: which is a technique where raw values in the data (e.g., age) are replaced by conceptual labels (e.g., teen, adult, senior) or interval labels (e.g., 0–9, 10–19, etc.)

Big Data Analytics

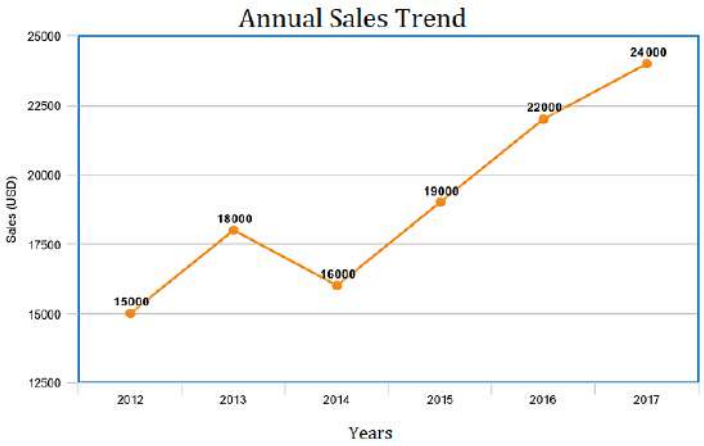
Businesses are recognizing the unrevealed potential value of this massive data and putting forward the tools and technologies to capitalize on the opportunity. The key to deriving business value from big data is the potential use of analytics. Collecting, storing, and preprocessing the data creates a little value. It has to be analyzed and the end users must make decisions out of the results to derive business value from the data. Big data analytics is a fusion of big data technologies and analytic tools. Analytics is not a new concept: many analytic techniques, namely, regression analysis and machine learning, have existed for many years. The different types of analytics are descriptive analytics, predictive analytics, and prescriptive analytics.

Visualizing Big Data

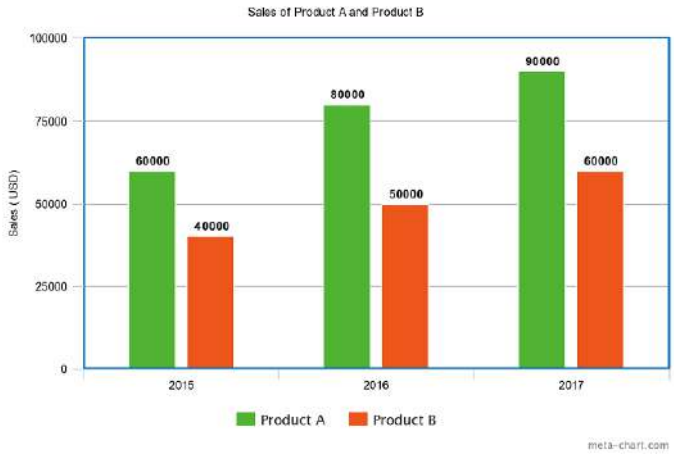
Visualization makes the life cycle of big data complete assisting the end users to gain insights from the data. From executives to call center employees, everyone wants to extract knowledge from the data collected to assist them in making better decisions. Regardless of the volume of data, one of the best methods to discern relationships and make crucial decisions is to adopt advanced data analysis and visualization tools.

Line graphs, bar charts, scatterplots, bubble plots, and pie charts are conventional data visualization techniques.

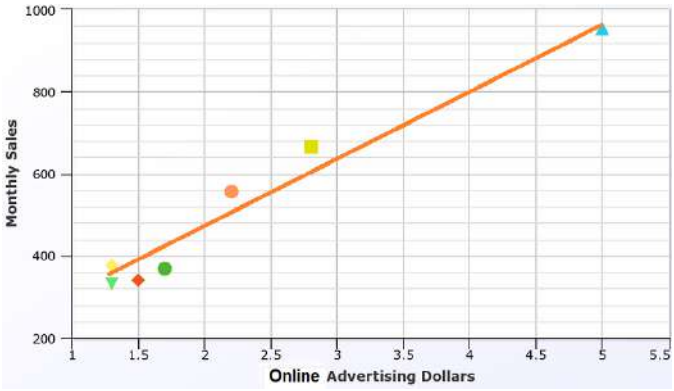
Line graphs are used to depict the relationship between one variable and another.



Bar charts are used to compare the values of data belonging to different categories represented by horizontal or vertical bars, whose heights represent the actual values.

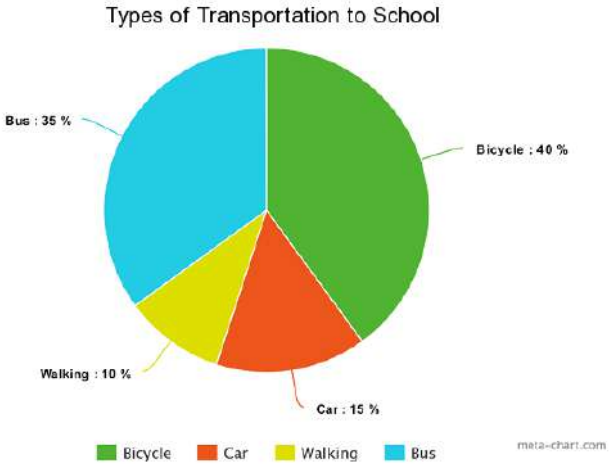


Scatterplots are used to show the relationship between two variables (X and Y).



A bubble plot is a variation of a scatterplot where the relationships between X and Y are displayed in addition to the data value associated with the size of the bubble.

Pie charts are used where parts of a whole phenomenon are to be compared.



Big Data Applications and Use Cases

Big Data Applications

Big data applications refer to the use of large and complex datasets to derive insights, make informed decisions, and create value across various industries. These applications leverage advanced technologies and tools to process, analyze, and interpret massive volumes of data that traditional data processing methods would struggle to handle effectively. Here are some notable big data applications across different sectors:

Healthcare:

Big data is transforming healthcare by enabling personalized medicine, disease prediction, and improved patient outcomes. Electronic health records (EHRs), medical images, genetic data, and wearable devices generate vast amounts of data. Big data analytics can:

- Identify patterns and correlations in patient data to predict disease risks and outcomes.
- Develop precision medicine approaches by tailoring treatments to an individual's genetic makeup.
- Monitor patient vitals in real-time to provide early intervention.
- Optimize hospital operations by predicting patient admissions and resource requirements.

Finance:

In the finance sector, big data analytics is used for risk management, fraud detection, and investment strategies. Data from transactions, market feeds, and customer interactions is processed to:

- Identify fraudulent activities and prevent financial crimes in real time.
- Assess credit risk by analyzing historical financial data and customer behavior.
- Develop algorithmic trading strategies by analyzing market trends and historical data.
- Improve customer experience through personalized financial recommendations.

Retail and E-commerce:

Big data drives e-commerce success through recommendation systems, customer insights, and supply chain optimization. Analyzing customer behavior, browsing history, and purchase patterns helps:

- Provide personalized product recommendations to enhance the shopping experience.
- Optimize inventory management and supply chain logistics for efficient operations.
- Predict customer preferences and trends to make data-driven marketing decisions.
- Monitor sentiment on social media to gauge product popularity and brand perception.

Manufacturing and Industry:

Industry 4.0 leverages big data for predictive maintenance, process optimization, and quality control. Sensors, IoT devices, and production data are analyzed to:

- Predict equipment failures and schedule maintenance before breakdowns occur.
- Optimize production processes to reduce waste, increase efficiency, and lower costs.
- Monitor product quality in real time to ensure adherence to standards.
- Enable remote monitoring and control of manufacturing operations.

Telecommunications:

Big data helps telecom companies manage network performance, optimize resource allocation, and improve customer experience. Network logs, call data records, and customer feedback are analyzed to:

- Monitor network performance and predict network congestion.
- Optimize network resource allocation for better coverage and bandwidth.
- Understand customer behavior and preferences for targeted marketing campaigns.
- Analyze customer complaints and feedback to enhance service quality.

Energy and Utilities:

Big data is used in the energy sector for smart grid management, energy optimization, and renewable energy integration. Data from sensors, smart meters, and grid equipment is used to:

- Monitor energy consumption patterns and predict peak demand.
- Optimize energy distribution and manage power supply in real time.
- Integrate renewable energy sources efficiently based on weather forecasts.
- Improve maintenance planning for power generation and distribution equipment.

Transportation and Logistics:

Big data is employed in transportation and logistics to optimize routes, enhance safety, and manage fleet operations. GPS data, sensors, and traffic data help:

- Optimize route planning to minimize fuel consumption and delivery times.
- Predict traffic congestion and adapt routes in real time.
- Monitor driver behavior to improve safety and compliance with regulations.
- Enhance supply chain visibility for efficient inventory management.

Smart Cities and Urban Planning:

Big data contributes to the development of smart cities by improving infrastructure, public services, and citizen engagement. Data from IoT devices, sensors, and social media platforms is used to:

- Optimize traffic flow and parking management in urban areas.
- Monitor air quality, noise levels, and waste management for environmental sustainability.
- Enhance public safety through real-time monitoring of security cameras and sensors.
- Engage citizens by providing data-driven services and involving them in urban planning decisions.

Agriculture:

Big data applications in agriculture, known as precision agriculture, involve analyzing data from sensors, satellites, and drones to optimize crop yield, reduce resource usage, and improve farming practices. Big data helps:

- Monitor soil conditions, moisture levels, and weather patterns for efficient irrigation.
- Optimize fertilizer and pesticide application based on crop health indicators.
- Predict disease outbreaks and pest infestations to take timely preventive measures.

- Increase crop yield and minimize environmental impact through data-driven decision-making.

Media and Entertainment:

Big data is used in the media and entertainment industry for content recommendation, audience analysis, and content creation. Viewer preferences, streaming behavior, and social media interactions are analyzed to:

- Provide personalized content recommendations to viewers.
- Analyze audience sentiment and engagement to tailor marketing campaigns.
- Optimize content creation and distribution based on viewer feedback and trends.
- Enhance user experience through data-driven insights into viewing patterns.

These examples showcase how big data applications are transforming various industries by harnessing the power of data to derive insights, make informed decisions, and drive innovation. The ability to process and analyze large datasets in real time or near-real time is revolutionizing the way organizations operate and interact with their customers, leading to improved efficiency, effectiveness, and competitiveness.

Big Data Use Cases

Big data use cases encompass a wide range of applications across various industries. These use cases highlight how organizations can leverage massive datasets to drive innovation, improve decision-making, and create value. Here are some prominent big data use cases:

1) **Healthcare and Life Sciences:**

   Big data is transforming healthcare and life sciences with data-driven insights for improved patient care, disease management, and medical research.

   Predictive Analytics: Analyzing patient data, medical records, and genetic information helps predict disease risks and outcomes, allowing for early intervention and personalized treatment plans.

Genomics Research: Big data analytics accelerates genomics research by processing and analyzing massive DNA sequencing datasets, identifying genetic markers for diseases, and enabling precision medicine.

Drug Discovery: Data analysis of molecular interactions and clinical trial data helps identify potential drug candidates, predict drug efficacy, and streamline drug development processes.

2) **Financial Services:**

The financial industry leverages big data to enhance risk management, fraud detection, and customer service.

Fraud Detection: Big data analytics detects unusual patterns in financial transactions, enabling real-time fraud detection and prevention.

Algorithmic Trading: Analyzing market data and historical trends helps develop algorithmic trading strategies for better investment decisions.

Credit Scoring: Evaluating vast amounts of customer data improves credit risk assessment, leading to more accurate lending decisions.

Customer Insights: Analyzing customer behavior and transaction data provides insights for targeted marketing, personalized offers, and improved customer experiences.

3) **Retail and E-commerce:**

Big data transforms the retail industry by enabling personalized shopping experiences, optimizing supply chains, and enhancing marketing strategies.

Recommendation Systems: Analyzing customer purchase history and browsing behavior leads to personalized product recommendations, increasing sales and customer satisfaction.

Inventory Management: Big data helps optimize inventory levels, reduce stockouts, and minimize overstock situations through demand forecasting.

Dynamic Pricing: Analyzing market trends and competitor prices allows retailers to adjust pricing dynamically to maximize profits.

Customer Sentiment Analysis: Monitoring social media and customer reviews provides insights into brand perception and helps adapt marketing strategies accordingly.

4) **Manufacturing and Industry 4.0:**

Big data is integral to Industry 4.0 initiatives, enabling predictive maintenance, process optimization, and quality control.

Predictive Maintenance: Analyzing sensor data from machinery detects anomalies and predicts equipment failures, reducing downtime and maintenance costs.

Quality Control: Real-time analysis of production data ensures product quality and reduces defects by identifying manufacturing issues early.

Supply Chain Optimization: Big data improves supply chain visibility, allowing manufacturers to optimize logistics, reduce lead times, and enhance production planning.

Energy Efficiency: Monitoring energy consumption data optimizes resource utilization, reduces energy costs, and supports sustainability efforts.

5) **Telecommunications:**

Big data is essential for managing network performance, optimizing resource allocation, and enhancing customer experience in the telecommunications industry.

Network Optimization: Analyzing network data and performance metrics helps optimize network infrastructure, reducing downtime and improving service quality.

Customer Experience: Analyzing call records and customer interactions allows providers to address issues promptly, leading to improved customer satisfaction.

Churn Prediction: Data analysis helps identify customers at risk of churning, enabling targeted retention efforts and improved customer retention rates.

6) **Energy and Utilities:**

The energy sector uses big data to optimize energy distribution, monitor renewable energy sources, and improve overall efficiency.

Smart Grid Management: Analyzing real-time data from smart meters and sensors helps manage energy distribution, predict peak demand, and reduce energy waste.

Renewable Energy Integration: Big data supports the integration of renewable energy sources by predicting solar and wind energy production based on weather data.

Predictive Maintenance: Analyzing data from energy infrastructure improves equipment maintenance scheduling and extends asset lifetimes.

7) **Smart Cities and Urban Planning:**

Big data is integral to developing smart cities, improving infrastructure, and enhancing public services.

Traffic Management: Analyzing data from traffic sensors and GPS devices helps optimize traffic flow, reduce congestion, and improve urban mobility.

Environmental Monitoring: Real-time analysis of data from air quality sensors and weather stations helps manage pollution levels and supports environmental conservation efforts.

Emergency Response: Big data aids emergency services by providing real-time insights during disasters, optimizing response times, and managing resources effectively.

8) **Agriculture:**

In agriculture, big data is used for precision farming, optimizing crop yield, and sustainable resource management.

Precision Agriculture: Analyzing data from sensors, satellites, and drones helps optimize irrigation, fertilizer application, and pest control, leading to increased crop yields.

Weather Forecasting: Analyzing weather data helps farmers plan planting and harvesting times and mitigate risks due to adverse weather conditions.

Soil Health Monitoring: Analyzing soil data helps farmers assess soil health, nutrient levels, and pH, leading to improved crop management practices.

9) **Media and Entertainment:**

Big data enhances the media and entertainment industry by enabling personalized content delivery, audience analysis, and content creation.

Content Recommendation: Analyzing user viewing behavior and preferences drives personalized content recommendations, increasing viewer engagement.

Audience Analysis: Analyzing social media interactions and user feedback helps media companies understand audience sentiment and preferences for better content strategies.

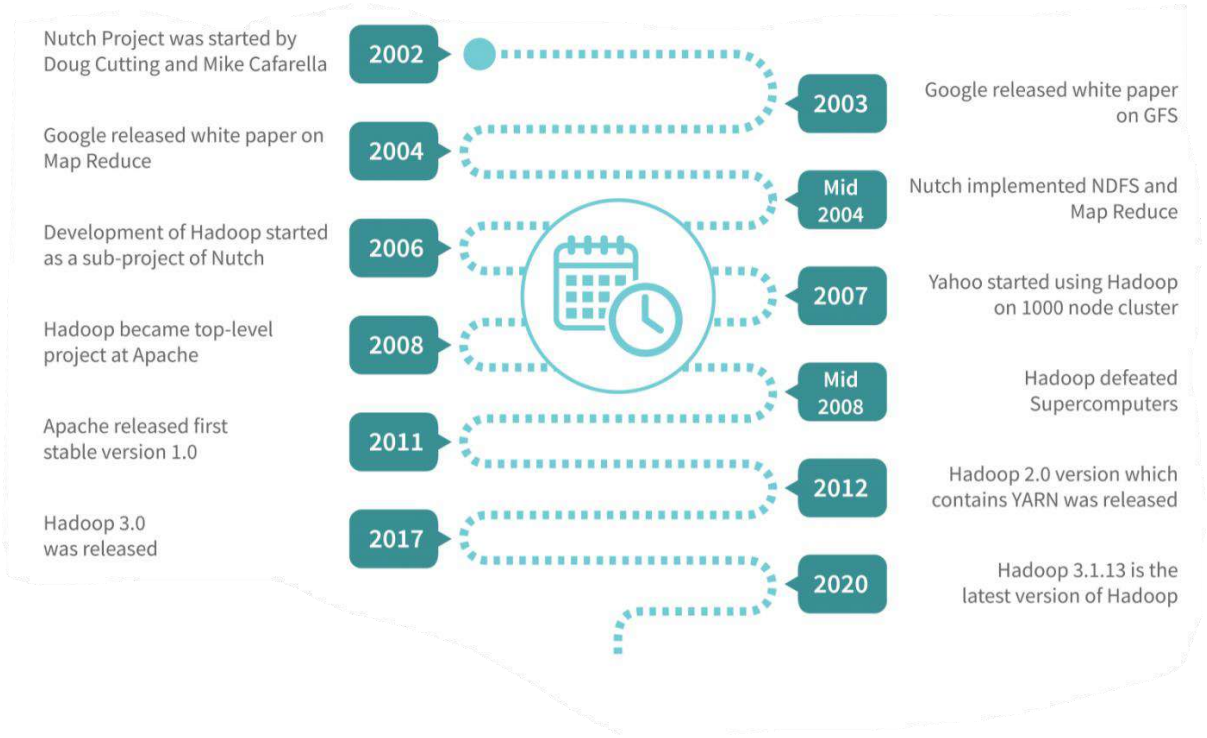
Content Creation: Big data supports data-driven content creation by analyzing viewer feedback and trends to create content that resonates with the audience.

These use cases demonstrate how big data is reshaping various industries, improving processes, optimizing resource utilization, and providing valuable insights for decision-making and innovation. The application of big data technologies continues to evolve, enabling organizations to unlock the potential of their data and derive meaningful value from it.

Introduction to Hadoop

A brief history of Apache Hadoop

Hadoop, a groundbreaking open-source framework, has played a pivotal role in the evolution of big data processing and analytics. Its history can be summarized as follows:



HDFS Architecture / The Design of HDFS

Hadoop comes with a distributed filesystem called HDFS, which stands for Hadoop Distributed Filesystem.

Hadoop Distributed File System follows the master-slave architecture. Each cluster comprises a **single master node** and **multiple slave nodes**. Internally the files get divided into one or more blocks, and each block is stored on different slave machines depending on the replication factor.

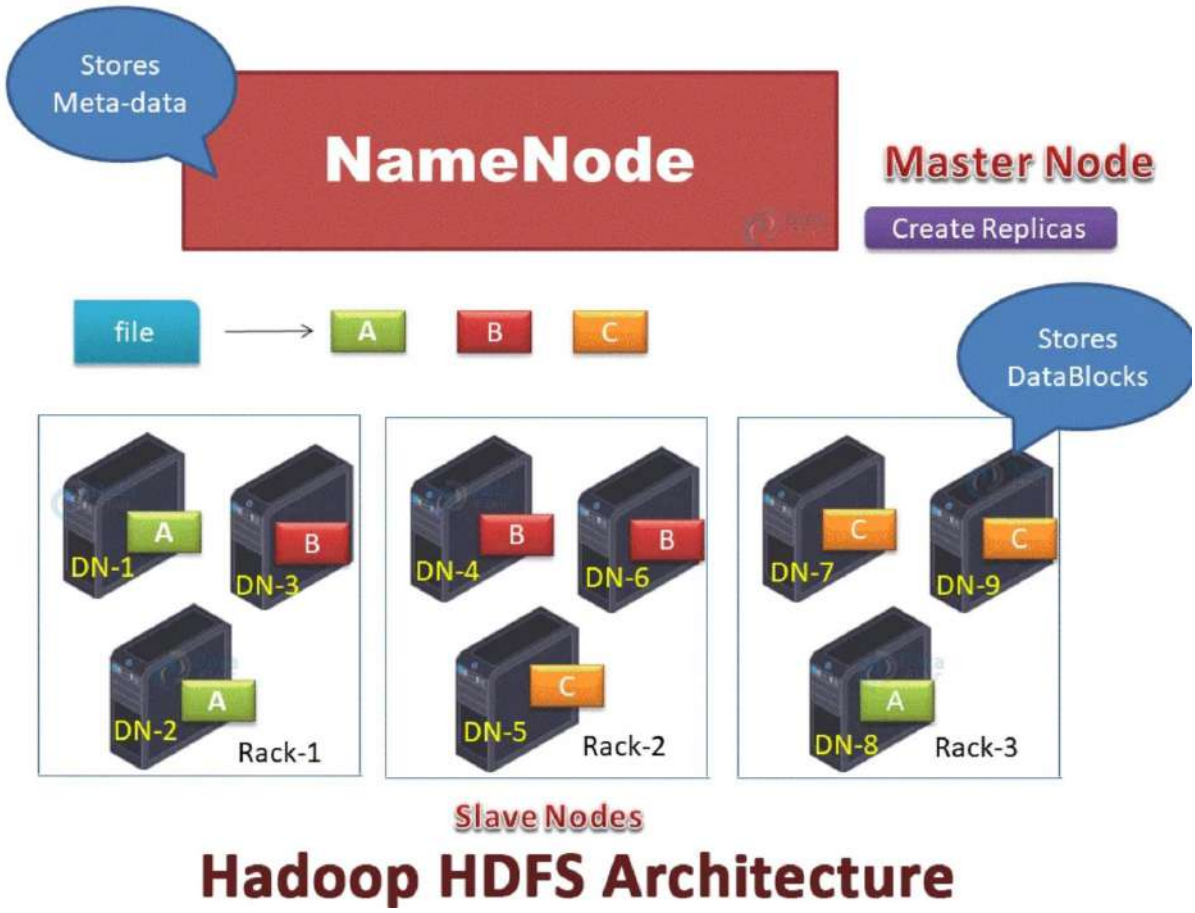
In Hadoop HDFS, NameNode is the master node and DataNodes are the slave nodes. The file in HDFS is stored as data blocks.

The master node stores and manages the file system namespace, that is information about blocks of files like block locations, permissions, etc. The slave nodes store data blocks of files.

The file is divided into blocks (A, B, C in the below figure). These blocks get stored on different DataNodes based on the Rack Awareness Algorithm. Block A on DataNode-1(DN-1), block B on DataNode-6(DN-6), and block C on DataNode-7(DN-7).

To provide Fault Tolerance, replicas of blocks are created based on the replication factor.

In the below figure, 2 replicas of each block is created (using default replication factor 3). Replicas were placed on different DataNodes, thus ensuring data availability even in the case of DataNode failure or rack failure.



HDFS is an Open-source component of the Apache Software Foundation that manages data.

HDFS has scalability, availability, and replication as key features. Name nodes, data nodes, and blocks all make up the architecture of HDFS.

HDFS is fault-tolerant and is replicated.

Files are distributed across the cluster systems using the Name node and Data Nodes.

HDFS is composed of master-slave architecture, which includes the following elements:

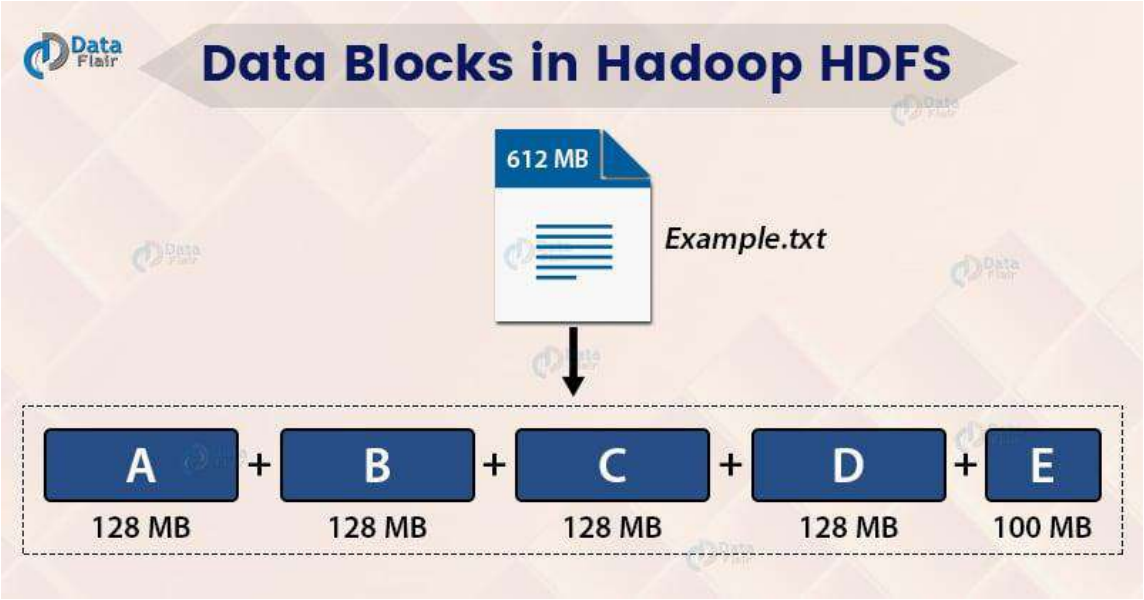
HDFS Concepts –

- **Blocks,**
- **Name Node,**
- **Data Node,**
- **Secondary Name Node / Check Point Node,**
- **Block Caching,**
- **HDFS Federation and**
- **HDFS High Availability**

Blocks

A disk has a block size, which is the minimum amount of data that it can read or write. Filesystems for a single disk build on this by dealing with data in blocks, which are an integral multiple of the disk block size. Filesystem blocks are typically a few kilobytes in size, whereas disk blocks are normally 512 bytes.

HDFS, too, has the concept of a block, but it is a much larger unit—128 MB by default. Like in a filesystem for a single disk, files in HDFS are broken into block-sized chunks, which are stored as independent units. Unlike a filesystem for a single disk, a file in HDFS that is smaller than a single block does not occupy a full block's worth of underlying storage. (For example, a 1 MB file stored with a block size of 128 MB uses 1 MB of disk space, not 128 MB.)



For example, if there is a file of size 612 Mb, then HDFS will create four blocks of size 128 Mb and one block of size 100 Mb.

The file of a smaller size does not occupy the full block size space in the disk.

For example, the file of size 2 Mb will occupy only 2 Mb space in the disk.

The user doesn't have any control over the location of the blocks.

Why are blocks in HDFS huge?

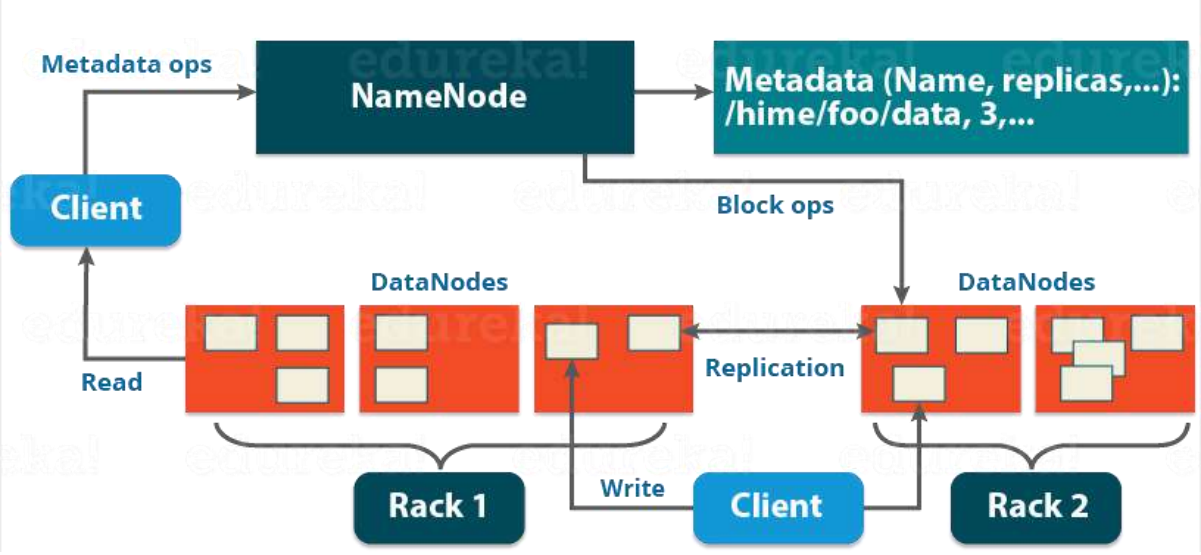
The default size of the HDFS data block is 128 MB. The reasons for the large size of blocks are:

- To minimize the cost of seek: For the large size blocks, time taken to transfer the data from disk can be longer as compared to the time taken to start the block. This results in the transfer of multiple blocks at the disk transfer rate.
- If blocks are small, there will be too many blocks in Hadoop HDFS and thus too much metadata to store. Managing such a huge number of blocks and metadata will create overhead and lead to traffic in a network.

HDFS's fsck command understands blocks. For example, running: **% hdfs fsck / -files -blocks**

will list the blocks that make up each file in the filesystem.

Name Node



The namenode manages the filesystem namespace. It maintains the filesystem tree and the metadata for all the files and directories in the tree. This information is stored persistently on the local disk in the form of two files: the namespace image and the edit log. The namenode also knows the datanodes on which all the blocks for a given file are located; however, it does

not store block locations persistently, because this information is reconstructed from datanodes when the system starts.

Functions of NameNode

- It is the master daemon that maintains and manages the DataNodes (slave nodes)
- It records the metadata of all the files stored in the cluster, e.g. The location of blocks stored, the size of the files, permissions, hierarchy, etc. There are two files associated with the metadata:
    - **FsImage:** It contains the complete state of the file system namespace since the start of the NameNode.
    - **EditLogs:** It contains all the recent modifications made to the file system with respect to the most recent FsImage.
- It records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog.
- It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live.
- It keeps a record of all the blocks in HDFS and in which nodes these blocks are located.
- The NameNode is also responsible to take care of the replication factor of all the blocks.
- In case of the DataNode failure, the NameNode chooses new DataNodes for new replicas, balance disk usage and manages the communication traffic to the DataNodes.

Data Node

DataNodes are the slave nodes in HDFS. Unlike NameNode, DataNode is a commodity hardware, that is, a non-expensive system which is not of high quality or high-availability. The DataNode is a block server that stores the data in the local file ext3 or ext4.

Functions of DataNode

- These are slave daemons or process which runs on each slave machine.
- The actual data is stored on DataNodes.
- The DataNodes perform the low-level read and write requests from the file system’s clients.
- They send heartbeats to the NameNode periodically to report the overall health of HDFS, by default, this frequency is set to 3 seconds.

Secondary NameNode

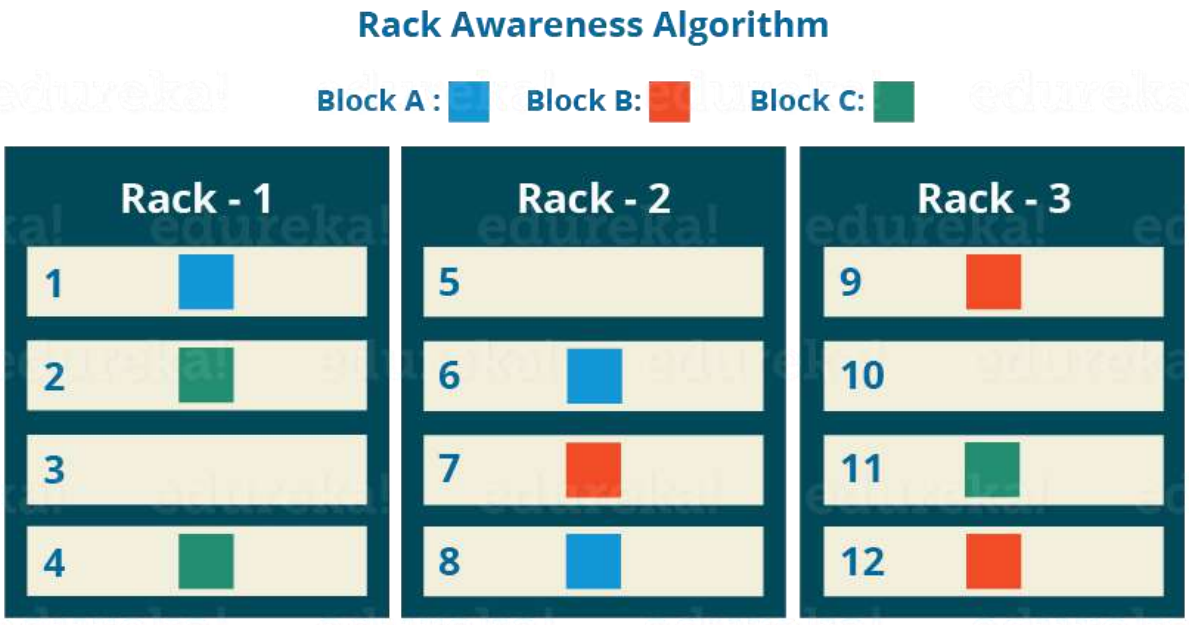
Apart from these two daemons, there is a third daemon or a process called Secondary NameNode. The Secondary NameNode works concurrently with the primary NameNode as a helper daemon. And don't be confused about the Secondary NameNode being a backup NameNode because it is not.

Functions of Secondary NameNode

- The Secondary NameNode is one which constantly reads all the file systems and metadata from the RAM of the NameNode and writes it into the hard disk or the file system.
- It is responsible for combining the EditLogs with FsImage from the NameNode.
- It downloads the EditLogs from the NameNode at regular intervals and applies to FsImage. The new FsImage is copied back to the NameNode, which is used whenever the NameNode is started the next time.

Hence, Secondary NameNode performs regular checkpoints in HDFS. Therefore, it is also called CheckpointNode.

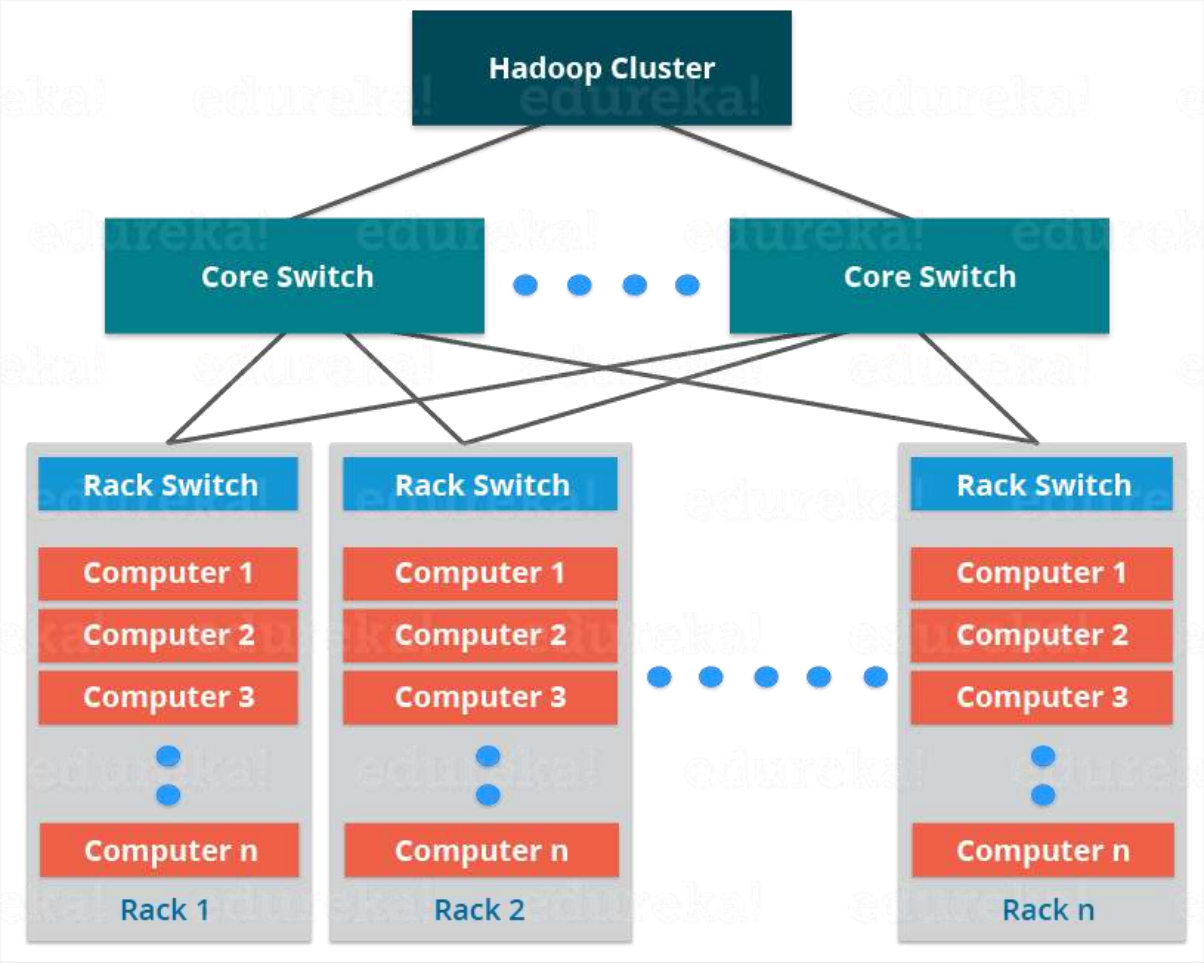
Rack Awareness:



The NameNode also ensures that all the replicas are not stored on the same rack or a single rack. It follows an in-built Rack Awareness Algorithm to reduce latency as well as provide fault tolerance. Considering the replication factor is 3, the Rack Awareness Algorithm says that the

first replica of a block will be stored on a local rack and the next two replicas will be stored on a different (remote) rack but, on a different DataNode within that (remote) rack as shown in the figure above. If you have more replicas, the rest of the replicas will be placed on random DataNodes provided not more than two replicas reside on the same rack, if possible.

This is how an actual Hadoop production cluster looks like. Here, you have multiple racks populated with DataNodes:



Block Caching

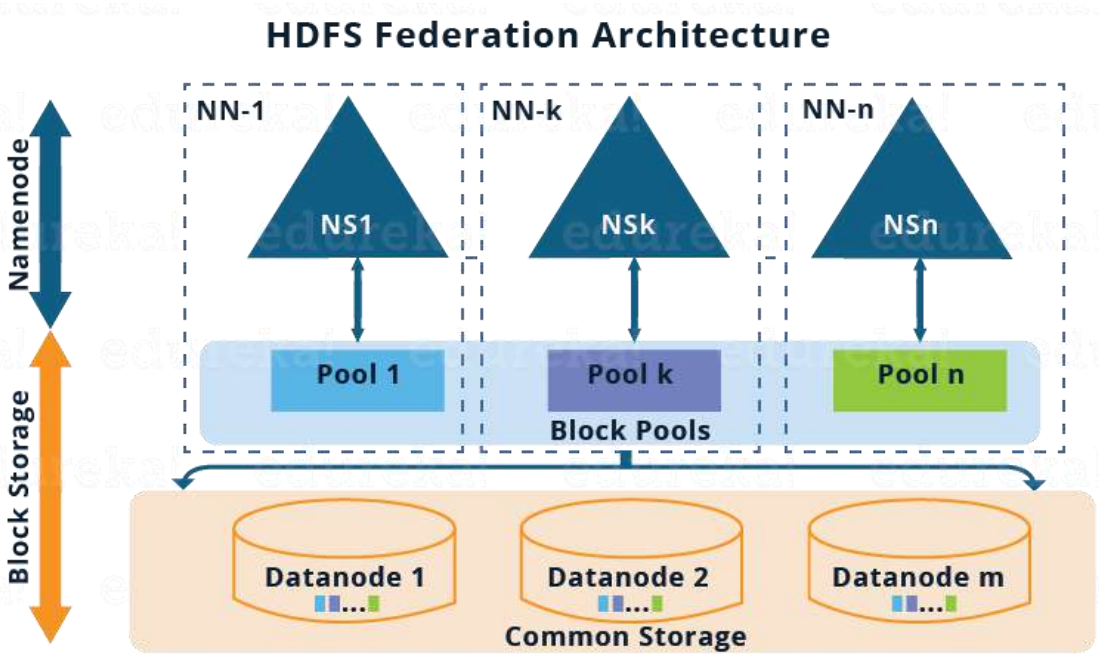
Normally a datanode reads blocks from disk, but for frequently accessed files the blocks may be explicitly cached in the datanode's memory, in an off-heap block cache. By default, a block is cached in only one datanode's memory, although the number is configurable on a per-file basis. Job schedulers (for MapReduce, Spark, and other frameworks) can take advantage of cached blocks by running tasks on the datanode where a block is cached, for increased read performance. A small lookup table used in a join is a good candidate for caching, for example.

Users or applications instruct the namenode which files to cache (and for how long) by adding a cache directive to a cache pool. Cache pools are an administrative grouping for managing cache permissions and resource usage.

HDFS Federation

The namenode keeps a reference to every file and block in the filesystem in memory, which means that on very large clusters with many files, memory becomes the limiting factor for scaling.

HDFS federation, introduced in the 2.x release series, allows a cluster to scale by adding NameNodes (NN), each of which manages a portion of the filesystem namespace. For example, one namenode might manage all the files rooted under /user, say, and a second namenode might handle files under /share.



Under federation, each namenode manages a namespace volume, which is made up of the metadata for the namespace, and a block pool containing all the blocks for the files in the namespace. Namespace volumes are independent of each other, which means namenodes do not communicate with one another, and furthermore the failure of one namenode does not affect the availability of the namespaces managed by other namenodes. Block pool storage is not partitioned, however, so datanodes register with each namenode in the cluster and store blocks from multiple block pools.

To access a federated HDFS cluster, clients use client-side mount tables to map file paths to namenodes. This is managed in configuration using ViewFileSystem and the

viewfs:// URIs.

HDFS High Availability (HA)

NameNode Availability:

In the standard configuration of HDFS cluster, the NameNode becomes a single point of failure. It happens because the moment the NameNode becomes unavailable, the whole cluster becomes unavailable until someone restarts the NameNode or brings a new one.

The reasons for unavailability of NameNode can be:

- A planned event like maintenance work such has upgradation of software or hardware.
- It may also be due to an unplanned event where the NameNode crashes because of some reasons.

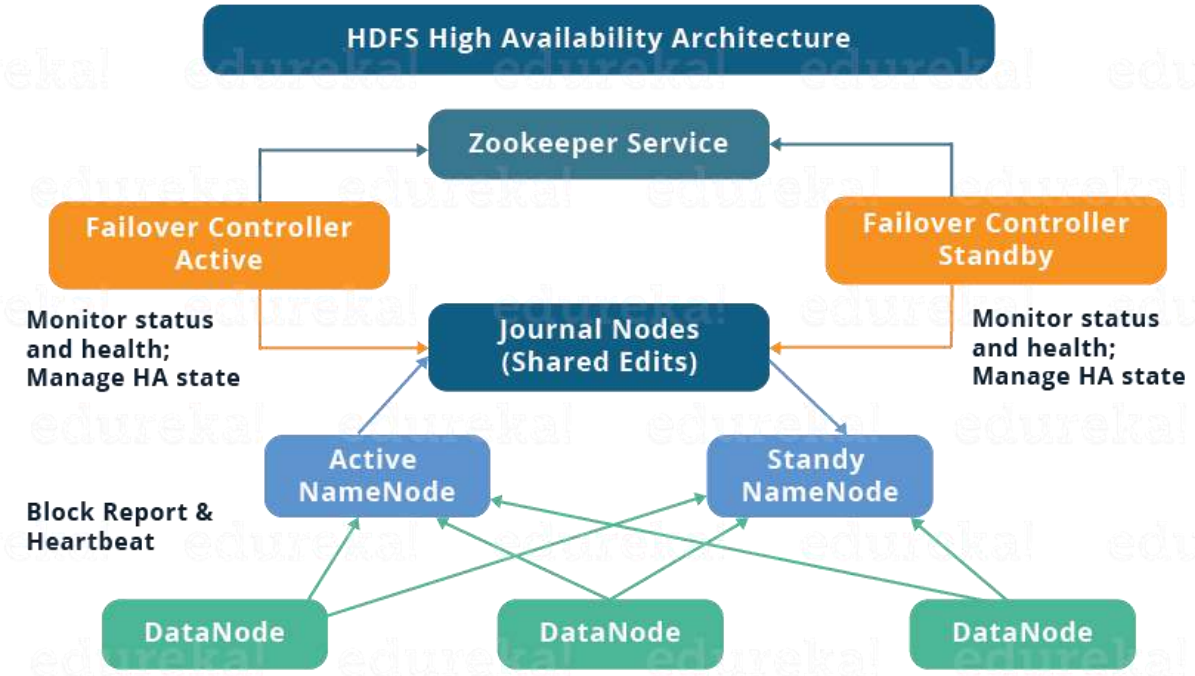
In either of the above cases, we have a downtime where we are not able to use the HDFS cluster which becomes a challenge.

HDFS HA Architecture:

The HA architecture solved this problem of NameNode availability by allowing us to have two NameNodes in an active/passive configuration. So, we have two running NameNodes at the same time in a High Availability cluster:

- **Active NameNode**
- **Standby/Passive NameNode.**

If one NameNode goes down, the other NameNode can take over the responsibility and therefore, reduce the cluster down time. The standby NameNode serves the purpose of a backup NameNode (unlike the Secondary NameNode) which incorporate failover capabilities to the Hadoop cluster. Therefore, with the StandbyNode, we can have automatic failover whenever a NameNode crashes (unplanned event) or we can have a graceful (manually initiated) failover during the maintenance period.



There are two issues in maintaining consistency in the HDFS High Availability cluster:

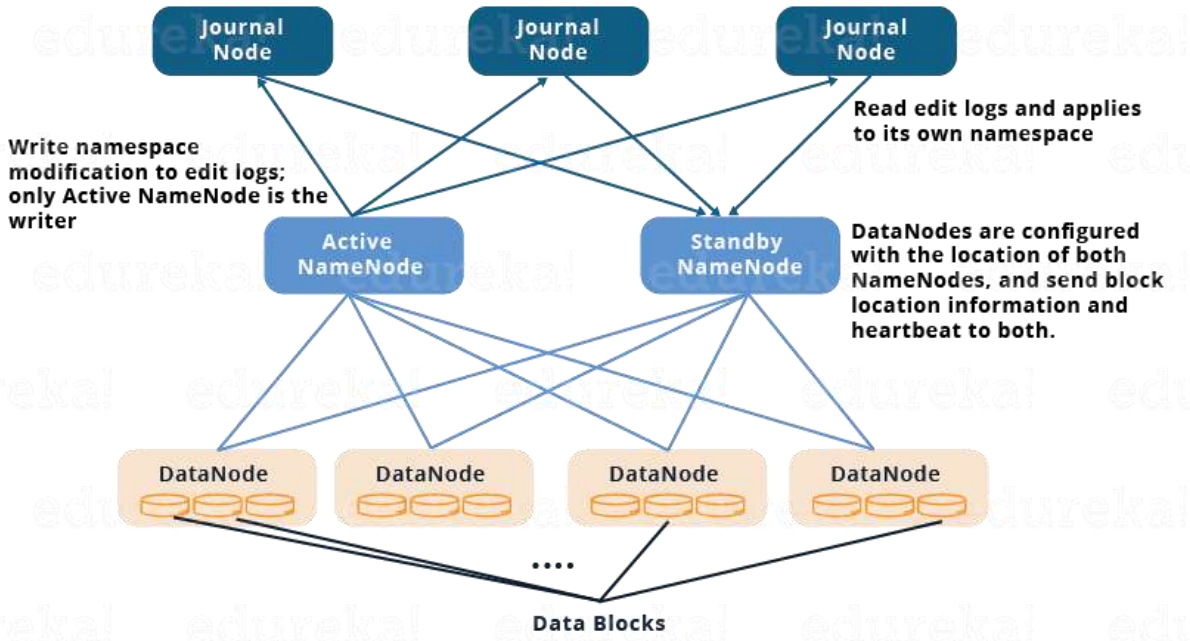
- Active and Standby NameNode should always be in sync with each other, i.e. They should have the same metadata. This will allow us to restore the Hadoop cluster to the same namespace state where it got crashed and therefore, will provide us to have fast failover.
- There should be only one active NameNode at a time because two active NameNode will lead to corruption of the data. This kind of scenario is termed as a split-brain scenario where a cluster gets divided into smaller cluster, each one believing that it is the only active cluster. To avoid such scenarios fencing is done. Fencing is a process of ensuring that only one NameNode remains active at a particular time.

Implementation of HA Architecture:

We can implement the Active and Standby NameNode configuration in following two ways:

1. Using Quorum Journal Nodes
2. Shared Storage using NFS

1. Using Quorum Journal Nodes:



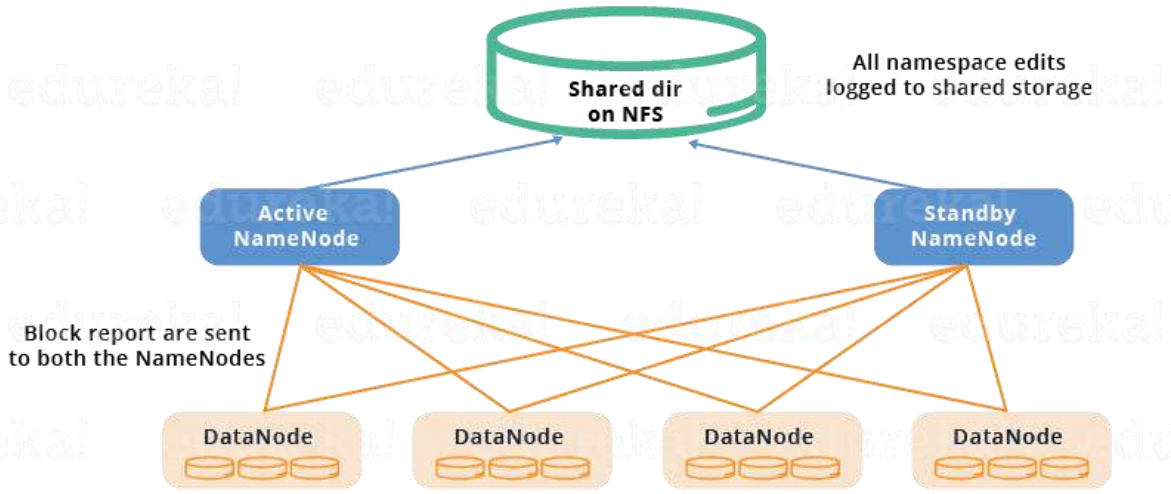
- The standby NameNode and the active NameNode keep in sync with each other through a separate group of nodes or daemons -called JournalNodes. The JournalNodes follows the ring topology where the nodes are connected to each other to form a ring. The JournalNode serves the request coming to it and copies the information into other nodes in the ring.This provides fault tolerance in case of JournalNode failure.
- The active NameNode is responsible for updating the EditLogs (metadata information) present in the JournalNodes.
- The StandbyNode reads the changes made to the EditLogs in the JournalNode and applies it to its own namespace in a constant manner.
- During failover, the StandbyNode makes sure that it has updated its meta data information from the JournalNodes before becoming the new Active NameNode. This makes the current namespace state synchronized with the state before failover.
- The IP Addresses of both the NameNodes are available to all the DataNodes and they send their heartbeats and block location information to both the NameNode. This provides a fast failover (less down time) as the StandbyNode has an updated information about the block location in the cluster.

Fencing of NameNode:

Now, as discussed earlier, it is very important to ensure that there is only one Active NameNode at a time. So, fencing is a process to ensure this very property in a cluster.

- The JournalNodes performs this fencing by allowing only one NameNode to be the writer at a time.
- The Standby NameNode takes over the responsibility of writing to the JournalNodes and forbid any other NameNode to remain active.
- Finally, the new Active NameNode can perform its activities safely.

2. Using Shared Storage:



- The StandbyNode and the active NameNode keep in sync with each other by using a shared storage device. The active NameNode logs the record of any modification done in its namespace to an EditLog present in this shared storage. The StandbyNode reads the changes made to the EditLogs in this shared storage and applies it to its own namespace.
- Now, in case of failover, the StandbyNode updates its metadata information using the EditLogs in the shared storage at first. Then, it takes the responsibility of the Active NameNode. This makes the current namespace state synchronized with the state before failover.
- The administrator must configure at least one fencing method to avoid a split-brain scenario.

- The system may employ a range of fencing mechanisms. It may include killing of the NameNode's process and revoking its access to the shared storage directory.
- As a last resort, we can fence the previously active NameNode with a technique known as STONITH, or "shoot the other node in the head". STONITH uses a specialized power distribution unit to forcibly power down the NameNode machine.

Automatic Failover:

Failover is a procedure by which a system automatically transfers control to secondary system when it detects a fault or failure. There are two types of failover:

- **Graceful Failover:** In this case, we manually initiate the failover for routine maintenance.
- **Automatic Failover:** In this case, the failover is initiated automatically in case of NameNode failure (unplanned event).

Apache Zookeeper is a service that provides the automatic failover capability in HDFS High Availabilty cluster. It maintains small amounts of coordination data, informs clients of changes in that data, and monitors clients for failures. Zookeeper maintains a session with the NameNodes. In case of failure, the session will expire and the Zookeeper will inform other NameNodes to initiate the failover process. In case of NameNode failure, other passive NameNode can take a lock in Zookeeper stating that it wants to become the next Active NameNode.

The ZookeerFailoverController (ZKFC) is a Zookeeper client that also monitors and manages the NameNode status. Each of the NameNode runs a ZKFC also. ZKFC is responsible for monitoring the health of the NameNodes periodically.

The Design of HDFS

HDFS is a filesystem designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware.

Very large files

"Very large" in this context means files that are hundreds of megabytes, gigabytes, or terabytes in size. There are Hadoop clusters running today that store petabytes of data.

Streaming data access

HDFS is built around the idea that the most efficient data processing pattern is a write-once, read-many-times pattern. A dataset is typically generated or copied from source, and then various analyses are performed on that dataset over time. Each analysis will involve a large proportion, if not all, of the dataset, so the time to read the whole dataset is more important than the latency in reading the first record.

Commodity hardware

Hadoop doesn't require expensive, highly reliable hardware. It's designed to run on clusters of commodity hardware (commonly available hardware that can be obtained from multiple vendors) for which the chance of node failure across the cluster is high, at least for large clusters. HDFS is designed to carry on working without a noticeable interruption to the user in the face of such failure.

It is also worth examining the applications for which using HDFS does not work so well. Although this may change in the future, these are areas where HDFS is not a good fit today:

Low-latency data access

Applications that require low-latency access to data, in the tens of milliseconds range, will not work well with HDFS. Remember, HDFS is optimized for delivering a high throughput of data, and this may be at the expense of latency.

Lots of small files

Because the namenode holds filesystem metadata in memory, the limit to the number of files in a filesystem is governed by the amount of memory on the namenode. As a rule of thumb, each file, directory, and block takes about 150 bytes. So, for example, if you had one million files, each taking one block, you would need at least 300 MB of memory. Although storing millions of files is feasible, billions is beyond the capability of current hardware.

Multiple writers, arbitrary file modifications

Files in HDFS may be written to by a single writer. Writes are always made at the end of the file, in append-only fashion. There is no support for multiple writers or for modifications at arbitrary offsets in the file.

HDFS Command Line Interface: Basic File System Operations

The Hadoop Command-Line Interface (CLI) provides a set of commands that allow users to interact with and manage Hadoop Distributed File System (HDFS) and run MapReduce jobs on a Hadoop cluster. Here are some common Hadoop CLI commands:

HDFS Commands:

Hadoop's Basic Filesystem Operations are performed through the Hadoop Distributed File System (HDFS), which is designed to store and manage large datasets across clusters of commodity hardware. Here are some of the fundamental filesystem operations you can perform in HDFS using the Hadoop CLI:

1) First of all Find out which commands are available in HDFS
   hduser@ip-172-31-55-84:~$ hdfs
2) Find out the Hadoop version which is installed
   hduser@ip-172-31-55-84:~$ hdfs version
3) To run a filesystem command on the file systems supported in Hadoop.
   hduser@ip-172-31-55-84:~$ hdfs dfs
4) HDFS Command to display the list of Files and Directories in HDFS.
   hduser@ip-172-31-55-84:~$ hdfs dfs -ls /
5) HDFS Command to create the directory in HDFS.
   hduser@ip-172-31-55-84:~$ hdfs dfs -mkdir /vishal
6) Now create some files in your local machine
   hduser@ip-172-31-55-84:~$ vim input

   hduser@ip-172-31-55-84:~$ vim input1

   hduser@ip-172-31-55-84:~$ vim input2
7) Now Move the File from local to hdfs directory

   hduser@ip-172-31-55-84:~$ hdfs dfs -moveFromLocal /home/hduser/input /vishal
8) Now Copy a file from local to HDFS directory

   hduser@ip-172-31-55-84:~$ hdfs dfs -copyFromLocal /home/hduser/input1 /vishal
9) Now Copy a file from HDFS to Local Directory

   hduser@ip-172-31-55-84:~$ hdfs dfs -copyToLocal /vishal/input /home/hduser/

10) Now create a new directory in HDFS

hduser@ip-172-31-55-84:~$ hdfs dfs -mkdir /reddy

11) Now move the file from One directory of HDFS to other Directory

hduser@ip-172-31-55-84:~$ hdfs dfs -mv /vishal/input /reddy

12) HDFS Command that reads a file on HDFS and prints the content of that file to the standard output.

hduser@ip-172-31-55-84:~$ hdfs dfs -cat /vishal/input1

13) HDFS Command to create a file in HDFS with file size 0 bytes.

hduser@ip-172-31-55-84:~$ hdfs dfs -touchz /vishal/input3

14) To copy a file from one directory of HDFS to Another Directory of HDFS

hduser@ip-172-31-55-84:~$ hdfs dfs -cp /vishal/input3 /reddy/

15) To remove a file from HDFS directory

hduser@ip-172-31-55-84:~$ hdfs dfs -rm /reddy/input3

16) To put a file local to HDFS directory

hduser@ip-172-31-55-84:~$ hdfs dfs -put /home/hduser/input2 /reddy

17) To get file level ACL permissions

hduser@ip-172-31-55-84:~$ hdfs dfs -getfacl /reddy

file: /reddy

owner: hduser

group: supergroup

18) To get number of directories and files

hduser@ip-172-31-55-84:~$ hdfs dfs -count /

19) To check the health of HDFS

hduser@ip-172-31-55-84:~$ hdfs fsck /

20) To remove a directory

hduser@ip-172-31-55-84:~$ hdfs dfs -rm -r /vishal

21) To Check the status of a file

hduser@ip-172-31-55-84:~$ hdfs dfs -stat "*%F %u:%g %b %y %n" /reddy/input

22) To check the size of a file

hduser@ip-172-31-55-84:~$ hdfs dfs -du /reddy/input

23) HDFS Command to copy files from hdfs to the local file system.

hduser@ip-172-31-55-84:~$ hdfs dfs -get /reddy/input /home/hduser/

24) To change the permission of a directory and a file

hdfs dfs -chmod 777 /reddy

25) To change the ownership of a directory or a file

hduser@ip-172-31-55-84:~$ hdfs dfs -chown hduser:hadoop /reddy

MapReduce Commands:

hadoop jar <jar_file> <main_class> <input_path> <output_path>: Run a MapReduce job using the specified JAR file, main class, input path, and output path.

hadoop job -list: List all running MapReduce jobs.

hadoop job -kill <job_id>: Kill a running MapReduce job by its job ID.

hadoop job -history <job_id>: Display the history of a completed MapReduce job.

Other Commands:

hadoop version: Display the Hadoop version information.

hadoop dfsadmin -report: Display a report on the HDFS cluster's overall status.

hadoop fsck <hdfs_path>: Check the health of files and directories in HDFS.

hadoop dfsadmin -safemode [enter | leave | get | wait]: Manage the HDFS safe mode.

Hadoop File Systems: The Java Interface

Hadoop File Systems

Hadoop has an abstract notion of filesystems, of which HDFS is just one implementation. The Java abstract class org.apache.hadoop.fs.FileSystem represents the client interface to a filesystem in Hadoop, and there are several concrete implementations.

The main ones that ship with Hadoop are described in below Table.

Filesystem	URI scheme	Java implementation (all under org.apache.hadoop)
Local	file	fs.LocalFileSystem
HDFS	hdfs	hdfs.DistributedFileSystem
WebHDFS	webhdfs	hdfs.web.WebHdfsFileSystem
Secure WebHDFS	swebhdfs	hdfs.web.SWebHdfsFileSystem
HAR	har	fs.HarFileSystem
View	viewfs	viewfs.ViewFileSystem
FTP	ftp	fs.ftp.FTPFileSystem
S3	s3a	fs.s3a.S3AFileSystem

Hadoop provides many interfaces to its filesystems, and it generally uses the URI scheme to pick the correct filesystem instance to communicate with. For example, the filesystem shell that we met in the previous section operates with all Hadoop filesystems. To list the files in the root directory of the local filesystem, type:

% hadoop fs -ls file:///

The Java Interface

Reading Data from a Hadoop URL

One of the simplest ways to read a file from a Hadoop filesystem is by using a **java.net.URL object to open a stream to read the data from.**

There's a little bit more work required to make Java recognize Hadoop's hdfs URL scheme. **This is achieved by calling the setURLStreamHandlerFactory() method on URL with an instance of FsUrlStreamHandlerFactory.** This method can be called only once per JVM, so it is typically executed in a static block. This limitation means that if some other part of your program perhaps a third-party component outside your control sets a URLStreamHandlerFactory, you won't be able to use this approach for reading data from Hadoop.

Example 1: Displaying files from a Hadoop filesystem on standard output using a URLStreamHandler

```
public class URLCat {
static {
URL.setURLStreamHandlerFactory(new FsUrlStreamHandlerFactory());
}
public static void main(String[] args) throws Exception {
InputStream in = null;
try {
in = new URL(args[0]).openStream();
IOUtils.copyBytes(in, System.out, 4096, false);
} finally {
IOUtils.closeStream(in);
}
}
}
```

We make use of the handy IOUtils class that comes with Hadoop for closing the stream in the finally clause, and also for copying bytes between the input stream and the output stream (System.out, in this case). The last two arguments to the copyBytes() method are the buffer size used for copying and whether to close the streams when the copy is complete. We close the input stream ourselves, and System.out doesn't need to be closed.

Reading Data Using the FileSystem API

A file in a Hadoop filesystem is represented by a Hadoop Path object.

You can think of a Path as a Hadoop filesystem URI, such as hdfs://localhost/user/quangle.txt.

FileSystem is a general filesystem API, so the first step is to retrieve an instance for the filesystem we want to use HDFS, in this case. There are several static factory methods for getting a FileSystem instance:

public static FileSystem get(Configuration conf) throws IOException
public static FileSystem get(URI uri, Configuration conf) throws IOException
public static FileSystem get(URI uri, Configuration conf, String user) throws IOException

A Configuration object encapsulates a client or server's configuration, which is set using configuration files read from the classpath, such as etc/hadoop/core-site.xml.

- The first method returns the default filesystem (as specified in core-site.xml, or the default local filesystem if not specified there).
- The second uses the given URI's scheme and authority to determine the filesystem to use, falling back to the default filesystem if no scheme is specified in the given URI.
- The third retrieves the filesystem as the given user, which is important in the context of security.

In some cases, you may want to retrieve a local filesystem instance. For this, you can use the convenience method getLocal():

public static LocalFileSystem getLocal(Configuration conf) throws IOException

With a FileSystem instance in hand, we invoke an open() method to get the input stream for a file:

public FSDataInputStream open(Path f) throws IOException

public abstract FSDataInputStream open(Path f, int bufferSize) throws IOException

Example 2. Displaying files from a Hadoop filesystem on standard output by using the FileSystem directly

```
public class FileSystemCat {
public static void main(String[] args) throws Exception {
String uri = args[0];
Configuration conf = new Configuration();
FileSystem fs = FileSystem.get(URI.create(uri), conf);
InputStream in = null;
try {
in = fs.open(new Path(uri));
IOUtils.copyBytes(in, System.out, 4096, false);
} finally {
IOUtils.closeStream(in);
}
}
}
```

Writing Data

The FileSystem class has a number of methods for creating a file. The simplest is the method that takes a Path object for the file to be created and returns an output stream to write to:

public FSDataOutputStream create(Path f) throws IOException

There are overloaded versions of this method that allow you to specify whether to forcibly overwrite existing files, the replication factor of the file, the buffer size to use when writing the file, the block size for the file, and file permissions.

There's also an overloaded method for passing a callback interface, Progressable, so your application can be notified of the progress of the data being written to the datanodes:

```
package org.apache.hadoop.util;
public interface Progressable {
public void progress();
}
```

As an alternative to creating a new file, you can append to an existing file using the append() method (there are also some other overloaded versions):

public FSDataOutputStream append(Path f) throws IOException

The append operation allows a single writer to modify an already written file by opening it and writing data from the final offset in the file. With this API, applications that produce unbounded files, such as logfiles, can write to an existing file after having closed it. The append operation is optional and not implemented by all Hadoop filesystems. For example, HDFS supports append, but S3 filesystems don't.

Example 3. Copying a local file to a Hadoop filesystem

```
public class FileCopyWithProgress {
public static void main(String[] args) throws Exception {
String localSrc = args[0];
String dst = args[1];
InputStream in = new BufferedInputStream(new FileInputStream(localSrc));
Configuration conf = new Configuration();
FileSystem fs = FileSystem.get(URI.create(dst), conf);
OutputStream out = fs.create(new Path(dst), new Progressable() {
public void progress() {
System.out.print(".");
}
});
IOUtils.copyBytes(in, out, 4096, true);
}
}
```

Directories

FileSystem provides a method to create a directory:

public boolean mkdirs(Path f) throws IOException

This method creates all of the necessary parent directories if they don't already exist, just like the java.io.File's mkdirs() method. It returns true if the directory (and all parent directories) was (were) successfully created. Often, you don't need to explicitly create a directory, because writing a file by calling create() will automatically create any parent directories.

Deleting Data

Use the delete() method on FileSystem to permanently remove files or directories:

public boolean delete(Path f, boolean recursive) throws IOException

If f is a file or an empty directory, the value of recursive is ignored. A nonempty directory is deleted, along with its contents, only if recursive is true (otherwise, an IOException is thrown).

JAVA API INTERFACE WITH HDFS for LS, CREATE A DIR, COPY A FILE, CAT

```
import java.io.IOException;
import java.io.InputStream;
import java.net.URI;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;
public class FR {
public static void main (String args[]) throws IOException{

String uri = args[0];
Configuration conf = new Configuration();
FileSystem fs = FileSystem.get(URI.create(uri), conf);

// how to do ls command

// FileStatus[] files = fs.listStatus(new Path(uri));
// for(FileStatus file : files){
// System.out.println(file.getPath().getName());
// }


// How to create a directory mkdir
// boolean yesorno = fs.mkdirs(new Path(uri));
// System.out.println("The status of the dir is : "+yesorno);

// CopyFromLocal Operation
String src = args[1];
fs.copyFromLocalFile(new Path(src),
new Path(uri));

// how to cat a file
// InputStream in = null;
// try{
// in = fs.open(new Path(uri));
// IOUtils.copyBytes(in, System.out, 4096,false);
// }
// finally{
// IOUtils.closeStream(in);
// }
}
}
```