Big Data (Chapter 1)

Introduction to Big Data

- **What is Data?**
  Quantities, characters or symbols on which operations are performed by a computer.
- **What is Big Data?**
  Data but with a huge size. Huge in volume and growing exponentially with time.

Memory Sizes

Name	Equal To	Size (In Bytes)
Bit	1 Bit	1/8
Nibble	4 Bits	1/2 (rare)
Byte	8 Bits	1
Kilobyte	1024 Bytes	
Megabyte	1024 Kilobytes	
Gigabyte	1024 Megabytes	
Terrabyte	1024 Gigabytes	
Petabyte	1024 Terrabytes	
Exabyte	1024 Petabytes	
Zettabyte	1024 Exabytes	
Yottabyte	1024 Zettabytes	

Big Data Characteristics

1. **Volume**: Very Large Amount of Data
2. **Velocity**: Produce Data at very Fast Rate
3. **Variety**: Produce Data in Different formats
4. **Veracity**: The Correctness of Data.
5. Value

Types of Digital Data

- **Structured:**
  → Data that can be stored, accessed and processed in the form of fixed format.
- **Unstructured:**
  → Data with unknown form or structure
  → Ex: Heterogeneous data source containing a combination of simple text files, images, videos, ...
- **Semi-Structured:** structured in form but it is actually not defined
  → Contain both the forms of data
  → Ex: XML File

Big Data Analytics

- Process of collecting, organizing and analyzing large sets of data to discover patterns and useful information.
- Analysts want the knowledge that comes from analyzing the data.
- **High Performance Analytics Required:**
  → Big Data Analytics is performed using specialized software tools and applications for:
    - Predictive Analytics
    - Data Mining
    - Text Mining
    - Forecasting
    - Data Optimization
- **Challenges:**
  → Breakdown data silos to access all data an organization stores in different places / systems.

→ Creating platforms that can pull in unstructured data

- Benefits:

→ An enterprise can boost sales, increase efficiency, improve operations, customer service and risk management

Application of Big Data

- Retail
- Healthcare
- Education
- E-commerce
- Media and Entertainment
- Finance
- Travel Industry
- Telecom
- Automobile

- Financial firms were among the earliest adopters of Big Data and Analytics.

- Digital banking and payments are two of the most trending buzzwords around, and Big data has been at the heart of it. Big Data is bossing the key areas of financial firms such as fraud detection, risk analysis, algorithmic trading, and customer contentment.

- This has brought much-needed fluency in their systems. They are now empowered to focus more on providing better services to their customers rather than focusing on security issues.
Big Data has now enhanced the financial system with answers to its hardest of the challenges

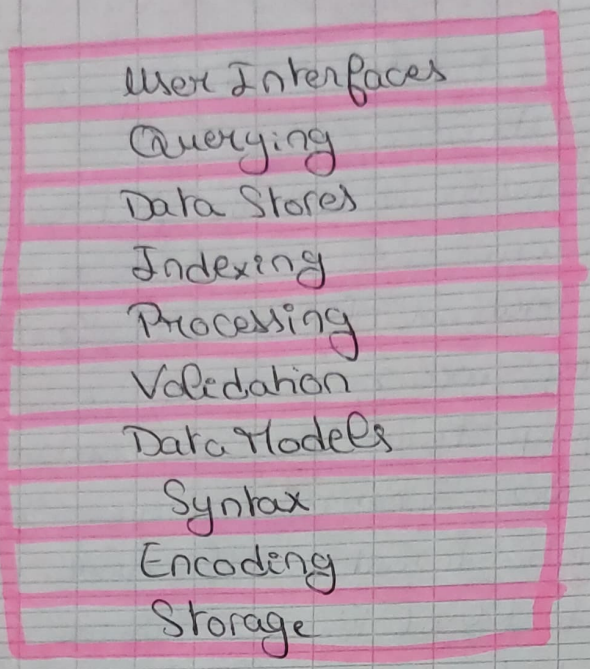
Big Data (Chapter 2)

Cloud Storage

Storing Data

- 99% of what we learn in the past 50+ years can be directly reused:
  - → The Relational Algebra: selection, projection, grouping, sorting, joining.
  - → Language Features: SQL, declarative languages, functional languages, optimizations, query plans, indices.
  - → The Table layout: tables, rows, columns, primary keys
- Indeed, in NOSQL these concepts are replaced with:
  - → Nested data: data that is not in first normal form
  - → Heterogeneous data: data that does not fulfil domain integrity (it may or may not have a schema) and also not relational integrity.
- At large scales, it becomes desirable to sometimes de-normalize data for better performance.
- ACID → CAP
  - → Atomicity → → Consistency (Atomic)
  - → Consistency → → Availability
  - → Isolation → → Partition tolerance
  - → Durability

The Technology Stack



- **Storage:**
  - → Local file system on your laptop
  - → Network file systems (NFS)
  - → Distributed file systems (HDFS, GFS)
  - → Hosted and fully managed cloud storage (S3, Azure Blob Storage)
- **Encoding:**
  - → Convert data to bits.
  - → ASCII → ISO_8859_1 → Unicode (UTF_8)
  - → Binary encodings (BSON)
- **Syntax:**
  - → Is used when data is stored as text (human_readable)
  - → CSV (for tables), XML and JSON (for trees), RDF/XML and Turtle (for graphs) and XBRL (for trees).
  - → Binary forms (BSON, Parquet, Protocol Buffers).
- **Data Models:**
  - → Tables (Relational Model)
  - → Trees (XML infoset, JSONiq data model)

→ Graphs (RDF triples and labeled property graphs)
→ Cubes (OLAP)

- Validation:
  → Languages to add constraints to data.
  → XML Schema, JSON Schema, JSound, Relational Schemas (in relational database systems), XBRL taxonomies.
- Processing:
  → Tez, Apache Spark, Apache Flink, Ray
  → Lower level frameworks at the level of VM (Amazon EC2)
- Indices:
  → Accelerate data lookups
- Data Stores:
  → Relational Database Management System (RDBMS)
  → Document Stores (MongoDB, CouchBase, Elastic Search)
  → Data Processing (Hive, HBase, Databricks, Cassandra)
  → Snowflake, MarkLogic.
- Querying:
  → Higher-level languages are supported by some but not all data stores.
  → SQL is the undisputed choice for tables
  → For trees (JSONiq, XQuery, N1QL, PartiQL)
  → For graphs (Cypher, SPARQL)
  → REST APIs provide a lower-level access to a data store
- User Interface:
  → Spreadsheet software, BI tools (Tableau, QlikView, Access).
  → Voice Assistants (Alexa, Siri)

Databases vs. DataLakes

Database

- Data can be imported into the database (ETL, Extract, Transform, Load). This causes more work for the user, as they need to import all the data to query it. (faster queries)

Datalake

- Data can be stored on some file system (local / distributed / cloud) and queried in place (in situ) by a data processing engine either with an API (Pandas, Apache Spark) or query language (Rombe DB). (Slower Queries)

Local file Systems

- On a single machine, the data is stored on the local file system.
- Newer generation of disks called SSD (Solid State Disk) more expensive and smaller, 1TB (ex)
- When a file is stored on local filesystem, it consists of both content and metadata.
- Files are read blocks by blocks (4KB)
- Metadata:
  - → Name of the file
  - → Access Rights
  - → Owner
  - → Group of the owner
  - → Creation time
  - → last modification time
  - → Size

More Users, more files:

- A disk can be made accessible via the network (LAN, Local Area Network) and shared with other people.
- WAN (Wide Area Network)
- In order to make this scale, the approach for large scales, namely here, object storage, is to:
  - → Throw away the hierarchy (no directories)
  - → Make the metadata flexible (no schema)
  - → Simple data model: flat list of files (called objects) identified with an identifier (ID)
  - → Use large number of cheap machines > super-computer.

Scale Up vs. Scale Out

- Scale Up:
  - → Buy a bigger machine: more memory, more or faster CPU cores, larger disk.
  - → So expensive and impractical
- Scale Out:
  - → Buy more, similar machines and share the work across them.
  - → Scale out is worth it in 2 cases:
    - the data does not fit on your laptop
    - your laptop cannot read the data fast enough.

Data Centers

- In a data center, there are thousands to tens of thousands of machines.
- A server is called a node in the context of data center.
  (1 → 64 cores) (16GB → 6TB memory)
  Each node

- Local Storage Available (SSD or HDD): 1 → 20TB
- Network Bandwidth: 1 → 100 Gb/s.
- Nodes are typically flat, rectangular boxes that are piled up in what is called a racks.
- Cluster: room filled with racks.
- Each module in a racks can be:
  - → a server
  - → pure storage (many disks)
  - → network switch

  (1 → 4 RU, Rack Units)
- Cloud Providers:
  - → Amazon Web Services (AWS)
  - → MicroSoft Azure
  - → Google Cloud

Amazon S3

- Amazon's object storage system is called Simple Storage Services (S3).
- Objects are organized in buckets, buckets are identified with a Bucket ID, and each object within a bucket is identified with an Object ID.
- Object ≡ File (text file, picture, video, dataset ...)
- Object Stores has no hierarchy.
- An object can be at most 5TB.
- A user can have up to 100 buckets (+ more on request)

Azure Blob Storage

- It differs from S3 in several ways:

→ Its architecture is documented (online) publicly => easier to understand
→ Objects are identified with 3, rather than 2 IDs:
  - An account
  - A container
  - A blob
→ It exposes more details to the user (objects are divided in several blocks)
→ It differentiates between Block Blobs, Append Blobs (for logging) and Page Blobs (for storing and accessing the memory of VM)
→ Sizes:
  - 195 GB for an Append Blob
  - 190.7 TB for a Block Blob.

- Azure Blob Storage is organized in storage stamps located in various data centers worldwide.
  → Each storage stamp consist of 10 to 20 racks, with each rack containing around 18 storage nodes (disks + servers)
  → Storage stamp can store up to 30 PB of data
  → 80% total capacity

Service Level Agreements (SLA)

- Most cloud services come with a cost, but also a promise. This promise is a contract between the provider and the user called a Service Level Agreement (SLA).

Th. CAP Theorem

- CAP theorem is an impossibility triangle, a system

cannot guarantee at the same time:

→ Consistency (Atomic): All nodes see the same data

→ Availability: the system is available for requests at all times

→ Partition Tolerance: the system continues to function even if the network linking its machines is occasionally partitioned

- If there is a partition of the network leading to 2 disconnected sub-networks and the propagation of some data did not complete, there are 2 possible design decisions:
  - → System is temporarily put to a halt and becomes unavailable until the network partition is resolved. (Consistent but not Available)
  - → The servers continue to serve requests, but the data served by 2 disconnected sub-networks is different (Available but not Consistent)
- A third way is for the system not to be partition tolerant; the system is available and consistent but only for as long as no network partition occures.

Rest APIs

- Data Stores expose their functionality via an API, mostly REST API.
- REST = REpresentational State Transfer (HTTP done right)
- REST API makes an integration with any host language very easy, because HTTP clients exist in almost any host language (Java, Python, PHP, R...).
- Other possibilities are a native driver (MongoDB).

- Another protocol is XML-based SOAP protocol.
- A client and server communicated with the HTTP protocol interact in terms of methods applied to resources.
- A resource can be anything (document, PDF, person, physical object, calendar entry), and is referred to with URI (Uniform Resource Identifier)
- http://www.ex.com/api/collection/foobar?id=foobar#head

scheme ↙ authority ↓ path ↓ query ↓ fragment

- A client can act on resources by invoking methods:
  - → GET (without a body): return a representation of the resource in some format (text, XML, JSON...). No side effects.
  - → PUT: creates or updates a resource from a representation of a newer version of it, in some format. PUT has a side effect that a subsequent GET asking for the same format should return the same representation.
  - → DELETE (without a body): deletes a resource. ~~No side effects~~ Side effect that a subsequent GET asks for representation of the resource with a not-found error (404).
  - → POST: blanket choice, it acts on resource in a way the data store seems fit. A typical use of POST is to create a new resource by letting the REST server pick a resource URI for this new resource.

Object Stores in practice

- Although object stores are based on a flat key-value model, most object storage services emulate a file hierarchy by allowing slash (/) characters in keys as virtual paths.

Static Website Hosting

- Object stores such as S3 and Azure Blob Storage can be used for static website hosting.
- Static website hosting means that there is no dynamically generated content (PHP, JavaEE).
- HTML pages, picture, CSS stylesheets, JavaScript scripts are delivered as is.
- CDN (Content Delivery Network) to accelerate and cache these files at multiple places on the planet.

Key-value Stores

- While object stores are able to store large objects, their latency (100 milliseconds or +) is higher than typical database systems (1 → 9 ms).
- It exists a paradigm that has a similar key-value model (get/put/delete API) with low latency but smaller objects and no metadata.
- A key-value store differs from a typical relational database:
  - → Its API is simpler
  - → It does not ensure atomic consistency, instead it guarantees eventual consistency.
  - → Its very fast also at large scales.

Key-value store API

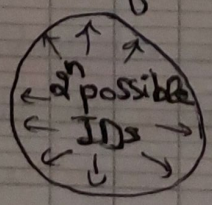
- A key-value store has three access methods:
  - → get(key)
  - → put(key, value)
  - → delete(key)

Design Principles

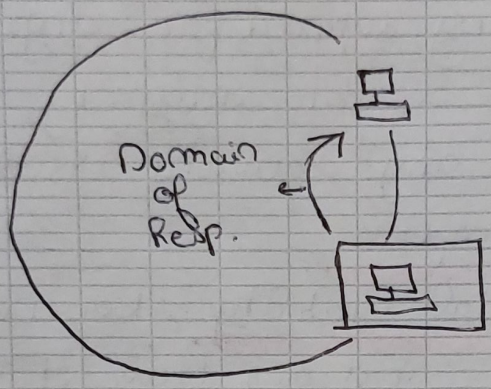
- Distributed Hash Tables are generally:
  - → Highly scalable
  - → Robust against failures
  - → Self organizing
- However, they do not:
  - → provide any support for range queries
  - → Guarantees on data integrity
  - → Deal with any security issues.
- Design Principle:
  - → Incremental Stability: New nodes can join the system at any time, and can leave it.
  - → Symmetry: No node is particular in any way
  - → Decentralization: No central node that orchestrates the others.
  - → Heterogeneity: Nodes may have different CPU power, amounts of memory, ...

A ring to rule them all

- That are several ways that nodes can be connected together:
  - → Peer-to-Peer (every node speaks to every other node)
  - → Centralized (Coordinator Node that controls Worker Nodes)
- Distributed Hash Tables => Peer-to-Peer.
- Every logical key is hashed to bits that we'll call IDs (Dynamo => hash is made of 128 bits ; 7 bytes)



All nodes (machines in the data center) get a random position on that ring (random 128-bit number)
The domain of responsibility of a single node n is the ring interval separating it from the next node counterclockwise (all keys whose hash lands in that interval are stored on and the responsibility of node n).



- New Node n joins the cluster:
  - → It gets assigned a position on the ring in the domain of responsibility of some node m
  - → The ring interval is going to be split (everything between m and n remains the resp. of m, but the other half is the resp. of n).
- Gracefully Exit of n:
  - → n transfer all its interval of resp. to the next node clockwise (m)
- Abrupt Exit of n:
  - → Data for that node is gone
  - → This problem is solved with data replication (data within an interval will not be stored only on the node that follow the interval clockwise but on the next N nodes that follow the interval clockwise).

Preference Lists

- In the Chord protocol, a technology called "finger table" is used. Each node knows the next node clockwise, and the next, and 4th, and the 8th ... (powers of two). So a binary search is used to find the desired node.
- Dynamo changes the design to "preference lists": each node knows, for every key, which node(s) are responsible (and hold a copy of it) of it.
- The size of every list must be at least N.
- It can get (temporarily) larger than N if there are network partitions or nodes leaving and taking over.
- The first node of this list is called "Coordinator".

An Improvement: Tokens and Virtual Nodes

- Virtualize the nodes: instead of directly mapping each node to one position on the ring, a larger quantity of (virtual) nodes, called tokens, are partitioned on the ring.

Ex: - 100 nodes → 1000 tokens on the ring (assigned to the nodes)

Partition Tolerance and Vector Clocks

- Distributed hash tables are typically AP (accept inconsistencies, downgrading the guarantee to eventual consistency).
- A fundamental tool in AP systems is the use of vector clocks.
- System with No Partitions:
  - → Versions of data are kept.
  - → Identified with integers.

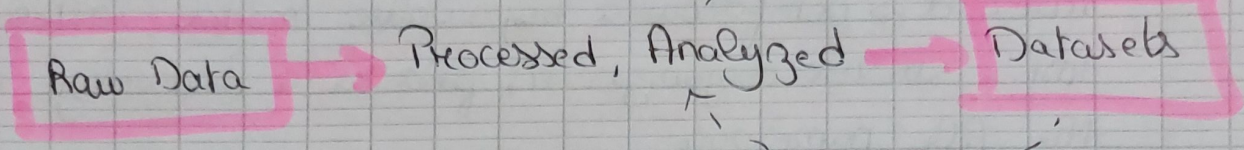
- Network Partitions :
→ DAG (Directed Acyclic Graph)

Big Data (Chapter 3)

Distributed File Systems

Main Requirements of a Distributed File System

- Data goes through a lifecycle:



- The storage technology must be capable of:
  - → Monitoring itself
  - → Detecting failures
  - → Automatically recovering
  - → Being Fault Tolerant.
- For distributed data storage though, (raw data ---> datasets) we need to be able to:
  - → Efficiently scan a large file (in its entirety) - for data analysis
  - → Append efficiently new data at the end of an existing large file - particularly for logging and sensors.
- Going back to our capacity-throughput-latency view of storage, a DFS is designed so that, in cruise mode, its bottleneck will be the data flow (throughput), not the latency.
- Capacity ↑ much faster than Throughput ⇒ This can be solved with parallelism
- Throughput ↑ much faster than Latency ↓ ⇒ This can be solved with batch processing.
- DFS → Parallelism
  - → Batch Processing

The model behind HDFS

File Hierarchy

- In HDFS, the word "file" is used instead of "object".
- HDFS does not follow a key-value model, instead an HDFS cluster organizes its files as a hierarchy, called the file namespace.
- Files are organized in directories, similar to local file system.

Blocks

- Unlike in S3, HDFS files are not stored as monolithic blocks boxes, but HDFS exposes them as list of blocks (like local file systems)
- In Google's file System (GFS), blocks are called chunks.
- Blocks, Chunks, Splits, Shards, Partitions ...

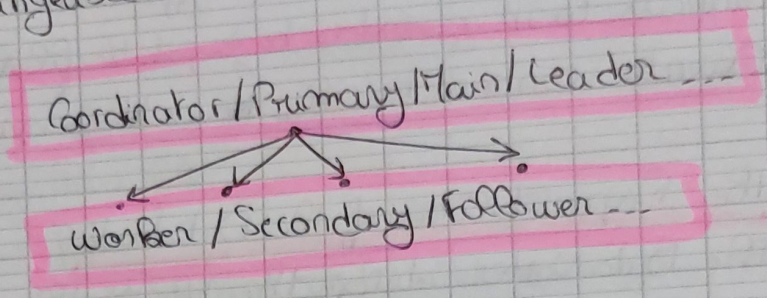
The Size of the Blocks

- HDFS blocks are typically 64 MB or 128 MB large
- Blocks of 128 MB are large enough that time is not lost in latency waiting for a block to arrive, and small enough for a large file to be conveniently spread over many machines, allowing for parallel access.

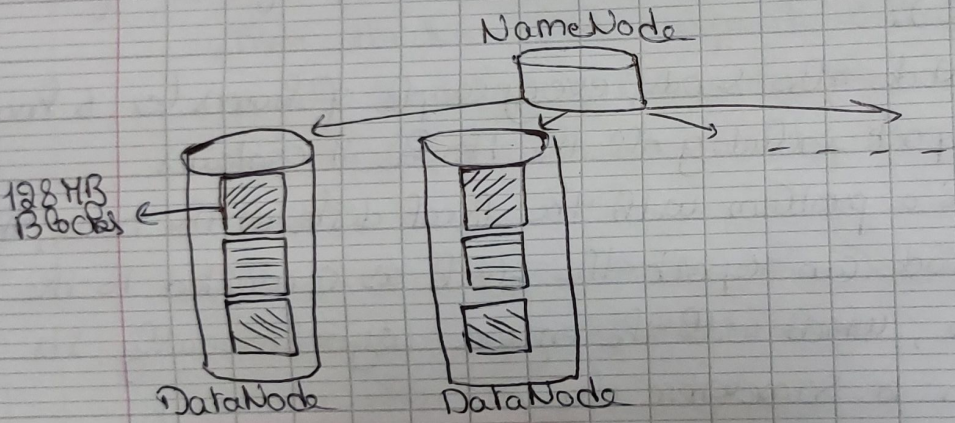
Physical Architecture

- One way to connect machines is "Peer-to-Peer", decentralized network, where each machine talks to any other machine.
- But, HDFS is implemented on a fully centralized architecture, where one node is special and all others

are interchangeable and connected to it.



- HDFS :
  - → Central Node : NameNode
  - → Others : DataNodes



- Each block is replicated and stored on several DataNodes.
- The Replication Factor by default is 3.

The Responsibilities of the NameNode

- It stores three things :
  - → The file namespace : the hierarchy of directory names and file names, and any ACL (Access Control List) info.
  - → A mapping from each file to the list of its blocks : each block, in this list, is represented with 64-bit identifier
  - → A mapping from each block, represented with its 64-bit identifier to the locations of its replicas (list of DataNodes)

- The NameNode updates this information whenever a client connects to it in order to update files and directories.
- Clients connect to the NameNode via the Client Protocol.

The Responsibilities of the DataNode

- The DataNodes stores the blocks themselves.
- If the last blocks of an HDFS file is less than 128MB, then the physical file has exactly the size of the blocks.
- DataNodes send regular heartbeats to the NameNode. The frequency of these heartbeats is by default a few seconds (ex. 3 sec).
- The DataNode also sends, every couple of hours (ex. 6 hours), a full report including all the blocks it contains.
- If there's a problem with the local disk on the node, then the DataNode can report the blocks as corrupted to the NameNode, which will the ensure, asynchronously, its replication to somewhere else.
- A NameNode never initiates a connection to the DataNode
- DataNodes are capable of communicating with each other by forming replication pipelines.
- when a replication pipeline is ongoing and a new block is being written, the content of the block is sent in small packets (64KB ex) and not a 128MB packet.

File System Functionality

- HDFS exposes to the client, via the NameNode, an API that allows typical operations available on a file system:
  → Creating a directory

→ Deleting a directory
→ Listing the contents of a directory
→ Creating a file
→ Appending to a file
→ Deleting a file.

- However, reading, writing and deleting a file involves communication across the cluster.
- Reading a file :
  → The client first connect to the NameNode to initiate the read and request info on the file.
  → Then, the NameNode responds with the list of all blocks, as well as a list of DataNodes that contains a replica of each block (they are sorted by increasing distance to the client).
  → The client then connects to the DataNodes in order to download a copy of the blocks.
- Write a New File :
  → The client first connects to the NameNode formulating its intent to create a new file.
  → The NameNode responds with a list of DataNodes to which the content of the first block should be sent.
  → The client connects to the first DataNode and instructs it to organize a pipeline with the other DataNodes.
  → The client starts sending through the content of that block
  → The client receives regular acknowledgements from the 1st data Node, that the bits have been received.
  → When all bits have been acknowledged, the client

connect to the second block.

→ Once the last block has been written, the client informs the NameNode that the file is complete and asks to release the locks.

→ The NameNode checks for minimal replication through the DataNode protocol and gives a final acknowledgement to the client.

Big Data (Chapter 4 - Part 1)

Distributed File Systems

Replication Strategy

- By default, 3 replicas of each blocks are stored, although this can be changed by users for every file.
- Servers, called nodes, are organized in racks, and several racks are placed in the same room or data center.
- Distance between 2 nodes:
  - → Two nodes in the same racks have a distance of 2:
    - One edge from the first node to the racks
    - One from the racks to the other node.
  - → Two nodes in different racks have a distance of 4:
    - One edge from the first node to the first racks
    - One from the first racks to the data center
    - One from the data center to the other racks
    - One from the other racks to the other node.
- This distance is used by the NameNode to sort DataNodes by increasing distance from the client.
- The client that is writing the file is a process running on one of the nodes of the HDFS cluster.
- When users create clusters in public clouds such as AWS, or Azure or Google Cloud, the connect to the machines using SSH (a safe protocol for remotely accessing a machine)
- The first replica of the blocks, by default, gets written to the same machine that the client is running on (DataNode).
- The second replica is written on a DataNode sitting in a different rack than the client, that we call B.
- The third replica is written to another DataNode on the same rack B.

- And further replicas are written mostly at random.
- "At most one replica per node, and at most two replicas per rack".

Fault Tolerance and Availability

- HDFS has a single point of failure: the NameNode. If the metadata stored on it is lost, then all the data on the cluster is lost.
- For this reason the metadata is backed up (snapshot).
- Updates to the file system arriving after the snapshot has been made are stored in a journal, called edit log.
- The snapshot and edit log are stored either locally or on a network attached drive (not HDFS itself)
- If the NameNode crashes, it can be restarted, the snapshot can be loaded back into memory to get the file namespace and the mapping of the files to block IDs.
- Kinds of additional NameNodes:
  - → Checkpoint NameNode
  - → Backup NameNode
  - → Secondary NameNode
  - → Standby NameNode
- Checkpoint NameNode:
  - → The edit log is periodically merged back into a new, larger snapshot and reset to an empty edit log.
- Standby Name Node:
  - → Instantly take over from the NameNode in case of a crash.
- federated NameNode:

→ Several NameNodes can run at the same time, and each NameNode is responsible for specific directories within the hierarchy.

Using HDFS

- One of the ways to access HDFS is via the command line (a shell).
- HDFS command start with:

```
hdfs dfs <put command here>
```

- Although, we recommand the use of hadoop command:

```
hadoop fs <put command here>
```

- List the contents of the current directory:

```
hadoop fs -ls
```

- Print to screen the contents of a file:

```
hadoop fs -cat /user/hadoop/dir/file.json
```

- Delete a file:

```
hadoop fs -rm /user/hadoop/dir/file.json
```

- Create a directory:

```
hadoop fs -mkdir /user/hadoop/dir/file.json
```

- Upload files from local computer to HDFS:

```
hadoop fs -copyFromLocal localfile1 localfile2
                         /user/hadoop/targetdir
```

- Upload a file from HDFS to local computer:

```
hadoop fs -copyToLocal /user/hadoop/file.json
                       localfile.json
```

Paths and URIs:

- This what an HDFS URI looks like:

hdfs://www.ex.com:8021/user/hadoop/file.json

- This is what an S3 URI looks like:
  s3://my_bucket/directory/file.json
- In Azure Blob Storage:
  wasb://mycontainer@myaccount.blob.core.windows.net/directory/file.json
- On the local file system:
  file://home/user/file.json

Logging and Importing Data

- Apache Flume:
  → Lets you collect, aggregate and move log data to HDFS
- Apache Sqoop:
  → Lets you import data from a RDBMS to HDFS

Big Data (Chapter 4 - Part 2)

Hadoop Ecosystem

Hadoop Ecosystem

- The Hadoop Ecosystem is an open-source framework, which facilitates the distributed storage and parallel processing of huge volumes of data across clusters of nodes or computers.

Comparison between Traditional System & Hadoop Ecosystem

Traditional System	Hadoop Ecosystem
→ less scalable	→ Highly scalable
→ Vertical scaling	→ Horizontal scaling
→ Less fault-tolerant	→ More fault-tolerant
→ More investment needed	→ Economical
→ Handles only structured data	→ Handles any type of data

Components of the Hadoop Ecosystem

- **Data Storage:**
  - → Store large volumes of data in different formats (structured, unstructured and semi-structured).
  - → 2 components:
    - One for real-time data
    - One for archived data
- **Data Processing:**
  - → Facilitate the parallel processing of big data.
  - → 2 components:
    - YARN
    - MapReduce
- **Data Access:**
  - → Facilitate access to the big data repository to elicit the

information required as per business need.

→ Main components:
- HIVE
- PIG
- SQOOP
- Mahout
- Avro

- Data Management:
→ Monitor, manage, control and synchronize the data
→ Main Components:
- Oozie
- Flumes
- Ambari
- Zookeeper.

Data Storage Component

- Google File System (GFS):
→ The Apache Hadoop framework is based on the GFS.
→ GFS is a distributed systems to handle large data files developed by Google.
→ Based on Master Slave architecture:
- One master server
- Many Slave machines called chunk servers.
→ The files to be stored are divided into chunks and stored on the chunk servers.

- Bigtable:
→ GFS uses a scalable sparsely populated table which stores data in the form of a sorted key/value map.

→ The content of the Bigtable is divided into different tablets, each tablet is a block of contiguous rows.

. Hadoop Distributed file System (HDFS):

→ Hadoop's principle storage system is HDFS.

→ The master system is called the NameNode and the slave machines are called the DataNodes.

→ NameNode:
- Responsible for managing the file system namespace and allowing synchronized access to files by clients.
- Stores the metadata and performs operations like opening, closing and renaming files and directories.
- Provides the mapping of chunks storages to DataNodes.

→ DataNode:
- Store data chunks and transmit them to the clients in response to their requests.
- Perform chunk creation, deletion, and replication.

→ Features of HDFS:
- Hierarchical file organization: support the creation of directories to store files.
- Reliability: duplicate copies of files on different machines (replication factor).
- Data Integrity: uses the error checksum method.
- Robustness: heartbeats and data re-replication. Secondary NameNode copies the metadata from the primary NameNode to create checkpoints.

. Comparison Between GFS and HDFS:

Parameter	GFS	HDFS
Proprietary	Proprietary file system used by Google	Open-source file system used by anyone
Chunk Size	Files are divided into fixed sized chunks of 64 MB stored on chunk servers	Files are divided into fixed sized blocks of 128 MB stored on DataNodes
RPC: Remote Procedure Call	Communication Protocol	Chunks ↓ TCP Clusters	Blocks ↓ Clusters	. TCP/IP . RPC
Cache Management	Cache metadata are saved in client's memory	Private or publicly accessible
Database	BigTable	HBase

- HBase:
  → To handle huge volume datasets, the Hadoop framework uses a non-relational and scalable distributed data management system, Apache HBase.
  → HBase features:
  - Supports high linear and modular scalability
  - Consistent read and write operations

- Automatic splitting of large tables into blocks
- Robust
- Supports real-time queries using block caching and Bloom filters

. Comparison Between Bigtable and HBase:

Parameter	Bigtable	HBase
Proprietary	Proprietary System	Open-source project
Installation	Available only as a cloud service from Google	Can be installed on any environment
Concept	3 dimensional db: . Row key . Column key . Timestamp key	Column-oriented database
Data Integrity	CRC checksum	Checksum feature of HDFS
Security Management	Relies on Google Cloud Identity and Access Management	Uses ACL (Access Control Lists
Timestamp	In microseconds	In milliseconds

- Comparison between HDFS and HBase:

Parameter	HDFS	HBase
Concept	File distribution system with rigid architecture	HBase data storage runs on top of HDFS and supports dynamic changes
Access Method	Read/write operations. write once, read many times	Random access using hash table
Data Processing	Supports offline batch processing	Efficient for real-time processing
Latency	High latency	Low latency

Data Processing Component

- MapReduce:
  - → Main processing component of the Hadoop framework
  - → Facilitates the distributed parallel processing of data
  - → Google MapReduce can only work with the GFS file system, whereas Hadoop MapReduce is a part of Apache open-source project.
  - → Process large volumes of data (structured and un-structured)
  - → Divide and conquer approach; it splits a task into multiple sub goals and processes them concurrently on different machines
  - → MapReduce Stages:

- Map Stage: each map task executes different operations like filter, group, shuffle and sort on a split of input data on different machines simultaneously and generates a key-value pair.
- Reduce Stage: the output of the map stage is acted upon by the reduce stage where operations, like aggregation and summarization are performed.

. YARN:
→ "Yet Another Resource Negotiator"
→ Resource management component in Hadoop
→ Manage and allocate cluster resources and schedule the jobs with effective load balancing.
→ Responsible for monitoring and deployment of security controls.
→ YARN Components:
- Resource Manager: Jobs from users are submitted to the Resource Manager which schedules these jobs and allocates resources to them.
- Node Manager: Responsible for monitoring resource usage (processing units, memory, disk storage, network capacities) and report back to the resource manager.
- Application Master: Negotiate resources based on the job request from the Resource Manager and work with the Node Manager to execute tasks.
- Resource Containers: Contain a collection of resources including processing units, memory and disk storage.

→ YARN Features:

- Dynamic allocation of ~~feature~~ system resources
- Enables Hadoop real_time data processing
- Enhance resource utilization and application performance
- Supports various scheduling algorithms
- Reservation feature enable users to reserve cluster resources in advance.
- Increased scalability, support from 10000 to multiple ten of thousands of nodes.

Big Data (Chapter 5 - Part 1)

Hadoop Ecosystem

Data Access Component

- **Hive:**

→ Hive is an ETL and data warehousing tool developed by Apache.

→ Hive supports the summarization, querying and analysis of unstructured and semi-structured data.

→ The read, write and update operations are performed through Hive Query Language (HQL) and translates queries into MapReduce jobs automatically.

→ **Components of Hive:**
- Hive Command line Interface
- JDBC/ODBC driver (Java DataBase Connectivity, Object DataBase Connectivity).

→ **Hive Features:**
- High Scalability
- Real-time processing, batch query processing
- Compatibility with each and every SQL primitive data type
- Support for user-defined functions (UDF) along with pre-set functions
- Automatic translation of queries into map and reduce functions
- Efficient extensive sequential scans.

→ **Limitations of Hive:**
- Not effective where fast response times are crucial because queries may encounter high delays
- Read-based system, not so many write operations

- Apache Pig:
  → Platform for analyzing large datasets
  → High-level language for describing data analysis programs and infrastructure for evaluating those programs
  → Two parts of Pig:
    - Pig Latin (programming language)
    - Pig Runtime (environment)
  → Pig supports many data types (int, float, double, long, chararray, bytearray)
  → Pig advanced data types are:
    - Map
    - Tuple
    - Bag
  → Pig Latin can easily generate a sequence of MapReduce jobs in an abstract way.
- Apache Drill: