# Prosperity Fund Global Future Cities Programme

Bangkok – Integrated Data Hub

Introduction to Big Data for Data Engineer

November 2020

## Key Objectives Of This Course

- An understanding and appreciation of the depth and breadth of applications of Big Data Analytics.

## Contents

1

# What Is Big Data : Beyond The Hype

# Chapter 1 : Contents

# Chapter 1 : Structured, Semi-Structured & Unstructured Data

## Structured Data

- Structured data refers to all data which can be stored in a relational database table with rows and columns.
- It has relational keys and can be easily mapped into pre-designed fields. Today, such data is the most processed in development and the simplest way to manage information.
- Examples : CSV, TSV, Excel files, other relational database dumps, etc.

**5 – 10% of all data**

## Semi-Structured Data

- Semi-Structured Data refers to information that does not reside in a relational database but that does have some organizational properties that make it easier to analyse. They exist to ease space, clarity or compute.
- Certain types of semi-structured data can be stored in a relational database after some pre-processing.
- Some types of semi-structured data cannot be stored in a relational database.
- Examples : JSON, XML, txt, etc.

**5 – 10% of all data**

## Unstructured Data

- Unstructured data may have an internal structure, but is considered unstructured because it does not fit neatly in a relational database.
- Unstructured data is everywhere. Most individuals and organizations conduct their lives around unstructured data.
- Most unstructured data are either machine generated or human generated.
- Examples : Tweets, Emails, Photos/Videos, Docx, PDF, etc.

**80 – 90% of all data**

# Chapter 1 : What Launched The Big Data Era?

- The last few years has seen a growing torrent of data. 90% of all data in the world has been collected over the last few years.
- Social media, IoT have been major contributors of this torrent of data.
- In 2018, Facebook, a social media platform, set a record of having one billion people login in a single day. 30 billion shares on. Facebook every month .
- Storage and compute have decreased significantly over the last decade with a corresponding increase in network bandwidth. 1 Gbps in 2001, 40 Gbps in 2018.
- Paradigm shifts in data processing. Instead of transferring the data over the network, to be processed, data is now processed where it resides.
- Affordable, robust and horizontally scalable alternatives to expensive data warehousing solutions from the past.

# Chapter 1 : Big Data Growth Trends

- Every human on the planet will be creating 1.7 megabytes of data each every second.
- The accumulated world data will grow to 44 zettabytes that’s 44 trillion gigabytes.
- The revenues generated by BDA worldwide were $42 billion in 2018and in 2027, they’re projected to increase to $103 billion with a CAGR of 10.5%.
- Hadoop is the most popular big data processing tool and its market is expanding fast and anticipated to hold a CARG of 53.7% for the period of 2015 to 2022.
- The Chinese big data market is one of the fastest growing worldwide with a CAGR of 31.72%. By 2020, the revenue is projected to reach ¥57.8 billion – that’s $9 billion! In 2014, they were only at ¥8.4 billion, or $1.2 billion.
- Statistics show big data adoption can increase retail sales by 3% to 4%.
- As more and more companies harness the power of BDA, the need for tools to process the information rises as well. big data software is projected to grow at a CAGR of 12.6%, reaching $46 billion in 2027.
- By 2020, the IoT is projected to generate over $300 billion annually. The market will grow at a 28.5% CAGR.

# Chapter 1 : Industries Finding Value In Big Data

## Healthcare

- Consulting firm McKinsey estimated that big data analytics adoption can save up to 17% of healthcare costs.
- In 2013 that amounted to $493 billion dollars in reductions.

## Banking

- Adoption of big data in the field will bring up to 18% increase in revenue.
- American express accurately predict 24% of the accounts that will close within 4 months.

## Media

- Miniclip is one of the largest gaming websites, with more than 70 million users using bigdata to determine which games will be more successful.
- Netflix uses Hadoop, its own open-source solutions such as Lipstick and Genie, to gather, store, and process massive amounts of information. They use these platforms to predict what content to create and promote to viewers.

## Retail

- General Electric increased their efficiency by using information from sensors on turbines and engines.
- The company estimates big data can boost US productivity by 1.5% YoY.

# Chapter 1 : Sources of Big Data

- **Organization Generated Data**
  Generated from all the daily transactions that take place both online and offline. Invoices, payment orders, storage records, delivery receipts – all are characterized as transactional data yet data alone is almost meaningless, and most organizations struggle to make sense of the data that they are generating and how it can be put to good use.

- **User Generated Data**
  Social data comes from the Likes, Tweets & Retweets, Comments, Video Uploads, and general media that are uploaded and shared via the world's favorite social media platforms. This kind of data provides invaluable insights into consumer behavior and sentiment and can be enormously influential in marketing analytics. The public web is another good source of social data, and tools like Google Trends can be used to good effect to increase the volume of big data.

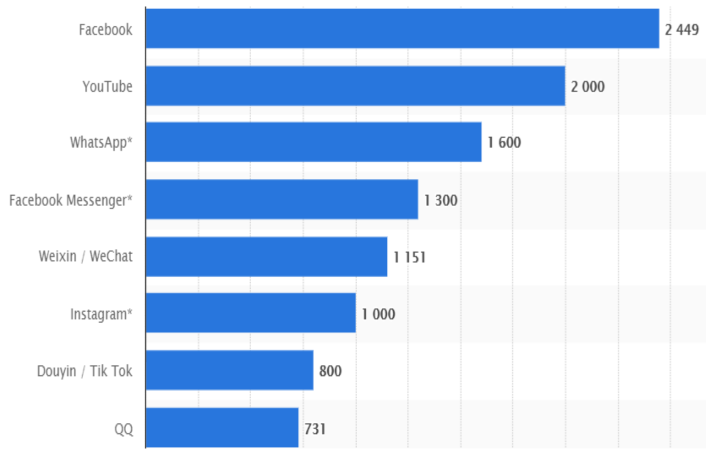
- **Machine Generated Data**
  Machine data is defined as information which is generated by industrial equipment, sensors that are installed in machinery, and even web logs which track user behavior. This type of data is expected to grow exponentially as the internet of things grows ever more pervasive and expands around the world. Sensors such as medical devices, smart meters, road cameras, satellites, games and the rapidly growing Internet Of Things will deliver high velocity, value, volume and variety of data in the very near future.

# Chapter 1 : Sources of Big Data (Organization Generated Data)

- UPS delivers 16 million shipments per day. They get around 40 million tracking requests and it has around 16 petabytes of data.
- Walmart gets 250 million customers across its 20,000 stores. They collect 2.5 petabytes of data per hour.

# Chapter 1 : Sources of Big Data (User Generated Data)

- User activity on social media has been one of the major contributors of the explosive growth in data volumes.
- Social media data comes from the Likes, Tweets & Retweets, Comments, Video Uploads and other general media that are uploaded and shared via popular social media platforms.
- In a study conducted in 2019, the following statistics help illustrate this explosive growth in social media data volumes:
    - Snapchat: Over 527,760 photos shared by users
    - LinkedIn: Over 120 professionals join the network
    - YouTube: 4,146,600 videos watched
    - Twitter: 456,000 tweets sent or created
    - Instagram: 46,740 photos uploaded
    - Netflix: 69,444 hours of video watched
    - Giphy: 694,444 GIFs served
    - Tumblr: 74,220 posts published
    - Skype: 154,200 calls made by users



Figures indicate number of users (in millions)

# Chapter 1 : Sources of Big Data (Machine Generated Data)

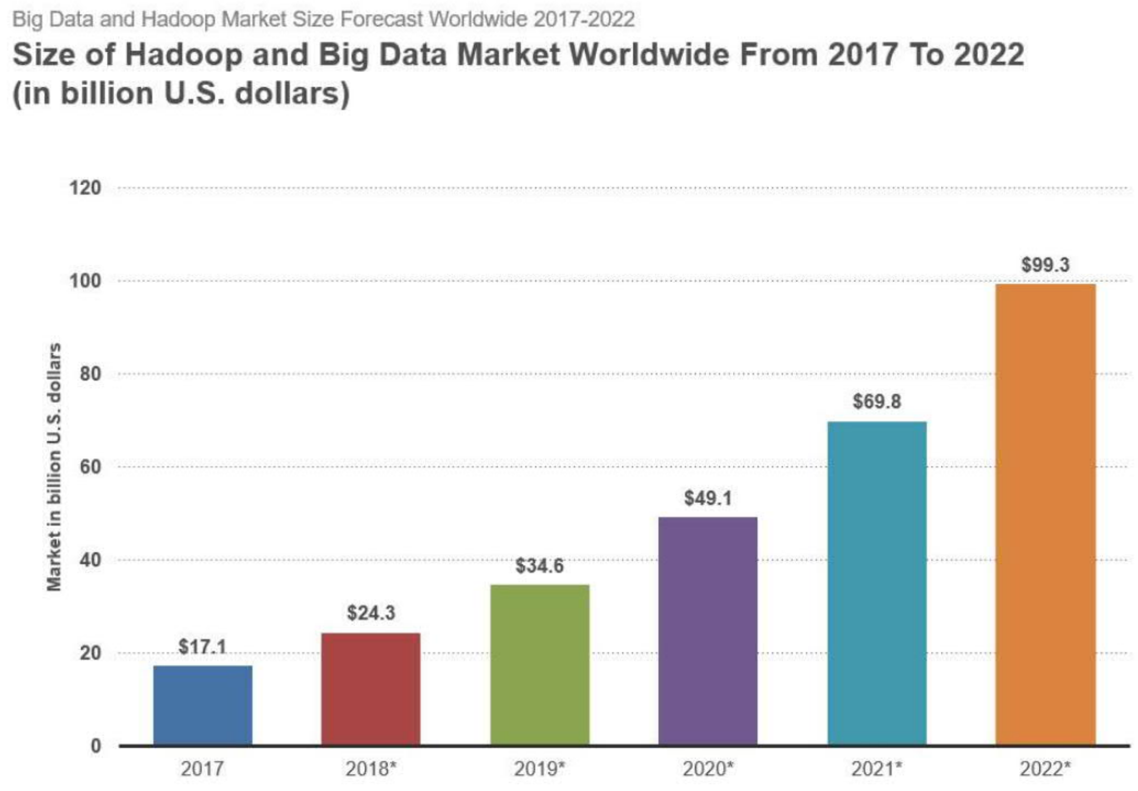
- A Boeing 787 aircraft generates up to 1 terabyte of data per flight.
- An Airbus A350 aircraft has close to 6,000 sensors generating up to 2.5 terabytes of data per day.
- NYSE (New York Stock Exchange) generates about 1 terabyte of new trade data per day.
- Facebook processes a few hundred terabytes of data per hour.

# Chapter 1 : Limitations Of The Past Data Warehouses

- Storing large volumes of data was a challenge because often the storage had a strict upper limit, i.e. limited to one system.
- Storing heterogeneous data (i.e. Unstructured, Semi-Structured and Structured) was a challenge.
- Limitations with vertical scaling. The systems were not designed for distributed storage and processing.
- Disparity between advances in storage capacity versus data access and processing speeds.
- Excessive IO operations incurred due to limitations with transfer and seek involving large datasets. For example, if the I/O channel had a bandwidth of 100 Mbps and 1 TB dataset was being processed, then the operation would take approximately 3 hours. In contrast, modern distributed systems with identical I/O channel characteristics would take approximately 43 minutes on a 4 node cluster. Shorter seek times are possible with increased number of nodes in the cluster.

# Chapter 1 : Growth Trends In Big Data Market

The growth trends in the Big Data Industry are an indicator of enterprise-wide Big Data adoption.



# Chapter 1 : The 6 Vs Of Big Data

## Volume

- The ability to ingest, process and store very large datasets.
- The data can be generated by machine, network, human interactions on system etc.
- The data is measured in petabytes or even in exabytes.

## Variety

- It refers to data from different sources and types which may be structured or unstructured.
- The unstructured data creates problems for storage, data mining and analyzing the data.
- With the growth of data, even the type of data has been growing fast.

## Velocity

- Speed of data generation and frequency of delivery.
- The data flow is massive and continuous which is valuable to researchers as well as business for decision making for strategic competitive advantages and ROI.
- For processing of data with high velocity tools for data processing known as Streaming analytics were introduced.

# Chapter 1 : The 6 Vs Of Big Data

## Veracity

- It refers to the biases, noises and abnormality in data.
- Ensuring accuracy of the dataset
- Verify that the data is suitable for its intended purpose and usable within the analytic model.
- Processes to keep ‘dirty data’ away from accumulating in systems.

## Variability

- Variability refers to data whose meaning is constantly changing.
- This refers to establishing if the contextualizing structure of the data stream is regular and dependable even in conditions of extreme unpredictability.
- Develop sophisticated programs in order to be able to understand context in them and decode their exact meaning.

## Value

- Refers to purpose, scenario or business outcome that the analytical solution has to address.
- Ensuring if the data is worth being stored or collected.
- Ensuring that ethical considerations are met.

2

## An Introduction To Hadoop

# Chapter 2 : Contents

# Chapter 2 : What Led To The Development Of Hadoop?

## An Overview of RDBMS

### RDBMS

- RDBMS stands for Relational Database Management System.
- Relational databases became dominant in the 1980s.
- Data in a relational database are organized as a set of tables with columns and rows.
- Most efficient and flexible way to access structured information.

### Database

- A database is an organized collection of structured information or data.
- A database is controlled by a database management system (DBMS).
- Data is modelled in rows and columns to make processing and data querying efficient.

### SQL

- SQL is a programming language used by nearly all relational databases to query, manipulate and define data.
- SQL was first developed at IBM in the 1970s with Oracle as a major contributor, which led to implementation of the SQL ANSI standard.
- SQL has spurred many extensions from companies such as IBM, Oracle, and Microsoft.

### Table

- The data in an RDBMS is stored in database objects which are called as tables.
- Table is a collection of related data entries and it consists of numerous columns and records.
- Each row in a table is called as a record.
- A column is a vertical entity in a table that contains all information associated with a specific field in a table.

# Chapter 2 : What Led To The Development Of Hadoop?

## A Brief History

- **2002**: Doug Cutting and Mike Cafarella started work on the Apache Nutch project, a process of building a search engine system that could index up to 1 billion pages. Any search engine requires ability to store and process very large files.
- **2003**: A paper on Google File System was published by Google. Cutting and Cafarella realized that this paper could solve their problem of storing very large files.
- **2004**: Google published one more paper on the technique MapReduce, which was the solution of processing those large datasets.
- **2005**: Cutting and Cafarella realized that Nutch was limited to only clusters of up to 40 nodes. It was critical for Nutch to run on larger clusters.
- **2006**: Cutting joined Yahoo along with Nutch project, with the goal of creating an open-source, reliable, scalable computing framework. He separated the distributed computing parts from Nutch and formed a new project called Hadoop.
- **2007**: Yahoo successfully tested Hadoop on a 1000 node cluster and started using it.
- **2008**: Yahoo released Hadoop as an open source project to Apache Software Foundation (ASF). In the same year, ASF tested Hadoop with 4000 node cluster.
- **2009**: Hadoop was successfully tested to sort a PetaByte of data in less than 17 hours. Cutting left the Yahoo and joined Cloudera to fulfill the challenge of wide spread adoption of Hadoop.
- **2011**: ASF released Apache Hadoop version 1.0.

# Chapter 2 : What is Hadoop?

## An Overview

- **Hadoop** is a distributed processing framework written in Java, capable of processing large data sets across clusters of computers using simple programming models.
- It is designed to scale up from a single machine to thousands of machines, each offering local computation and storage.
- Hadoop is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers.
- Hadoop makes it possible to run applications on systems with thousands of commodity hardware nodes, and to process terabytes of data.
- Two core concepts which powered Hadoop were the Hadoop MapReduce Paradigm and the Hadoop Distributed File System (HDFS).
- There were two major challenges with BigData :
  - Big Data Storage : To store huge volume of data in a flexible infrastructure that scales up in a cost effective manner, was critical. It was addressed using HDFS which could save huge volume of data.
  - Big Data Processing : Processing huge volume of data was a challenge efficiently. It was addressed using MapReduce, which enables processing large volumes of data in a distributed manner.

# Chapter 2 : What is Hadoop?

## Features of Hadoop

### Flexibility

- Hadoop is flexible in terms of ability to deal with structured, semi-structured and unstructured data.

### Reliability

- When machines are working in tandem, if one of the machines fail, another machine will take over the responsibility and work in a reliable and fault tolerant manner.
- Hadoop infrastructure has in-built fault tolerance features and hence, provides reliability in the event of one or more nodes failing.

### Economical

- Hadoop uses commodity hardware, thus saving cost by removing the need for specialized hardware.

### Scalability

- A Hadoop cluster can easily be scaled up to comprise of thousands of nodes.
- Yahoo has a 32,000 node cluster.

# Chapter 2 : What is Hadoop?

## RDBMS Vs Hadoop

| RDBMS | Hadoop |
|---|---|
| Scale up | Scale out |
| Structure data | Structured, semi-structured  and unstructured |
| Normalized / Joins | De-normalized |
| Designed for OLTP / Point queries / Real time processing | OLAP / Analytical queries / Batch processing |
| Declarative style (SQL) | Functional programming (MR) |
| Write on schema | Read on schema |
| Hot data | Cold data |

# Chapter 2 : Hadoop MapReduce Paradigm

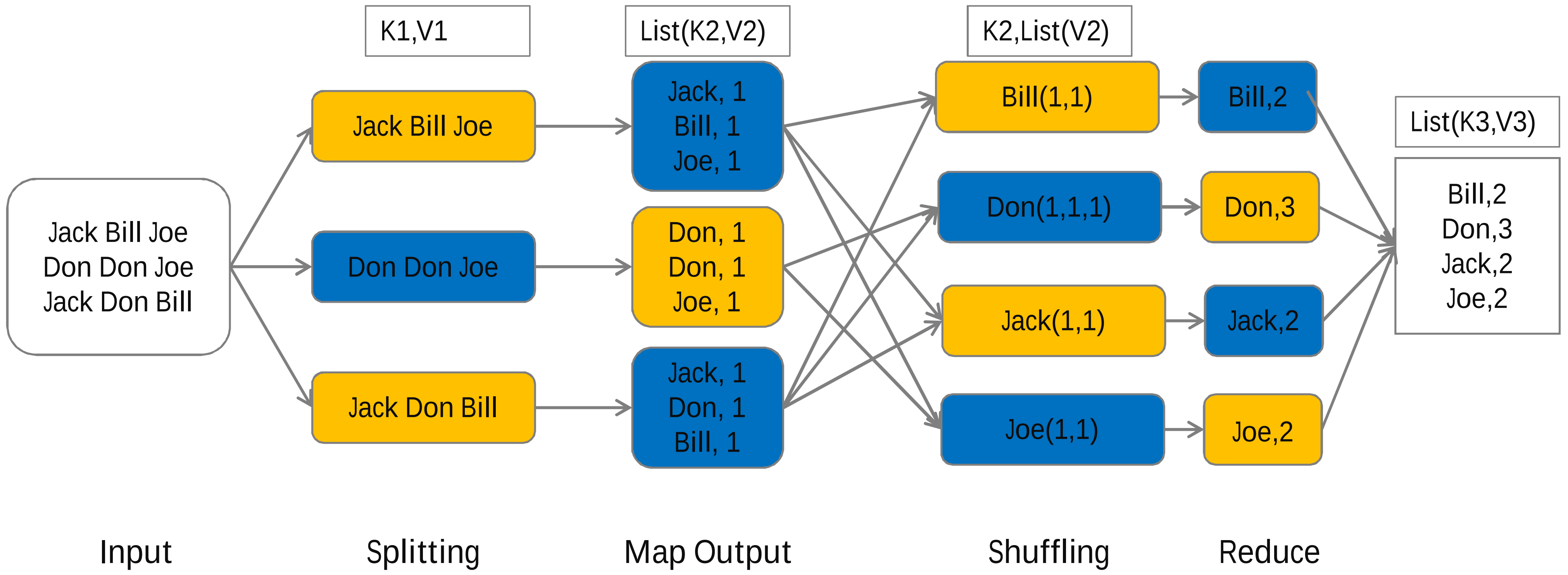
## An Introduction To Hadoop MapReduce Paradigm

- Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (typically multi-terabyte data-sets) in-parallel on large clusters comprised of hundreds or even thousands of nodes of commodity hardware in a reliable, fault-tolerant manner.
- A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and (when required) re-executing the failed tasks.

# Chapter 2 : Hadoop MapReduce Paradigm

## An Illustration Of A Hadoop MapReduce Job

The illustration below demonstrates the MapReduce job. A simple task of performing a Word Count is used for the illustration. The splitting phase converts the incoming text (in this example, 3 words) into key value pairs. The map output has a list of key value pairs. The shuffling phase aggregates based on the keys. The Reduce phase gets the counts.



# Chapter 2 : Hadoop MapReduce Paradigm

## MapReduce Building Blocks

### Map
During this phase, data transformations are performed at a record level. We do not do aggregations in the Map phase.

### Split
One split is generated for each block of data. Each mapper works on one split, if there are 5 splits generated for an input file, 5 mappers are initiated. A split is logical representation of the data which MapReduce uses to initiate mappers.

### Shuffle and Sort
Shuffle phase transfers the map output from Mapper to a Reducer in MapReduce. Sort phase covers the merging and sorting of map outputs. Data from the mapper are grouped by the key, split among reducers and sorted by the key. Every reducer obtains all values associated with the same key.

### Partitioner
A Partitioner is used to distribute the Map emitted data across the Reducers.

### Reducer
It is used to perform the aggregations.

# Chapter 2 : An Introduction To Hadoop Distributed File System (HDFS)

## An Overview

- HDFS is an open source , distributed file written in Java to store large volumes of data making use of a cluster of nodes using commodity hardware in highly reliable and fault tolerance manner.
- HDFS is a distributed file system, based on the Google File System, an early distributed file system. HDFS was previously known as the NDFS (Nutch Distributed File System).
- HDFS is the core component of Hadoop suitable for the distributed storage and processing.
- HDFS is designed to be written once and read many times. This is because data written into HDFS is considered immutable. There is no update, only append or delete.
- Hadoop provides a command line interface to interact with HDFS.
- HDFS provides file permissions and authentication.
- A file stored in HDFS is chopped up into 128 MB chunks internally by the framework which can reside on different machines in the cluster. Each chunk of data is referred to as a HDFS block.
- The blocks of a file are replicated in different machines for fault tolerance. The default replication in Hadoop is 3. The replication in HDFS ensures fault tolerance.

# Chapter 2 : An Introduction To Hadoop Distributed File System (HDFS)

## Goals Of Hadoop Distributed File System (HDFS)

- **Fault Detection And Recovery**
  Since a large number of commodity hardware were expected to be used, failure of components was expected to be frequent. HDFS was therefore required to have mechanisms for quick and automatic fault detection and recovery.

- **Large Cluster Support**
  HDFS was expected to support clusters comprised of hundreds of nodes, manage the applications having huge datasets.

- **Data Locality**
  An early goal of HDFS was to enable computation on data where it resides. This goal was particularly important to efficiently process large datasets, as it reduces the network traffic and increases the throughput.

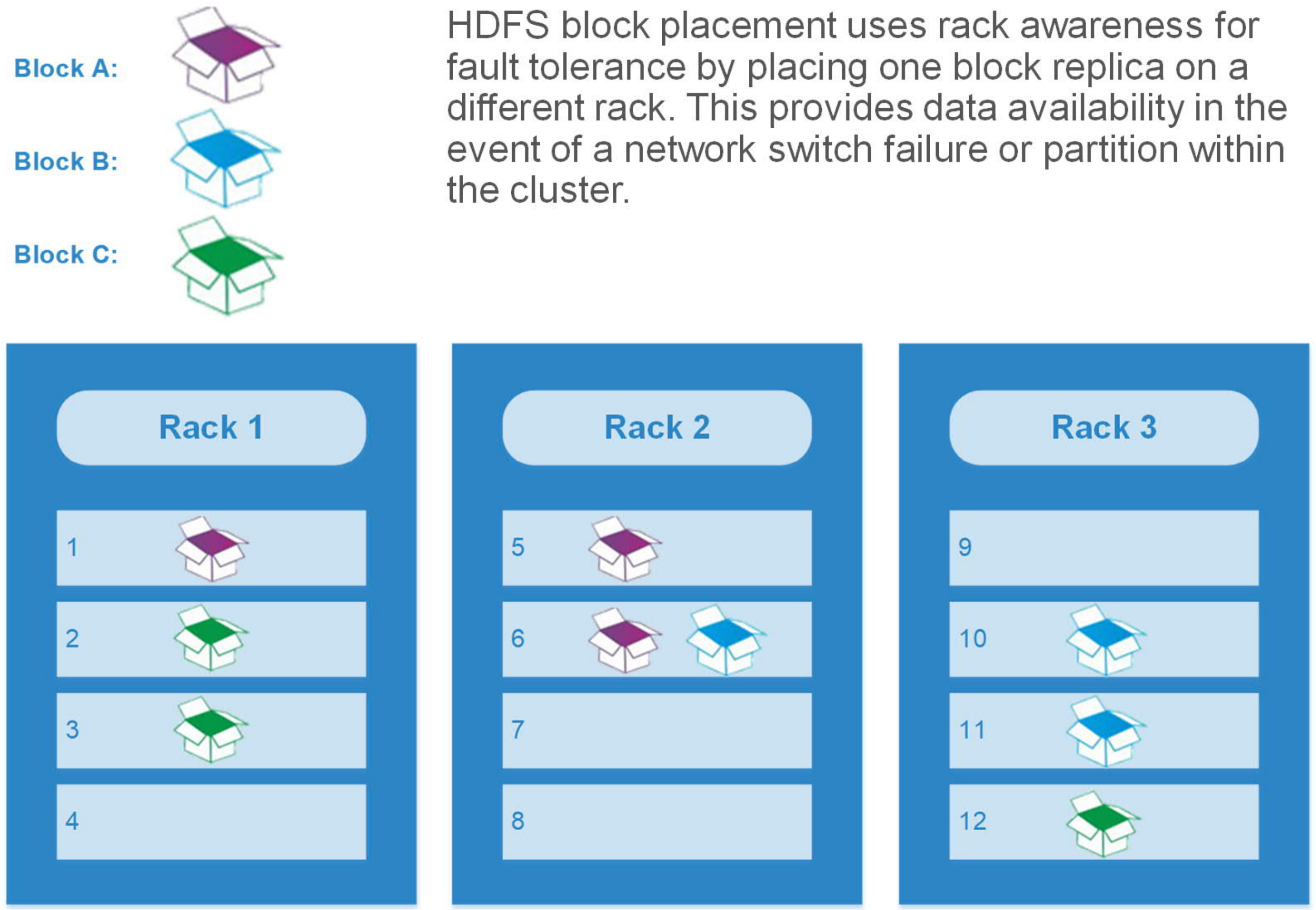
- **Streaming Data Access**
  HDFS was expected to provide streaming data access. This was enabled with the help of Posix interface on top of HDFS.

- **Optimizing IO**
  In most moderating operating systems, a block is 4KB. There is a one seek/transfer operation per block of data. However, for HDFS, each block is 128MB by default. This design was introduced to reduce the number of I/O operations.

- **Simple coherency model**
  HDFS is a single window like system from every data node we will be able to view the entire HDFS.

# Chapter 2 : An Introduction To Hadoop Distributed File System (HDFS)

## Rack Awareness



HDFS block placement uses rack awareness for fault tolerance by placing one block replica on a different rack. This provides data availability in the event of a network switch failure or partition within the cluster.

# Chapter 2 : An Introduction To Hadoop Ecosystem

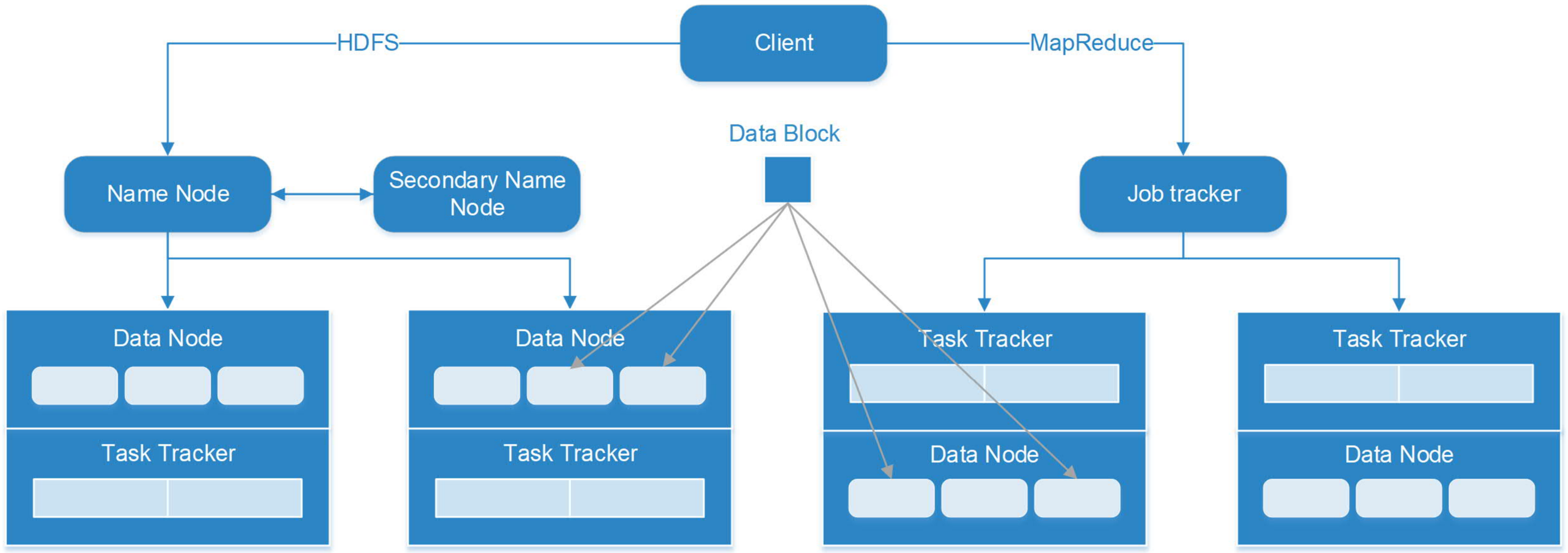
## Hadoop 1, Hadoop 2 and the rest of the Hadoop Ecosystem

The core components of Hadoop 1 and 2 are as follows :

- **Hadoop 1** :
  - HDFS (Storage)
  - Map Reduce (platform + application)
  - Common utilities
- **Hadoop 2** :
  - HDFS 2 (platform)
  - Map Reduce (application)
  - Yarn (platform)
- **The Hadoop Ecosystem** is comprised of the following :
  - Pig, Hive, HBase, Sqoop, Oozie, HUE, Mahout, Tez, Ranger, Sentry, Kmoxgate, Zookeeper, Lipstick, Flume, Falcon, Chukwa, Scribe, Kafka, Azkaban, Ambari, Samza, Drill, Impala, Presto, Tajo, Bookkeeper, Avro, Parquet, Treveni, Cassandra, Xaseure, Thrift, MRUnit

# Chapter 2 : Hadoop 1

## An Overview Of Hadoop 1 Components



# Chapter 2 : Hadoop 1

## Hadoop 1 Daemons

### JobTracker

- The Job Tracker is the master daemon for both Job resource management and scheduling / monitoring of jobs. It acts as a liaison between Hadoop and the running applications. It process runs on a separate node and not usually on a DataNode.
- The Job Tracker is an essential Daemon for MapReduce execution in MRv1. It is replaced by ResourceManager/ApplicationMaster in MRv2.
- The Job Tracker receives the requests from client for MapReduce job execution.
- It queries the NameNode to determine location of the data.
- It finds the best TaskTracker nodes to execute tasks based on the data locality (proximity of the data) and the available slots to execute a task on a given node.
- JobTracker monitors the individual TaskTrackers and the submits back the overall status of the job back to the client.
- JobTracker process is critical to the Hadoop cluster in terms of MapReduce execution.
- When the JobTracker is down, HDFS will still be functional but the MapReduce execution can not be started and the existing MapReduce jobs will be halted.

# Chapter 2 : Hadoop 1

## Hadoop 1 Daemons

### DataNode

- DataNodes are responsible for data storage.
- This is actual worker node were Read/Write/Data processing is handled.
- Upon receiving instructions from the Master, it performs the creation, replication, deletion of data blocks.
- Commodity hardware can be used for hosting DataNodes.

### TaskTracker

- TaskTracker runs on all the DataNodes.
- Mapper and Reducer tasks are executed on DataNodes administered by TaskTrackers.
- TaskTrackers will be assigned Mapper and Reducer tasks to execute by JobTracker.
- TaskTracker will be in constant communication with the JobTracker signalling the progress of the task in execution.
- TaskTracker failure is not considered fatal. When a TaskTracker becomes unresponsive, JobTracker will assign the task executed by the TaskTracker to another node.
- The TaskTracker is replaced by Node Manager in MRv2.

# Chapter 2 : Hadoop 1

## Hadoop 1 Daemons

### NameNode

- The NameNode stores the metadata of the actual data being stored. For example, filename, path, the number of Data Blocks, Block IDs, Block Location, the number of replicas, slave related configuration.
- It manages file system namespace.
- It regulates client access requests for data files.
- It assigns work to the slave nodes (DataNodes).
- The NameNode executes file system name space operations such as opening/closing files, renaming files and directories.
- NameNode requires a large memory, as it stores the metadata for fast retrieval. This should be hosted on reliable hardware.

### Secondary NameNode

- Stores a copy of FsImage file and edits log.
- Periodically applies edits log records to FsImage file and refreshes the edits log. And sends this updated FsImage file to NameNode so that NameNode doesn't need to re-apply the EditLog records during its start up process. Thus Secondary NameNode makes NameNode start up process fast.
- If NameNode is failed, File System metadata can be recovered from the last saved FsImage on the Secondary NameNode but Secondary NameNode can't take the primary NameNode's functionality.
- Check pointing of File system metadata is performed.

# Chapter 2 : Hadoop 1

## Hadoop NameNode Concept, FsImage

### FS image

- The entire file system namespace, including the mapping of blocks to files and file system properties, is stored in a file called the FsImage (File System Image).
- The FsImage is stored as a file in the NameNode’s local file system.
- It is read only file with respect to clients.
- It is used for metadata lookup (for lookup both FS image and edits are required).
- There is a minor difference in between the version of metadata which exists in memory with respect to the version on disk.
- If the cluster is stopped and restarted, the name node retrieves the disk version of metadata into memory.
- The FsImage needs to fit into the RAM of the NameNode. If the entire metadata cannot be loaded into NameNode memory, the NameNode will not start.

# Chapter 2 : Hadoop 1

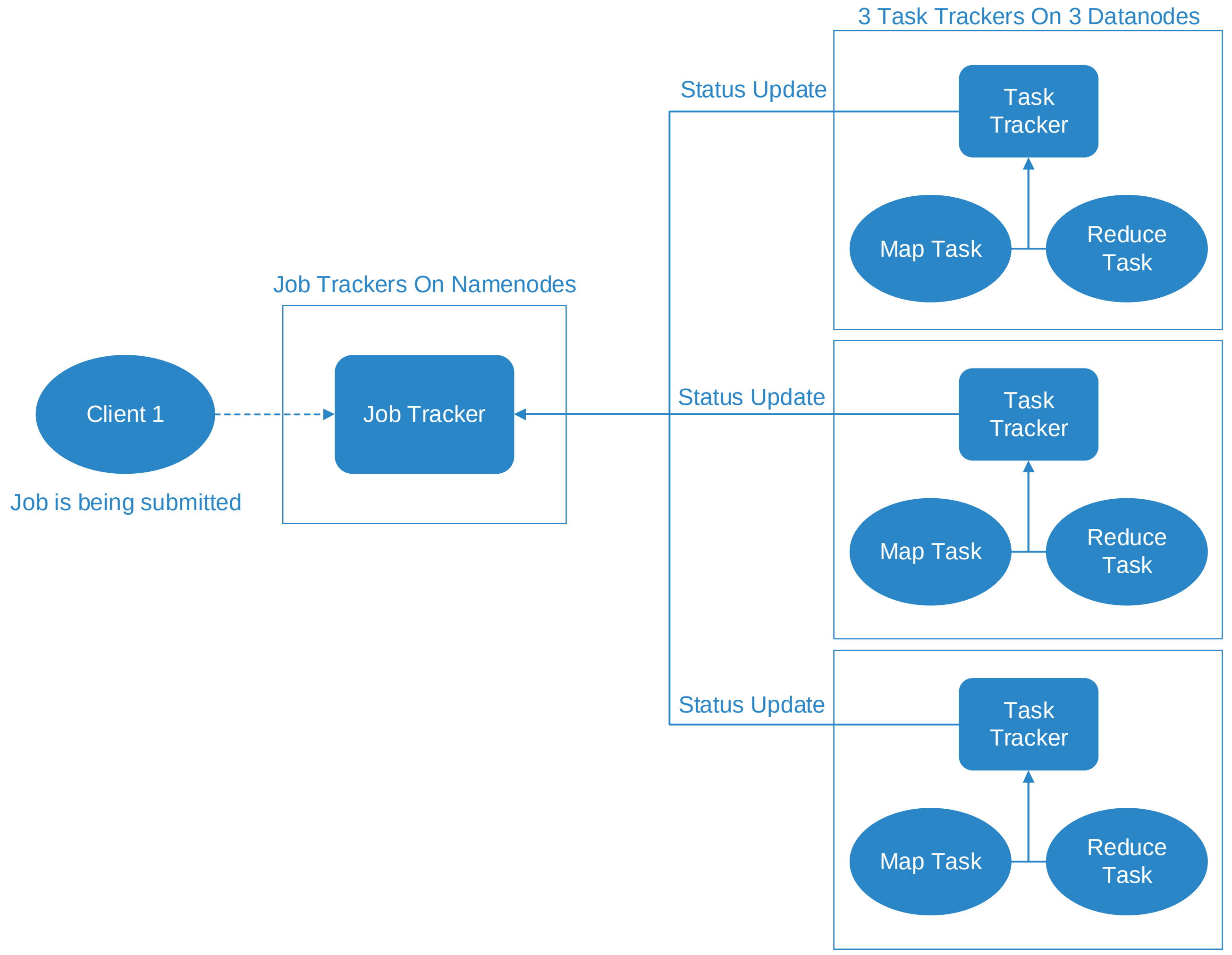
## Hadoop NameNode Concept, Edits

### Edits

- Edits is a small window temporary file. It contains the transactions on the HDFS from the last saved checkpoint.
- It is used to record all clients interactions with the HDFS in the current window.
- It contains metadata for file creation, updates  and deletions.
- The window size is 1 hour by default. The values are configurable and can be changed by the hadoop administrator.
- The window can also be configured on the edits size.
- Once the window of the configured interval is over, the edits gets merged with the FS Image file.
- When the NameNode is restarted, the Edits file gets merged with the FS Image irrespective of the window internal.
- When the NameNode starts, it loads the entire FS Image into memory. This leads to a delay of few seconds for the NameNode service to be functional. During this interval, the NameNode stays in safe mode.
- During safe mode, no transactions can be done on the NameNode. It does not allow any modifications to the file system. It is also called as read-only mode.

# Chapter 2 : Hadoop 1

## Interaction Of Hadoop 1 Components In A Job Flow



# Chapter 2 : Hadoop 1

## Interaction Of Hadoop 1 Components In A Job Flow

- The client applications submit jobs to the JobTracker daemon running on the NameNode.
- The JobTracker interacts with the NameNode to determine the location of the data.
- The JobTracker then locates the TaskTracker nodes with available slots at or near the data. This is done to avoid unnecessary movement of data over the network. The concept of data locality will be addressed later.
- The JobTracker submits the work to be done to the chosen TaskTracker nodes.
- The TaskTracker nodes are monitored by the JobTracker. If the TaskTracker nodes do not submit heartbeat signals often enough, they are deemed to have failed and the work is scheduled on a different TaskTracker.
- A TaskTracker will notify the JobTracker when a task fails. Upon receipt of a failure notification from a TaskTracker, the JobTracker decides what to do next :
  - it may resubmit the job elsewhere,
  - it may mark that specific record as something to avoid,
  - it may even blacklist the TaskTracker as unreliable.
- When the work is completed, the JobTracker updates its status.
- Client applications typically periodically poll the JobTracker for information on job status.

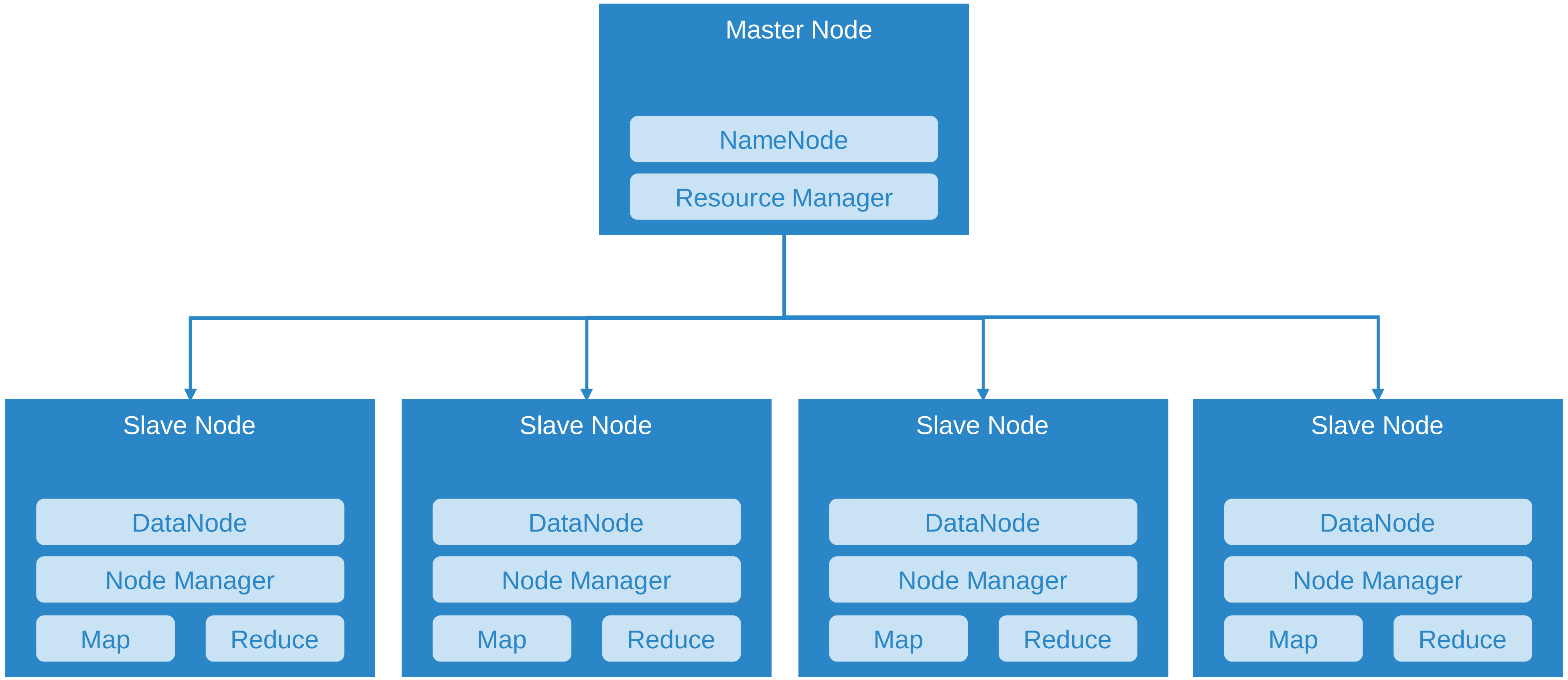
# Chapter 2 : Hadoop 1

## Limitations Of Hadoop 1.x

- JobTracker performs Resource Management, Job Scheduling, Job Monitoring and Re-scheduling of Jobs. Therefore significant amount of time and effort spent on managing the life-cycle of the applications.
- JobTracker and NameNode are single points of failure. Hadoop 1.x does not have support for high availability of JobTracker and NameNode.
- It cannot scale beyond 4000 Nodes per Cluster because of the JobTracker overburden.
- It supports only MapReduce jobs. Spark jobs cannot be run using hadoop 1.x. It makes hadoop 1.x paradigm specific.
- Hadoop 1.x is a batch processing framework there is no support for real time or near real time data processing.
- It has static Map and Reduce Slots. Once resources to Map-Reduce jobs are assigned, it cannot re-use them even though some slots are idle. These slots could be used to process other jobs.

# Chapter 2 : Hadoop 2

## An Overview Of Hadoop 2 Components



# Chapter 2 : Hadoop 2

## Hadoop 2 Daemons

### Resource Manager

- The ResourceManager is a master daemon which arbitrates available resources in the system among the competing applications and not concerning itself with per-application state management.
- The ResourceManager is not involved in the actual execution of the job unlike JobTracker in Hadoop 1. It is because of this clear segregation in roles that enables Hadoop 2 to be able to address scalability issues and support programming paradigms other than MapReduce.
- The ResourceManager works together with the per-node NodeManagers and the per-application ApplicationMasters, both of which are actually involved the execution of a job.
- NodeManagers take instructions from the ResourceManager and manage resources available on a single node. A NodeManager is a slave daemon running on each slave node.
- ApplicationMasters are responsible for negotiating resources with the ResourceManager. They work with the NodeManagers to start the containers. A container, in YARN, refers to a package of resources including RAM, CPU, Network, HDD, on a single node.
- The ResourceManager comprises of two components : Scheduler & ApplicationsManager.

# Chapter 2 : Hadoop 2

## Hadoop 2 Resource Manager Components

### Scheduler

- The Scheduler is responsible for allocating resources to the various running applications subject to constraints of capacities and queues.
- It performs its scheduling function based on the resource requirements of the applications such as memory, CPU, disk, network etc. Currently, only memory is supported and support for CPU will be available in future versions.

### Application Manager

- It is responsible for negotiating resources with the ResourceManager and for working with the NodeManagers to start the containers.

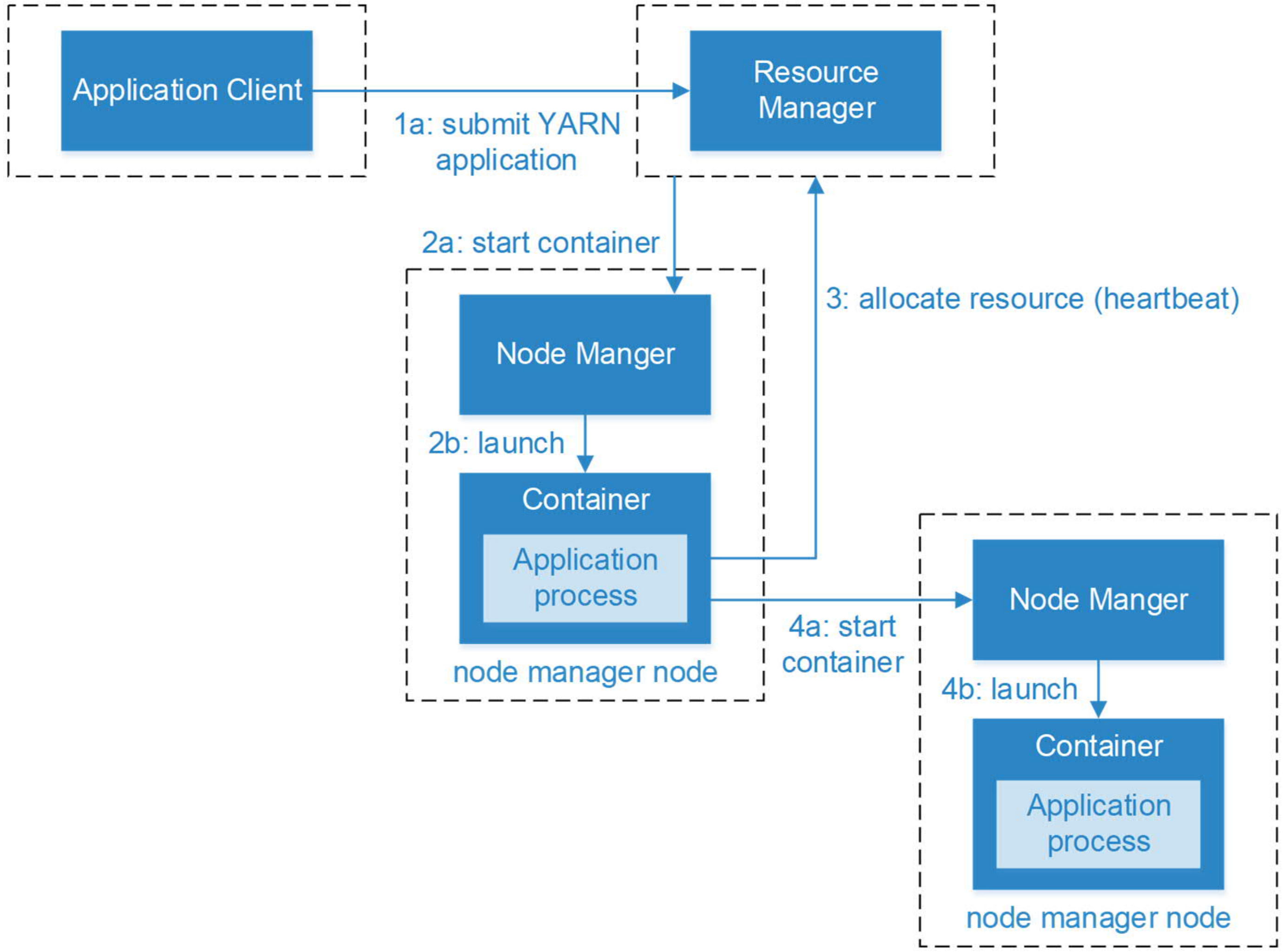
# Chapter 2 : Hadoop 2

## Application Submission in YARN

- User submits jobs to the Job Client present on client node. The Job Client requests for an application id from Resource Manager.
- A job consists of JAR files, class files and other required files which are copied to HDFS under a directory named after the ID assigned to the application, so that the job's resources (files) can be copied to nodes where it needs to be run.
- A job is submitted to the ResourceManager, which then contacts the NodeManager to launch a new container and run an ApplicationMaster in it.
- An ApplicationMaster checks the splits on which the job has to run and creates one task per split. It then interacts with ResourceManager for allocating the resources. The ResourceManager allocate resources considering the data locality so that the tasks can be run on same machine on which the data blocks are present. This is done to avoid unnecessary movement of data.
- The ApplicationManager gets information about the resources from the ResourceManager and it launches the container through NodeManager. Before running the task it copies all the job resources from HDFS.
- The task sends progress updates to the ApplicationMaster from time to time.
- In case of failure, ApplicationMaster launches the task on some other container.

# Chapter 2 : Hadoop 2

## YARN Job Flow



# Chapter 2 : Hadoop 2

## Schedulers in YARN

There are two  types of schedulers in YARN which are widely used :

### Capacity Scheduler

- The Capacity Scheduler is designed to run Hadoop applications as a shared, multi-tenant cluster in an operator-friendly manner while maximizing the throughput and the utilization of the cluster.
- Available resources in the Hadoop cluster are shared among multiple organizations/groups who collectively fund the cluster based on their computing needs.
- An organization or groups within the organization can access any excess capacity not being used by other group thus providing elasticity in a cost-effective manner.

### Fair Scheduler

- Fair scheduling is a method of assigning resources to applications such that all apps get an equal share of resources over time.
- Fair Scheduler allows assigning guaranteed minimum shares to queues, which is useful for ensuring that certain users, groups or production applications always get sufficient resources.
- When a queue contains apps, it gets at least its minimum share, but when the queue does not need its full guaranteed share, the excess is split between other running apps.

# Chapter 2 : Hadoop 2

## Hadoop Cluster Modes

### Local standalone mode

In this mode, all the components of Hadoop, such NameNode, DataNode, JobTracker, and TaskTracker, run in a single Java process. This mode is not suited or development as well as production.

### Pseudo-distributed mode

In this mode all hadoop services, including both the master and the slave services, run on a single node. It is useful for to test applications. A separate JVM is spawned for every Hadoop components and they communicate across network sockets, effectively producing a fully functioning and optimized mini-cluster on a single host.

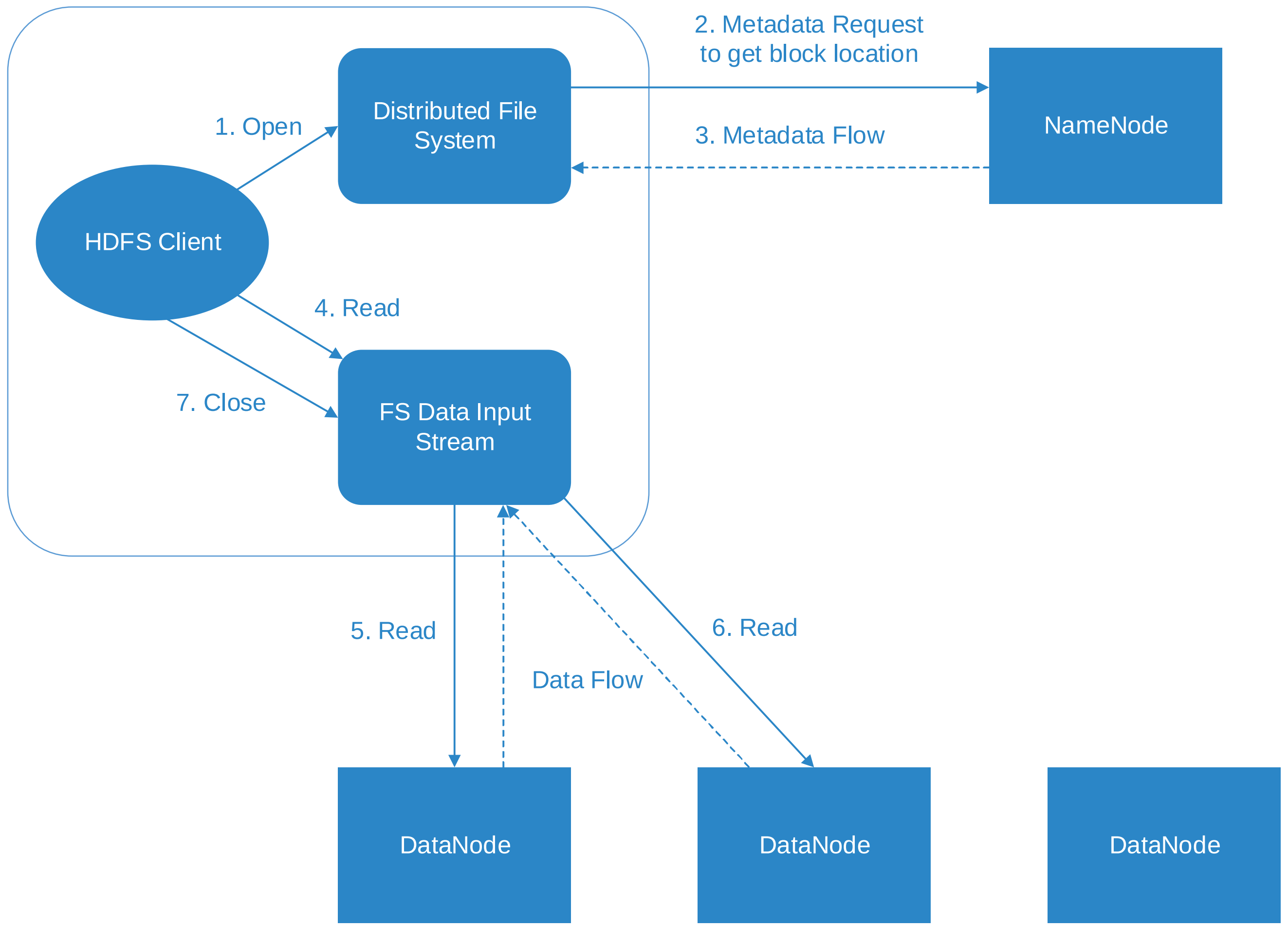
### Fully distributed mode

A Hadoop deployment in which the Hadoop master and slave services run on a separate nodes. It is used for production and development environments. Generally all the master and the slave services run on different machine. In this mode we realize the full potential of distributed computing.

# Chapter 2 : The HDFS Read process

- A client initiates read request by calling 'open()' method of DistributedFileSystem object.
- This object connects to NameNode using RPC and gets metadata information such as the locations of the blocks of the file.
- In response to this metadata request, addresses of the DataNodes having a copy of that block is returned back by the NameNode.
- Once addresses of DataNodes are received, an object of type FSDataInputStream is returned to the client.
- FSDataInputStream contains DFSInputStream which takes care of interactions with DataNode and NameNode.
- In step 4 shown in the above diagram, a client invokes 'read()' method which causes DFSInputStream to establish a connection with the first DataNode with the first block of a file.
- Data is read in the form of streams and the process of read() operation continues till it reaches the end of block.
- Once the end of a block is reached, DFSInputStream closes the connection and moves on to locate the next DataNode for the next block.
- Once a client has done with the reading, it calls a close() method.

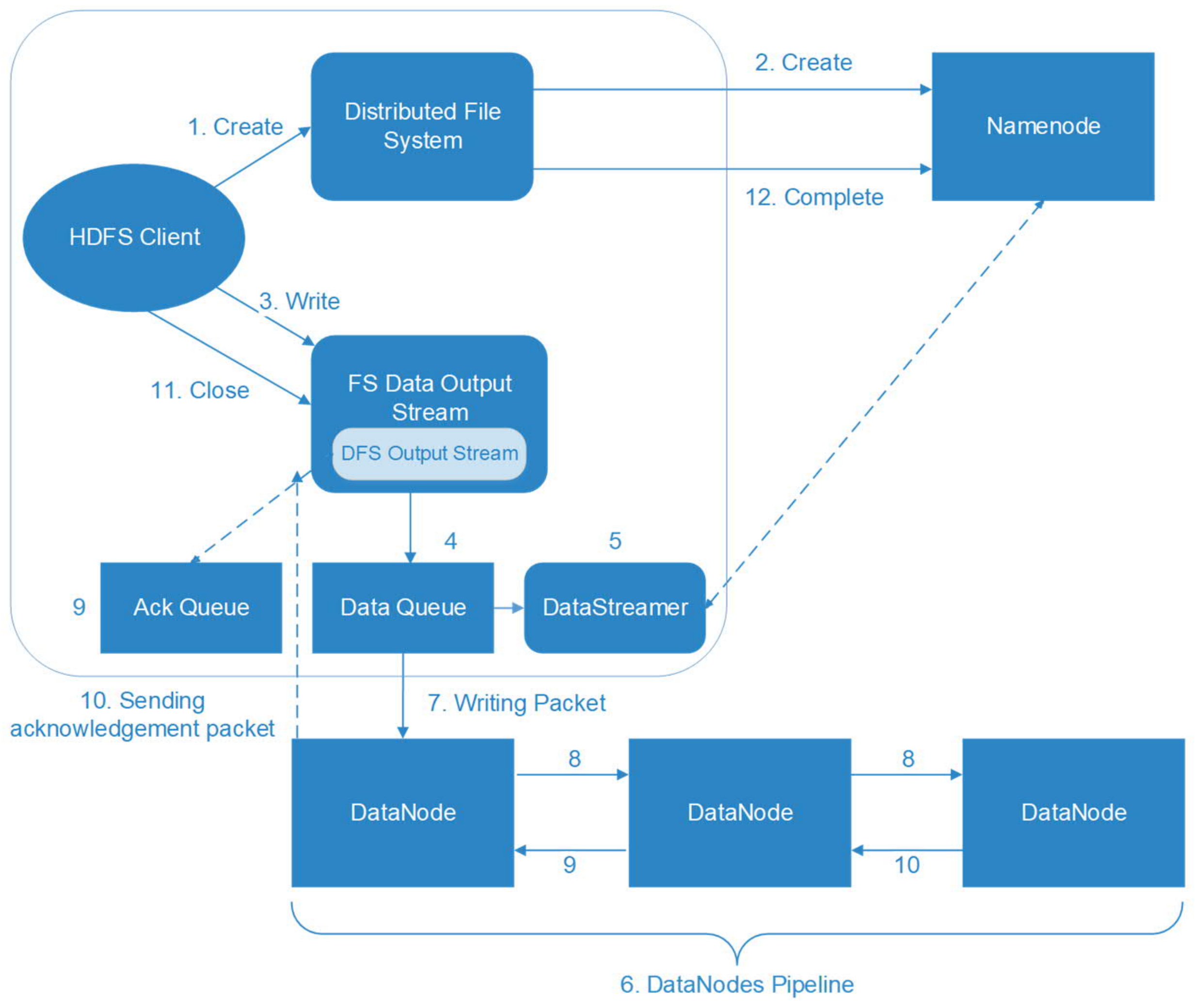
# Chapter 2 : The HDFS Read process



# Chapter 2 : The HDFS Write process

- A client initiates write operation by calling 'create()' method of DistributedFileSystem object which creates a new file.
- DistributedFileSystem object connects to the NameNode using RPC call and initiates new file creation.
- NameNode verifies that the file does not exist already and a client has correct permissions to create a new file.
- If a file already exists or client does not have sufficient permission to create a new file, then IOException is thrown to the client.
- Once a new record in NameNode is created, an object of type FSDataOutputStream is returned to the client. A client uses it to write data into the HDFS. Data write method is invoked
- FSDataOutputStream contains DFSOutputStream object which looks after communication with DataNodes and NameNode. While the client continues writing data, DFSOutputStream continues creating packets with this data. These packets are enqueued into a queue which is called as DataQueue.
- There is one more component called DataStreamer which consumes this DataQueue. DataStreamer also asks NameNode for allocation of new blocks thereby picking desirable DataNodes to be used for replication.
- The process of replication starts by creating a pipeline using DataNodes.
- The default replication number is 3 and hence data is copied across 3 DataNodes.

# Chapter 2 : The HDFS Write process



# Chapter 2 : The HDFS Write process

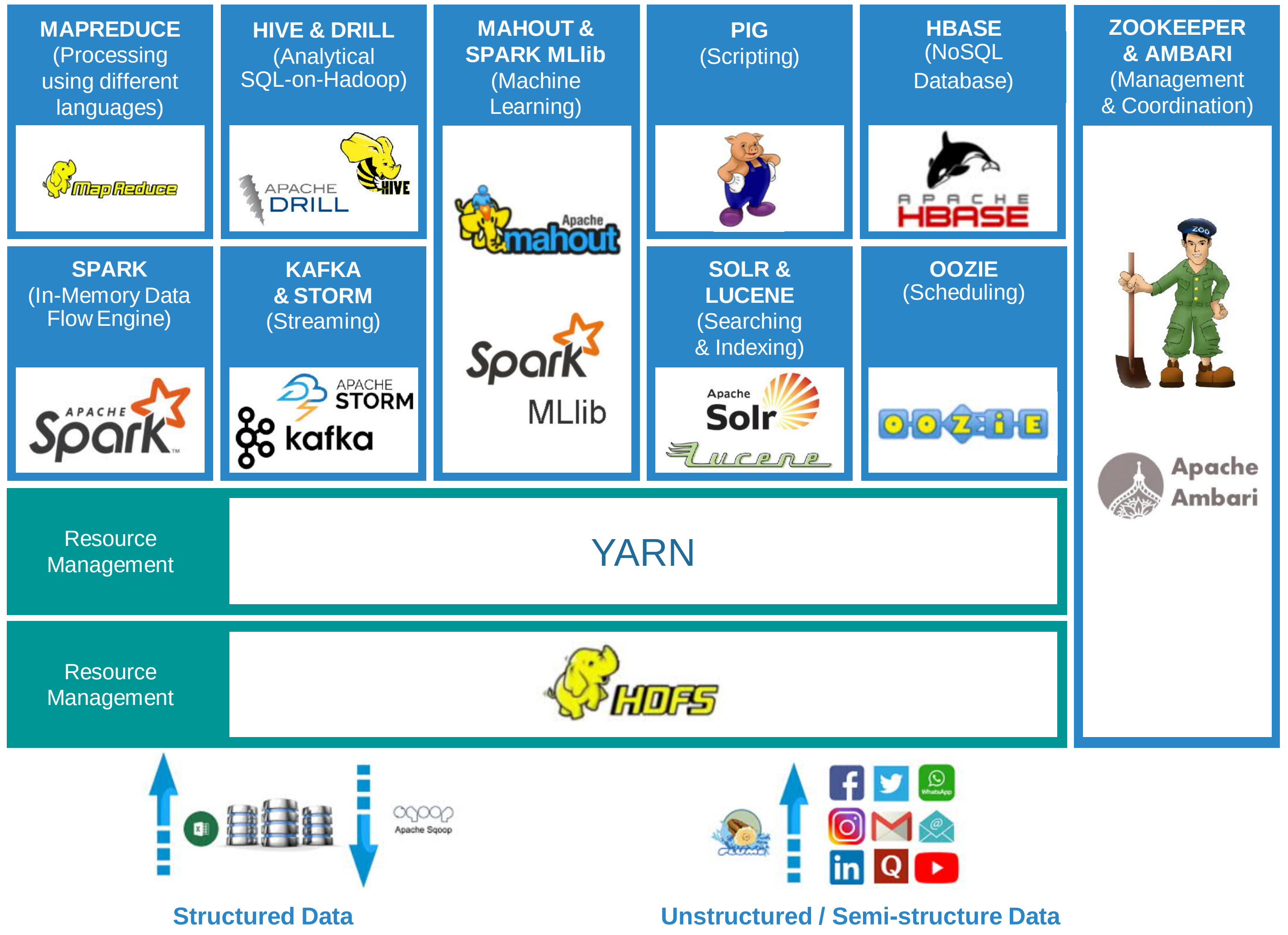
- The DataStreamer copies the packets into the first DataNode in the pipeline.
- Every DataNode in a pipeline stores packet received by it and forwards the same to the second DataNode in a pipeline.
- 'Ack Queue' is maintained by DFSOutputStream to store packets which are waiting for acknowledgment from DataNodes.
- Once acknowledgment for a packet in the queue is received from all DataNodes in the pipeline, it is removed from the 'Ack Queue'.
- In the event of any DataNode failure, packets from this queue are used to reinitiate the operation.
- After a client is done with the writing data, it calls a close() method and once acknowledgment is received, NameNode is contacted to inform that the file write operation is complete.

3

## Big Data Ecosystem

# Chapter 3 : Contents

# Chapter 3 : Overview Of Big Data Ecosystem


MAPREDUCE (Processing using different languages)
MapReduce
HIVE & DRILL (Analytical SQL-on-Hadoop)
APACHE DRILL
HIVE
MAHOUT & SPARK MLlib (Machine Learning)
Apache mahout
Spark MLlib
PIG (Scripting)
HBASE (NoSQL Database)
APACHE HBASE
ZOOKEEPER & AMBARI (Management & Coordination)
ZOO
Apache Ambari
SPARK (In-Memory Data Flow Engine)
APACHE Spark
KAFKA & STORM (Streaming)
APACHE STORM
kafka
SOLR & LUCENE (Searching & Indexing)
Apache Solr
Lucene
OOZIE (Scheduling)
OOZIE
Resource Management
YARN
Resource Management
HDFS
Apache Sqoop
Structured Data
Unstructured / Semi-structure Data

# Chapter 3 : Overview Of Big Data Ecosystem

## Hive

Apache Hive is a data warehouse software project built on top of Apache Hadoop for providing data query and analysis. Hive gives a SQL-like interface to query data stored in the file systems.

## Sqoop

Sqoop is a command-line interface application for transferring data between relational databases and Hadoop.

## Solr

Solr is a full text search engine that is built on Apache Lucene. HDFS stores large amount of data and Solr helps us in finding the required information from such a large source.

## Spark

Spark is a distributed data processing framework which is highly scalable and has in-memory capabilities. Spark can process data which is stored in HDFS.

## Pig

It is a data flow language which works on MapReduce paradigm and abstracts all the complexities of writing a MapReduce job.

# Chapter 3 : Overview Of Big Data Ecosystem

## Zookeeper

It is a service for distributed systems offering a hierarchical key-value store, which is used to provide distributed configuration, synchronization  and naming registry for large distributed systems.

## Kafka

Kafka provides a unified, high-throughput, low-latency platform for handling real-time data feeds.

## HBase

Apache HBase is a non-relational database modeled after Google's Bigtable used to have random, real-time read/write access to HDFS.

## Ambari

Ambari enables system administrators to provision, manage and monitor a Hadoop cluster.

## Zeppelin Notebook

It is an interactive browser-based notebook which enables data engineers, data analysts and data scientists to be more productive by enabling data-driven interactive analytics.

# Chapter 3 : Overview Of Big Data Ecosystem

## Flume

Apache Flume is a distributed, reliable system for efficiently collecting, aggregating, and moving large amounts of log data to Hadoop.

## MLlib

MLlib is Spark's machine learning library to make machine learning scalable and easy. It provides machine learning Algorithms such as classification, regression, clustering etc.

## Drill

Apache Drill is an open-source software framework that supports data-intensive distributed applications for interactive analysis of large-scale datasets.

## Tajo

Apache Tajo is a robust relational and distributed data warehouse system for Apache Hadoop. Tajo is designed for low-latency and scalable ad-hoc queries on large-data sets stored on HDFS.

# Chapter 3 : Birth Of Hadoop Distributions

- Apache Hadoop is an open source product  so there is no support available for the same.
- While using the Apache Hadoop, we need to install all the hadoop ecosystem projects individually.
- Anyone using the Apache version of Hadoop, has to address the incompatibilities  between the various components on their own, which can be very painful.
- Cloudera, Hortonworks, MapR are some of the widely used Hadoop distributions which are used. These distributions come bundled with all the components which are required by an enterprise.
- All the hadoop distribution vendors have come up with few components which are specific to their hadoop distribution for example Cloudera has Cloudera search, Impala, Cloudera navigator and Cloudera Manager.
- Hortonworks comes bundled with ambari, tez, ranger and MapR has MapR control system.
- Hadoop Distributions provide professional and operational services in which they help the enterprises to move the hadoop clusters to production quickly, painlessly and with optimal performance.

# Chapter 3 : Birth Of Hadoop Distributions

| Features | **Cloudera** | **Hortonworks** | **MapR** |
|---|---|---|---|
| Founded Year | 2009 | 2011 | 2009 |
| License | Open Source and Licensed | Open source | Licensed |
| GUI | Yes | Yes | Yes |
| Execution Environment | Local or cloud | Local or cloud | Local or cloud |
| Metadata Architecture | Centralized | Centralized | Distributed |
| Replication | Data | Data | Data and Metadata |
| Security | Supports default kerberos based authentication for hadoop services | Supports default kerberos based authentication for hadoop services | Supports default kerberos based authentication for hadoop services |
| Management Tools | Cloudera Manager | Ambari | MapR Control System |
| Cost | Cloudera Standard is free. Cloudera Enterprise version is proprietary. | HDP is completely open source data platform. | MapR Converged Community Edition is a free edition of the MapR. |

# Chapter 3 : An Introduction To Hive

- Hive is designed to enable easy data summarization, ad-hoc querying and analysis of large volumes of data.
- It was initially developed at Facebook. It is now an open source Apache project.
- It enables easy data ETL (**E**xtract, **T**ransform, **L**oad).
- It provides a mechanism to project structure on a variety of data formats.
- Query execution is done through MapReduce jobs. Hive abstracts the complexity of the MapReduce jobs from the user by providing a SQL like language called HiveQL.
- Hive supports user-defined Java/Scala functions, scripts and procedure languages to extend its functionality.
- Matured JDBC and ODBC drivers allow many applications to pull Hive data for seamless reporting.
- Cloudera provides their own flavor of Hive called as Impala, which works on massive parallel processing (MPP).
- Hive brings the benefits of the distributed processing to the section of the users who are not from Java.
- Hive is not designed for online transaction processing. It is best used for traditional data warehousing tasks.

# Chapter 3 : An Introduction To Hive

## RDBMS Vs Hive

| RDBMS | Hive |
|---|---|
| RDBMS is a Database. | HIVE is a Data warehouse. |
| RDBMS supports schema on write. | HIVE supports schema on read. |
| Read and Write Many times. | Write once and Read Many times. |
| Record level Insertion, Updates and deletes is possible. | Record level Insertion, Updates and deletes are not possible. |
| Maximum data size allowed will be 10s of Terabytes. | Maximum data size allowed will be 100s of Petabytes. |
| RDBMS is suited for the dynamic data analysis. | HIVE is suited for the static data analysis. |
| Online Transaction Processing (OLTP). | Online Analytical Processing (OLAP). |

# Chapter 3 : An Introduction To Hive

## Hive Components

### User Interface
It provides an user interface to submit queries and other operations to the system.

### Driver
The Driver receives the queries from the user interface .This component implements the session handles and provides execute and fetch APIs modelled on JDBC/ODBC interfaces.

### Compiler
The compiler parses the query and generates an execution plan with the help of the table metadata looked up from the MetaStore.

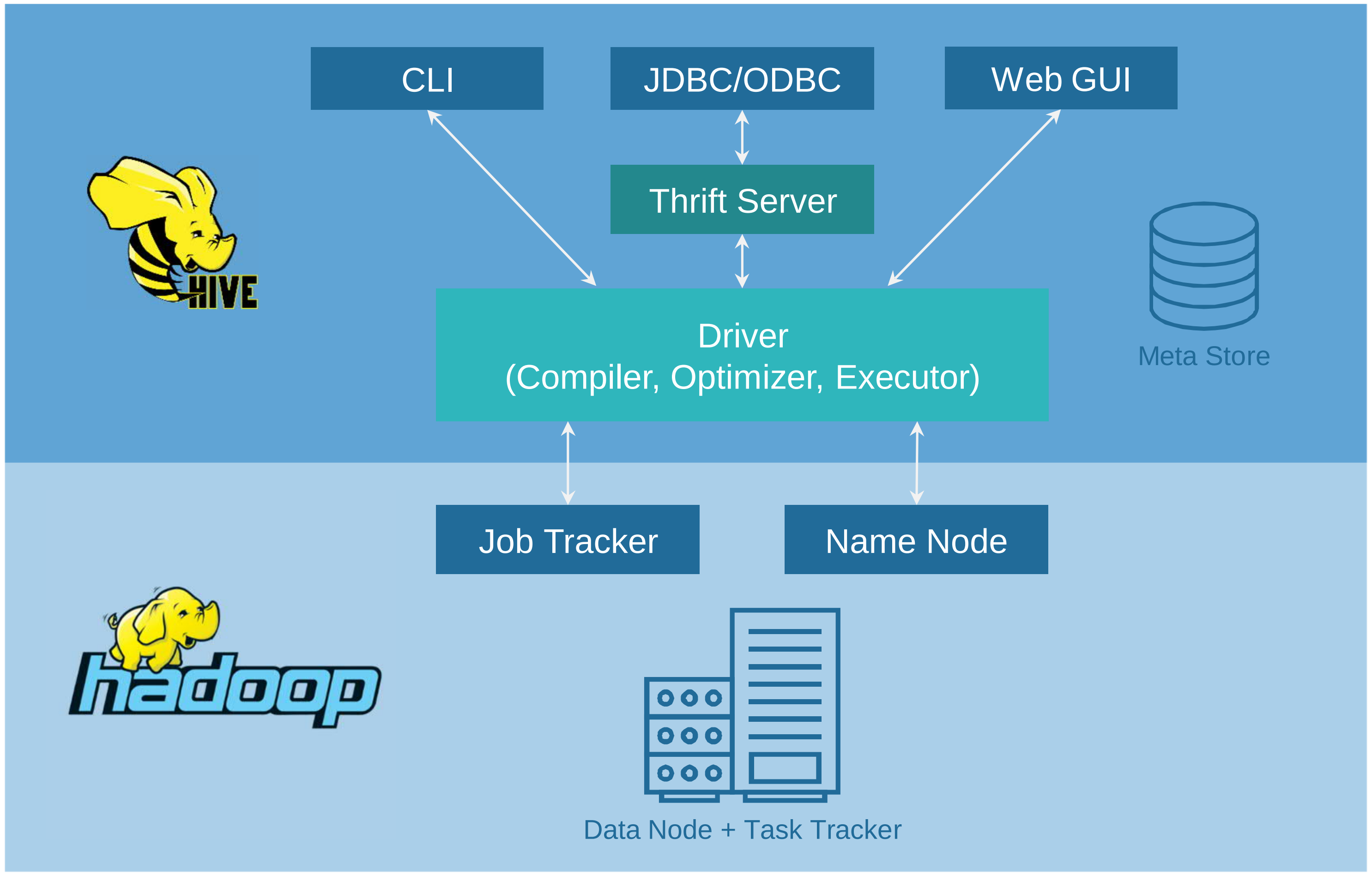
### MetaStore
The component that stores structure information of the tables and partitions in the warehouse. It also stores the column type information, the serializers and de-serializers necessary to read and write data. Metadata is usually stored in a RDBMS like MYSQL or PostgreSQL.

### Execution Engine
The component which executes the execution plan created by the compiler. The execution engine manages the dependencies between these different stages of the plan and executes these stages.

# Chapter 3 : An Introduction To Hive

**Hive**

Components



# Chapter 3 : An Introduction To Hive

## Managed Tables In Hive

- A managed table is stored under the hive.metastore.warehouse.dir path property, by default in a folder path /user/hive/warehouse/databasename.db/tablename/.
- The default location can be overridden by the LOCATION property during table creation.
- If a managed table or partition is dropped, the data and metadata associated with that table or partition are deleted.
- If the PURGE option is not specified, the data is moved to a trash folder for a defined duration.
- Managed tables should be used when Hive should manage the lifecycle of the table.
- A managed table can be identified using the DESCRIBE FORMATTED table_name command.
- The DDL to create an EXTERNAL table is as below :

```
create table
employee_ext(Name String, Sal Int)
row format delimited fields
terminated by ','
LOCATION '/employee';
```

# Chapter 3 : An Introduction To Hive

## External Tables In Hive

- An external table describes the metadata on external files.
- The external table has a EXTERNAL keyword in the DDL.
- External table files can be accessed and managed by processes outside of Hive.
- External tables are used when data is already present in a file or a directory and the data should remain even if the table is dropped.
- External table can be identified using the DESCRIBE FORMATTED table_name command
- The DDL to create an EXTERNAL table is as below :

```
create external table
employee_ext(Name String, Sal Int)
row format delimited fields
terminated by ','
LOCATION '/employee';
```

# Chapter 3 : An Introduction To Hive

## Hive Partitioning

- Partitioning is a way of dividing a table into related parts based on the values of particular columns like date, city, and department.
- Each table in hive can have one or more partition keys to identify a particular partition.
- Using partition it is easy to do queries on slices of the data.
- It helps to avoid full table scans and thus leads to improved performance.
- Too many partitions on a table should be avoided since there is a limit on creating number of partitions on a table. By default the limit is set to 1000.
- There is limit for number of partitions per node which is 100 by default.
- A corresponding directory is created for each partitioned column in hive. Partitioned column should be chosen after analyzing the data.
- NameNode stores the metadata for each directory created because of the partition thus storing the metadata for the directory can become an overhead.

# Chapter 3 : An Introduction To Hive

## Static and Dynamic Partitioning In Hive

### Static Partitioning

- Input data is inserted individually for each partition
- While loading big files into Hive tables static partitions are preferred.
- It is possible add a partition in the table and move the input file into the partition of the table.
- We can alter the partition in the static partition.
- To use static partition in the hive property set hive.mapred.mode=strict should be set. This property is set by default.

### Dynamic Partitioning

- Single insert statement to partition table is known as a dynamic partition.
- Dynamic Partition takes more time in loading data compared to static partition, since it loads data for all the partitions automatically
- We cannot alter a Dynamic partition
- To use dynamic partitioning in hive property set hive.mapred.mode=nonstrict should be set. This property is set to strict by default.

# Chapter 3 : An Introduction To Hive

## Static and Dynamic Partitioning In Hive

### Static Partitioning

```
INSERT INTO TABLE emp_inf_partition PARTITION(department='A')
SELECT employeeid,firstname,designation,salary,country,state
FROM emp_stg WHERE department='A';

INSERT INTO TABLE emp_inf_partition PARTITION(department='B')
SELECT employeeid,firstname,designation,salary,country,state
FROM emp_stg WHERE department='B';
```

We have to insert the data individually for each partition.

### Dynamic Partitioning

Set the below properties :

```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;
set hive.exec.max.dynamic.partitions=1000;
set hive.exec.max.dynamic.partitions.pernode=100;
```

Insert the data into the table using the below statement :

```
INSERT INTO TABLE emp_inf_partition PARTITION(department)
SELECT employeeid,firstname,designation,salary,country,state,department FROM
emp_stg;
```

We have only one statement irrespective of the number of departments in the final table.

# Chapter 3 : An Introduction To Hive

## Bucketing In Hive

- Bucketed tables allow much more efficient sampling than the non-bucketed tables.
- The bucketed tables are created by using CLUSTERED BY clause in the DDL.
- Bucketed tables allow mapside joins.
- It helps in addressing the data skew problems caused by partitioning a table.
- A file is created for each bucket thus relatively NameNode has to store less metadata.

- A bucketed table is created as below :

```
  CREATE TABLE bucketed_table(
  firstname VARCHAR(64),
  lastname  VARCHACREATE TABLE bucketed_user(
firstname VARCHAR(64),
  lastname  VARCHAR(64),
  address   STRING,
  city VARCHAR(64),
state  VARCHAR(64),
  post      STRING,
  phone1    VARCHAR(64),
  phone2    STRING,
  email     STRING,
  web       STRING
  )
COMMENT ‘A bucketed sorted user table’
  PARTITIONED BY (country VARCHAR(64))
CLUSTERED BY (state) SORTED BY (city) INTO 32 BUCKETS
  STORED AS SEQUENCEFILE;
```

# Chapter 3 : An Introduction To Sqoop

- Sqoop is a tool designed to transfer data between Hadoop and relational databases.
- Sqoop automates most of this process, relying on the database to describe the schema for the data to be imported.
- Sqoop uses MapReduce to import and export the data, which provides parallel operation as well as fault tolerance.
- With the help of Sqoop we can import data to Hive, HBase and HDFS.
- Sqoop identifies the primary key column in table  to be imported. It calculates the  low and high values for the Primary key column from the database, and the map tasks operate on evenly-sized components of the total range.
- By default 4 mappers are initiated by Sqoop.
- In case the table to be imported does not have a primary key, it distributes the work load of the mappers by using the split by splitting column.
- Sqoop provides a command line interface to execute Sqoop jobs.
- We can specify the value of the number of mappers in the Sqoop.
- We can import data to a partitioned table using Sqoop.
- Sqoop supports incremental loads.
- Sqoop is written in Java.

# Chapter 3 : An Introduction To Pig

- Pig is a data flow language.
- Platform for analyzing large sets of data.
- Pig Latin is used to express the queries and data manipulation operations in simple scripts.
- Pig converts the scripts into a sequence of underlying MapReduce jobs.
- Pig is used to process different file formats like CSV file, JSON file, Logs etc.
- Provides standard data processing operations for example LOAD, GENERATE, GROUP etc.
- Insulates Hadoop complexity.
- Abstracts MapReduce by providing  a simple SQL like language called Pig Latin.
- Increases programmer productivity by reducing the development time.
- Significantly reduces the number of line of code. The lines of code in Pig are 1/20th of MapReduce.
- Pig makes joining datasets in Hadoop very easy efficient.

# Chapter 3 : An Introduction To Pig

## What Is Pig Latin?

- Pig Latin is a SQL like data flow language.
- It provides abstraction over the complexity of the MapReduce.
- Under the covers, Pig runs the transformations into a series of Map Reduce jobs
- It reduces development times and is easy to test.
- The table shows the SQL statement and the equivalent Pig Latin for it.

| SQL | Pig Latin |
|---|---|
| SELECT column_name,column_name FROM table_name; | FOREACH alias GENERATE column_name,column_name; |
| SELECT * FROM table_name; | FOREACH alias GENERATE *; |
| SELECT DISTINCT column_name,column_name FROM table_name; | DISTINCT(FOREACH alias GENERATE column_name, column_name); |
| SELECT column_name,column_name FROM table_name WHERE column_name operator value; | FOREACH (FILTER alias BY column_name operator value) GENERATE column_name, column_name; |

# Chapter 3 : An Introduction To Pig

## Different Modes in Pig

### Local Mode

- pig -x local
- In this mode, entire pig job runs in a single jvm process
- Picks and stores data from local Linux path

### MapReduce Mode

- pig -x mapreduce
- In this mode, pig job runs as a series of map reduce job
- Input and output paths are assumed as HDFS paths

### Tez Mode

- pig -x tez
- In this mode, tez job runs instead of MapReduce
- TEZ is specific to Hortonworks data platform

# Chapter 3 : An Introduction To Pig

## Load and Storage Functions in Pig

| Functions | Description |
|---|---|
| PigStorage | Loads and stores data as structured text files |
| TextLoader | Loads unstructured data in UTF-8 format |
| HBaseStorage | Loads and stores data from an HBase table |
| AvroStorage | Loads and stores data from Avro files |
| OrcStorage | Loads from or stores data to ORC file |
| JsonLoader | It is used to Load Json data |
| JsonStorage | It is used to Store JSON data |
| BinStorage | Loads and stores data in machine-readable format |
| PigDump | Stores data in UTF-8 format |
| STORE | Used to save results in HDFS |

# Chapter 3 : An Introduction To Pig

## Pig Sample Code

```
users = load 'users.csv' as (username:chararray, age:int);
users_18_25= filters users by age>=18 and age <=25;
pages= load 'pages.csv' as (username: chararray, url: chararray);
joined = join users_18_25 by username, pages by username;
grouped = group joined by url;
summed = foreach grouped generate group as url,COUNT(joined) AS views;
sorted = order summed by views desc;
top_5 = limit 5;
store top_5 into 'top_5_sites.csc';
```

# Chapter 3 : An Introduction To Pig

## Where Not To Use Pig?

- Input data format is really nasty for example video, audio files etc.
- We need more fine grained control on processing.
- Pig lacks control structures, so loops are not possible and if/else or case statements are difficult.
- Whenever we write code in pig at back end compilation time , this code is going to convert into map reduce programs.

# Chapter 3 : An Introduction To Oozie

- Apache Oozie is a workflow scheduler for Hadoop.
- It runs the workflow of dependent jobs.
- It creates a  Directed Acyclic Graphs of workflows, which can be run in parallel and sequentially in Hadoop.
- Oozie has client API and command line interface which can be used to launch, control and monitor jobs.
- Using Oozie’s Web Service APIs one can control jobs from anywhere.
- Oozie is capable to execute jobs which are scheduled to run periodically.
- Oozie has a provision to send email notifications upon completion or failure of jobs.
- Oozie has two important files the workflow.xml and the coordinator.xml. The workflow.xml contains the various actions which needs to be performed and coordinator.xml has the frequency at which to run the workflow.
- Oozie consists of two parts :
  - **Workflow engine** : Responsibility of a workflow engine is to store and run workflows composed of Hadoop jobs e.g., MapReduce, Pig, Hive.
  - **Coordinator engine** : It runs workflow jobs based on predefined schedules and availability of data.

# Chapter 3 : An Introduction To HBase

- HBase is a scalable, distributed, column-oriented datastore.
- HBase provides real-time read/write random access to very large datasets hosted on HDFS.
- By default HBase comes with standalone mode.In this mode of operation, a single JVM hosts the HBase Master, an HBase RegionServer, and a ZooKeeper quorum peer.
- HBase stores data on the local filesystem, rather than using HDFS in the standalone mode.
- Standalone mode is only appropriate for initial testing.
- Most aspects of HBase are highly available in a standard configuration.
- Hbase cluster typically consists of one Master and three or more RegionServers, with data stored in HDFS.
- To ensure that every component is highly available, one or more backup Masters are configured.
- The backup Masters run on other hosts than the active Master.

# Chapter 3 : An Introduction To HBase

## Comparing HBase and Hive

| Features | HBase | Hive |
|---|---|---|
| Database model | Column store | Relational DBMS |
| Data Schema | Schema-free | With Schema |
| SQL Support | No, can use Phoenix for SQL feel | HQL(Hive query language) |
| Partition methods | Sharding | Sharding |
| Replication Methods | HDFS default replication | HDFS default replication |

# Chapter 3 : An Introduction To HBase

## Comparing HBase and RDBMS

| HBASE | RDBMS |
|---|---|
| Schema-less in database | Having fixed schema in database |
| Column-oriented databases | Row oriented data store |
| Designed to store De-normalized data | Designed to store Normalized data |
| Wide and sparsely populated tables present in HBase | Contains thin tables in database |
| Supports automatic partitioning | Has no built in support for partitioning |
| Well suited for OLAP systems | Well suited for OLTP systems |
| Read only relevant data from database | Retrieve one row at a time and hence could read unnecessary data if only some of the data in a row is required |
| Structured and semi-structure data can be stored and processed using HBase | Structured data can be stored and processed using RDBMS |
| Enables aggregation over many rows and columns | Aggregation is an expensive operation |

# Chapter 4 : Which NoSQL Database to choose?

## Comparing HBase and RDBMS

MongoDB, CouchDB and Cassandra are NoSQL databases that are feature specific and used as per their business needs. Listed below are different NoSQL databases as per their use case.

| Data Base Type Based on Feature | Example of Database | Use case (When to Use) |
|---|---|---|
| Key/ Value | Redis, MemcacheDB | Caching, Queue-ing, Distributing information |
| Column-Oriented | Cassandra, HBase | Scaling, Keeping Unstructured, non-volatile |
| Document-Oriented | MongoDB, Couchbase | Nested Information, JavaScript friendly |
| Graph-Based | OrientDB, Neo4J | Handling Complex relational information. Modeling and Handling classification. |

# Chapter 3 : Limitations Of Hadoop

- Hadoop is a batch processing framework, it is not capable of processing streamed data.
- Hadoop does not leverage the memory of the Hadoop cluster to the maximum.
- Hadoop is not suitable for real time or near real time processing of data.
- Hadoop works on the MapReduce paradigm which is IO intensive.
- Hadoop MapReduce jobs require lot of time to complete thereby increasing latency.
- In Hadoop it is not possible to cache the intermediate output of the MAP jobs.
- In Hadoop, optimizing the MapReduce jobs is complicated.
- Hadoop MapReduce model is complex, there is a need to handle the low-level APIs.

MOTT MACDONALD

4

# An Overview Of Big Data Systems

# Chapter 4 : Contents

# Chapter 4 : An Introduction To Apache Spark

- Apache Spark is a fast and highly distributed processing framework for processing data.
- It provides high-level APIs in Java, Scala, Python and R.
- It also supports a rich set of higher-level tools including Spark SQL for structured data processing, MLlib for machine learning, GraphX for graph processing and Spark Streaming.
- Spark provides in-memory computation thus reduces IO which leads to the faster processing of data.
- It is 100 times faster than MapReduce in memory and 10 times faster on disk.
- Spark has been used to sort 100 TB of data 3 times faster than Hadoop MapReduce on one-tenth of the machines.
- Spark is particularly fast on machine learning applications such as Naive Bayes and k-means.
- It is a low-latency computing framework.

# Chapter 4 : An Introduction To Apache Spark

## Spark Application Lifecycle

- Spark applications run as independent set of processes on a cluster, coordinated by the SparkContext object present in the driver program.
- SparkContext can connect to several types of cluster managers either Spark's own standalone cluster manager, Mesos or YARN which allocate resources across applications.
- Once connected, Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for your application.
- Spark Context sends application code (defined by JAR or Python files passed to SparkContext) to the executors.
- SparkContext sends tasks to the executors to run.
- Each application gets its own executor processes, which stay up for the duration of the whole application and run tasks in multiple threads.
- Spark is agnostic to the underlying cluster manager.
- The driver program must listen for and accept incoming connections from its executors throughout its lifetime.

# Chapter 4 : An Introduction To Apache Spark

## SparkContext

- SparkContext is the entry gate of Apache Spark functionality.
- The most important step of any Spark driver application is to generate SparkContext.
- It allows your Spark Application to access the Spark Cluster with the help of Resource Manager (YARN/Mesos).
- To create SparkContext, SparkConf should be created.
- The SparkConf has a configuration parameter that our Spark driver application will pass to SparkContext.
- To create a SparkContext we need to build a SparkConf object that contains information about your application.

```
val conf = new SparkConf().setAppName(appName).setMaster(master)

new SparkContext(conf)
```

- The appName parameter is a name for your application to show on the cluster UI,master is a Spark, Mesos or YARN cluster URL, or a  “local” string to run in local mode.

# Chapter 4 : An Introduction To Apache Spark

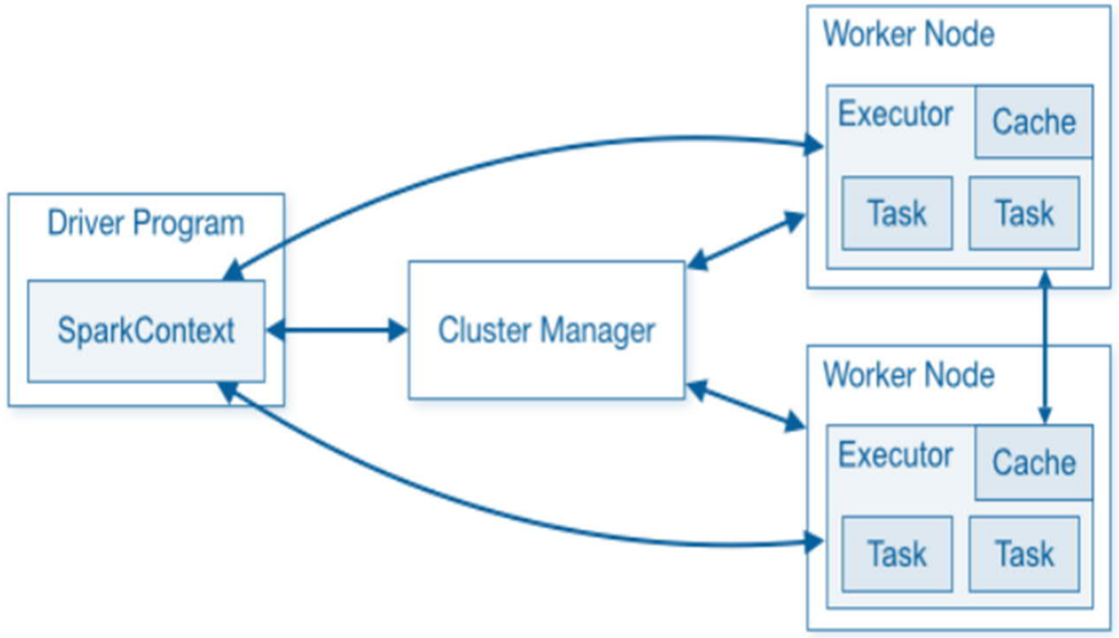
## Cluster Managers In Spark

### Standalone
A simple cluster manager included with Spark that makes it easy to set up a cluster.

### Apache Mesos
A general cluster manager that can also run Hadoop MapReduce and service applications.

### Hadoop YARN
The resource manager in Hadoop 2.

### Kubernetes
An open-source system for automating deployment, scaling and management of containerized applications. It is currently experimental(spark 2.4.5).



# Chapter 4 : An Introduction To Spark Resilient Distributed Dataset (RDD)

- RDDs are the main logical data units in Spark.
- They are a distributed collection of objects, which are stored in a distributed manner in the cluster.
- RDDs are divided into multiple logical partitions so that these partitions can be stored and processed on different machines of a cluster.
- RDDs are immutable (read-only) in nature.
- RDD cannot be changed, but can be transformed into another RDD using transformations.
- RDD in Spark can be cached and used again for future transformations.
- RDDs are lazily evaluated. This saves a lot of time and improves efficiency.
- RDDs track data lineage information to recover lost data, automatically on failure.
- A transformation on RDD takes one RDD as the input and produces another RDD as the output.
- An action takes an RDD as the input and returns the final result to the driver.

# Chapter 4 : An Introduction To Spark Resilient Distributed Dataset (RDD)

## Transformations Using RDDs

| Functions | Description |
|---|---|
| map() | Returns a new RDD by applying the function on each data element |
| filter() | Returns a new RDD formed by selecting those elements of the source on which the function returns true |
| reduceByKey() | Aggregates the values of a key using a function |
| groupByKey() | Converts a (key, value) pair into a (key, <iterable value>) pair |
| union() | Returns a new RDD that contains all elements and arguments from the source RDD |
| intersection() | Returns a new RDD that contains an intersection of the elements in the datasets |

# Chapter 4 : An Introduction To Spark Resilient Distributed Dataset (RDD)

## Narrow and Wide transformations

### Narrow transformation

- Each input partition contributes to only one output partition.
- A limited subset of partition is used to calculate the result. Narrow transformations are the result of :
  - map()
  - filter()

### Wide transformation

- In Input partitions is contributing to many output partitions
- These transformations cause shuffle where Spark exchanges partitions across the cluster. Wide transformations are the result of the below transformations :
  - groupbyKey()
  - reducebyKey()

# Chapter 4 : An Introduction To Spark Resilient Distributed Dataset (RDD)

## Actions Using RDDs

| Functions | Description |
|---|---|
| count() | Gets the number of data elements in an RDD |
| collect() | Gets all the data elements in an RDD as an array |
| reduce() | Aggregates data elements into an RDD by taking two arguments and returning one |
| take(n) | Fetches the first n elements of an RDD |
| foreach(operation) | Executes the operation for each data element in an RDD |
| first() | Retrieves the first data element of an RDD |

# Chapter 4 : An Introduction To Spark Resilient Distributed Dataset (RDD)

## Ways To Create Spark RDD

There are 3 ways to create Spark RDDs

### Parallelized collections

By invoking parallelize method in the driver program. It takes the existing collection in the program and passes it to the SparkContext parallelize() method.

```
val data = spark.sparkContext.parallelize(Seq(("maths",52),("english",75),("science",82), ("computer",65),("maths",85)))
val sorted = data.sortByKey()
sorted.foreach(println)
```

### External datasets

By loading a external dataset. It takes URL of the file and reads it as a collection of lines.

```
val dataRDD = spark.read.csv("path/of/csv/file").rdd
val dataRDD = spark.read.json("path/of/json/file").rdd
val dataRDD = spark.read.textFile("path/of/text/file").rdd
```

### Existing RDDs

By applying transformation operation on an existing RDDs.

```
val words = spark.sparkContext.parallelize(Seq("the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"))
val wordPair = words.map(w => (w.charAt(0), w))
wordPair.foreach(println)
```

# Chapter 4 : An Introduction To Spark SQL

- Spark SQL is a Spark module for structured data processing.
- Unlike the basic Spark RDD API, the interfaces provided by Spark SQL provide Spark with more information about the structure of data and the computation being performed.
- Spark SQL uses this extra information to perform extra optimizations.
- There are several ways to interact with Spark SQL including SQL and the Dataset API.
- It is independent of the API/Language being used for computation and uses the same execution engine irrespective of the API/language.
- Spark SQL can also be used to read data from an existing Hive installation.
- Spark SQL overcomes limitations of hive which uses MapReduce jobs for processing the data. It uses the Spark for processing the data.

# Chapter 4 : An Introduction To Spark SQL

## Hive Vs Spark SQL

| Hive | Spark SQL |
|---|---|
| It is a datawarehouse on the file system | It is a library, so integrating it with other libraries in the spark ecosystem is easy |
| Supports only Hive query language (HQL) | Supports both Spark SQL and HQL |
| Metastore has to be created to run hive queries | Metastore is optional and can use Hive Metastore |
| Has its own JDBC server – Hive Thrift Server | It does not have its own JDBC server but uses Hive Thrift Server |
| Users have to explicitly declare the schema in Hive | Automatically infers the schema |

# Chapter 4 : An Introduction To Spark DataFrames

- A DataFrame is an immutable distributed collection of data.
- In a DataFrame, data is organized into named columns, like a table in a relational database.
- It is designed to make it easier to process large data sets.
- A DataFrame allows developers to impose a structure onto a distributed collection of data, allowing higher-level abstraction.
- The DataFrame APIs merge with Datasets APIs, leading to unified data processing capabilities across libraries.
- A DataFrame uses Catalyst to generate an optimized logical and physical query plan.

# Chapter 4 : An Introduction To Spark Datasets

- A Dataset is a data structure in SparkSQL which is strongly typed meaning the developer cannot work around the type system of the language.
- Spark Dataset provides both type safety and object-oriented programming interface.
- Encoders translate between JVM objects and Spark’s internal binary format. Spark has built-in encoders which are highly optimized. They generate bytecode to interact with off-heap data.
- Dataset clubs the features of RDD and DataFrame.
- It provides the convenience of RDD, performs optimizations of DataFrame and provide Static type-safety of Scala.
- Datasets provides the compile-time safety which was not present in DataFrames.
- Dataset in Spark provides optimized query using Catalyst Query Optimizer and Tungsten.
- Spark Datasets are both serializable and queryable, hence can save it to persistent storage.
- While caching, it creates a more optimal layout which leads to less memory consumption.

# Chapter 4 : An Introduction To Spark Datasets

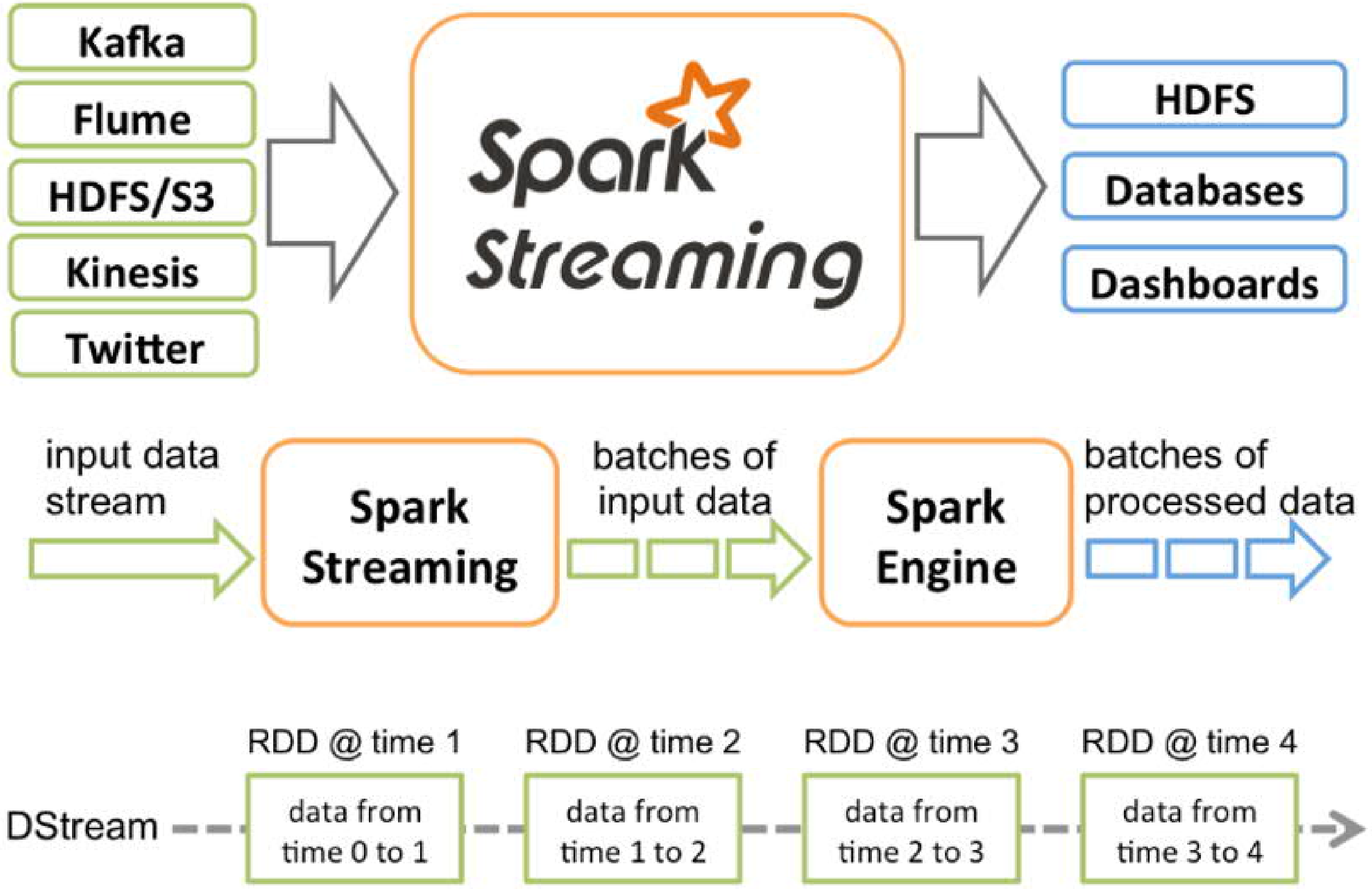
## When to use Spark DataFrames or Datasets?

- Spark DataFrames and Datasets provide rich semantics, high-level abstractions and language specific APIs (i.e. for Java, Python, R or Scala).
- Spark DataFrames or Datasets are used when the data processing demands high-level expressions, filters, maps, aggregation, averages, sum, SQL queries, columnar access and use of lambda functions on semi-structured data.
- Spark Datasets are used when there is a requirement for type-safety at compile time.
- Spark DataFrames and Datasets provide unification and simplification of APIs across Spark Libraries.
- Spark DataFrames are suitable for Python and R users. If there is a need for more control, Python users can use RDDs.

## Chapter 4 : An Introduction To Spark Streaming

- Spark Streaming offers near real-time processing of stream data.
- Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant processing of live data streams.
- Data can be ingested from many sources like Kafka, Flume, Kinesis, or TCP sockets and can be processed using complex algorithms expressed with high-level functions like map, reduce and join.
- Spark Streaming discretizes the streaming data into tiny, sub-second micro-batches. It accepts data in parallel and buffers it in the memory of Spark's workers nodes.
- The processed data can be pushed out to filesystems, databases  and live dashboards.
- We can apply Spark's machine learning and graph processing algorithms on the data streams.
- Spark Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches.
- DStream which are continuous streams of data can be created either from input data streams from sources such as Kafka, Flume, and Kinesis or by applying high-level operations on other DStreams.
- DStream also called as discretized stream  is represented as a sequence of RDDs internally.

# Chapter 4 : An Introduction To Spark Streaming



# Chapter 4 : An Introduction To Spark Streaming

## Discretized Streams in Spark (DStreams)

- Discretized Stream or DStream is the basic abstraction provided by Spark Streaming.
- It represents a continuous stream of data, either the input data stream received from source or the processed data stream generated by transforming the input stream.
- DStream is represented by a continuous series of RDDs internally.
- DStream can be created either from input data streams from sources such as Kafka, Flume, and Kinesis or by applying high-level operations on other DStreams.
- All operations on DStream translates to operations on the underlying RDDs.
- Underlying RDD transformations are computed by the Spark engine.
- The DStream operations hide most of these details and provide the developer with a higher-level API for convenience.

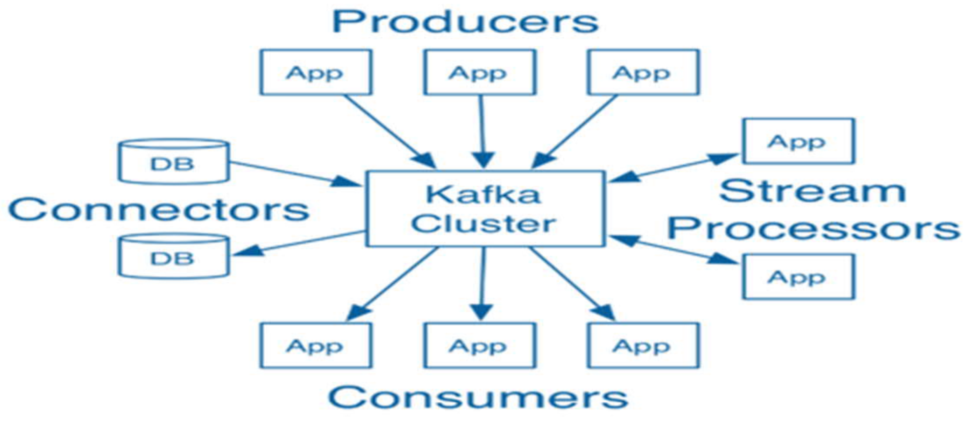
# Chapter 4 : An Introduction To Kafka

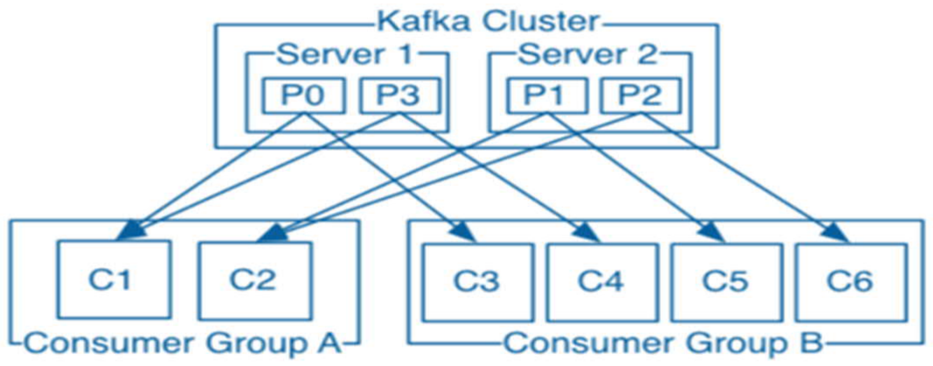
Apache Kafka is a distributed streaming platform. It has three key capabilities :

- Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system.
- Store streams of records in a fault-tolerant manner.
- Process streams of records as they occur.

Kafka is generally used for two broad classes of applications :

- Building real-time streaming data pipelines that reliably get data between systems or applications
- Building real-time streaming applications that transform or react to the streams of data





# Chapter 4 : An Introduction To Kafka

## Core APIs In Kafka

- The Producer API allows an application to publish a stream of records to one or more Kafka topics.
- The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them.
- The Streams API allows an application to act as a stream processor, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
- The Connector API allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

# Chapter 4 : An Introduction To Kafka

## Topics In Kafka

- A topic is a category or feed name to which records are published.
- Topics in Kafka are always multi-subscriber; that is, a topic can have zero, one, or many consumers that subscribe to the data written to it.
- The Kafka cluster maintains a partitioned log for each topic.
- Each partition is an ordered, immutable sequence of records that is continually appended to a structured commit log.
- The records in the partitions are each assigned a sequential id number called the offset that uniquely identifies each record within the partition.
- The Kafka cluster durably persists all published records irrespective of whether they are getting consumed or not till a configurable retention period.

# Chapter 4 : An Introduction To Kafka

## Producers And Consumers In Kafka

- Producers publish data to the topics of their choice.
- The producer is responsible for choosing which record to assign to which partition within the topic.
- This assignment can be done in a round-robin fashion to balance load or it can be done according to some semantic partition function.
- Consumers label themselves with a consumer group name, and each record published to a topic is delivered to one consumer instance within each subscribing consumer group.
- Consumer instances can be in separate processes or on separate machines.
- If all the consumer instances have the same consumer group, then the records will effectively be load balanced over the consumer instances.
- If all the consumer instances have different consumer groups, then each record will be broadcast to all the consumer processes.

# Chapter 4 : An Introduction To Kafka

## Kafka As A Messaging System

- Messaging traditionally has two models, queuing and publish-subscribe.
- In a queue, a pool of consumers may read from a server and each record goes to one of them.
- In publish-subscribe the record is broadcasted to all consumers.
- Queuing allows to divide up the processing of data over multiple consumer instances, which helps to scale processing.
- Queues aren't multi-subscriber, once one process reads the data it’s lost.
- Publish-subscribe allows to broadcast data to multiple processes making it difficult to scale processing since every message goes to every subscriber.

# Chapter 4 : An Introduction To Kafka

## Kafka As A Storage System

- Data written to Kafka is written to disk and replicated for fault-tolerance.
- Kafka allows producers to wait on acknowledgement, a write isn't considered complete until it is fully replicated and guaranteed to persist even if the server written to fails.
- The disk structures Kafka used by Kafka scale well.
- The write process in Kafka remains same irrespective of file size.
- Kafka delivers high-performance, low-latency commit log storage, replication and propagation.

# Chapter 4 : An Introduction To Kafka

## Kafka For Stream Processing

- Kafka a stream processor takes continual streams of data from input topics, processes this input and produces continual streams of data to output topics.
- It is possible to do simple processing directly using the producer and consumer APIs.
- Complex transformations are carried out by fully integrated Streams API provided by Kafka.
- Streams API allows building applications that compute aggregations of streams or join streams together.
- The streams API builds on the core primitives provided by Kafka.
- Streams API uses the producer and consumer APIs for input and Kafka for stateful storage.

5

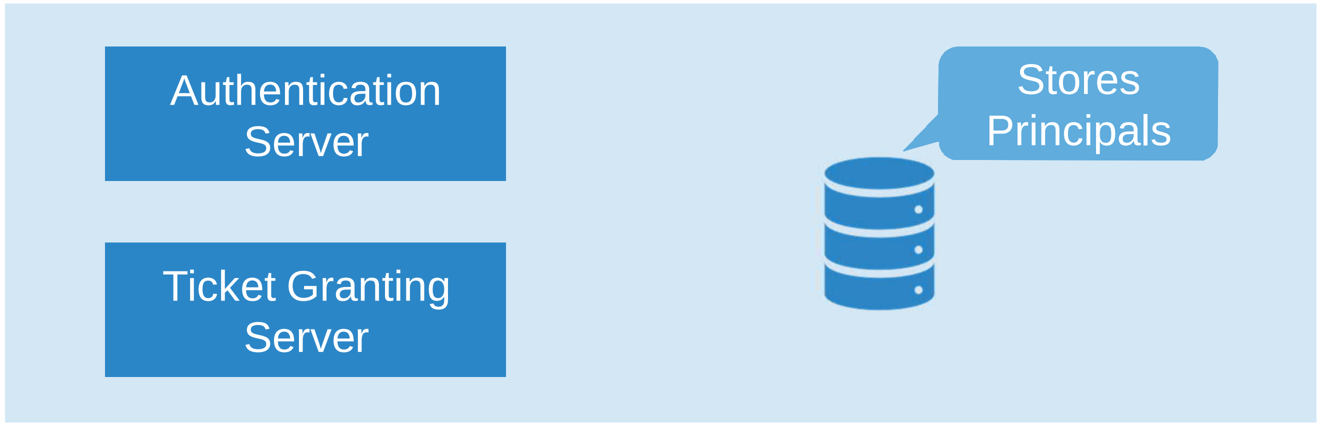
# Security In Big Data Platforms

# Chapter 5 : Contents

# Chapter 5 : Authentication

## Authentication Using Kerberos

- Kerberos is a third party authentication mechanism, in which users and services rely on a third party - the Kerberos server - to authenticate each to the other.
- The Kerberos server itself is known as the Key Distribution Center, or KDC. At a high level, it has three parts :



- A database of the users and services (known as principals) that it knows about and their respective Kerberos passwords.
- An Authentication Server (AS) which performs the initial authentication and issues a Ticket Granting Ticket (TGT).
- A Ticket Granting Server (TGS) that issues subsequent service tickets based on the initial TGT.
- A principal name in a given realm consists of a primary name and an instance name, in this case the instance name is the FQDN of the host that runs that service.

# Chapter 5 : Authentication

## Kerberos Terminology

| Term | Description |
|---|---|
| KDC | The trusted source for authentication in a Kerberos-enabled environment. |
| Kerberos KDC Server | The machine, or server, that serves as the Key Distribution Center (KDC). |
| Kerberos Client | Any machine in the cluster that authenticates against the KDC. |
| Principal | The unique name of a user or service that authenticates against the KDC |
| Keytab | A file that includes one or more principals and their keys. |
| Realm | The Kerberos network that includes a KDC and a number of Clients. |

# Chapter 5 : Authorization

## Hadoop Authorization Using Ranger

- Authorization is the process of ensuring that the rightful user has the appropriate level of access rights/privileges to resources, data, etc.
- Authorization in Hadoop is achieved through Ranger and Sentry, both of which provide enterprise grade security.
- It is a centralized, comprehensive platform for managing authorization, access control, auditing, administration and data protection for data stored in Hadoop.
- Ranger hooks into HDFS, WebHDFS, Hive, HBase and other hadoop services and offers a central authorization provider that each of those projects can use to validate data requests.
- Ranger also provides a comprehensive audit log for viewing requests and their status, as well as a centralized, Web-based administration console for configuring access rights.
- It provides fine-grained authorization for specific actions, operations with a Hadoop component, managed through a central administration tool.
- It provides standardized authorization method across all Hadoop components.
- Enhanced support for different authorization methods like role-based access control and attribute-based access control.
- Ranger helps implement tag-based global policies.
- Centralized auditing of all user access and administrative actions related to security for all components of Hadoop.

# Chapter 5 : Authorization

## Access Control Options Using Ranger

- The Apache Ranger access policy model consists of two major components :
  - Specification of the resources where has to be applied such as HDFS files and directories, Hive databases, tables and columns, HBase tables, column-families etc.
  - The specification of access conditions for specific users and groups.
- Apache Ranger supports the following access conditions :
  - Allow
  - Exclude from Allow
  - Deny
  - Exclude from Deny

# Chapter 5 : Authorization

## Column Masking Using Ranger

- Masking policies are similar to other Ranger access policies for Hive.
- Ranger allows to set filters for specific users, groups, and conditions.
- Column masking prevents sensitive information and helps organizations to meet the compliance requirements.
- Ranger provides the choice of mask which can be applied to a certain column for example: show last 4 characters, show first 4 characters, Hash, Nullify, and date masks (show only year).
- Each column should have its own masking policy.
- An audit log entry is generated each time a masking policy is applied to a column.

# Chapter 5 : Authorization

## Hadoop Authorization Using Sentry

- Sentry is a granular, role-based authorization module for Hadoop.
- Sentry provides the ability to control and enforce precise levels of privileges on data for authenticated users and applications on a Hadoop cluster.
- Sentry currently works with Apache Hive, Hive Metastore, HCatalog, Apache Solr and Impala.
- Sentry is designed to be a pluggable authorization engine for Hadoop components.
- It allows to define authorization rules to validate a user or application's access requests for Hadoop resources.
- Sentry is highly modular and can support authorization for a wide variety of data models in Hadoop.

# Chapter 5 : Data Encryption In Hadoop

## HDFS Encryption

### HDFS Data at Rest Encryption

Configuring and Starting the Ranger KMS →

- Key management
- Access control policies
- Audit

Configuring and Using HDFS “Data at Rest” Encryption →

- Preparing the Environment
- Create an Encryption Key
- Create an Encryption Zone
- Copying Files to or from an Encryption Zone
- Reading and Writing Files from or to an Encryption Zone
- Deleting Files from an Encryption Zone with Trash Enabled
- Create an HDFS Admin User

Configuring HDP Services for HDFS Encryption →

- Configure HBase for HDFS Encryption
- Configuring Hive for HDFS Encryption
- Configure YARN for HDFS Encryption
- Configuring Oozie for HDFS Encryption
- Configuring Sqoop for HDFS Encryption
- Configure WebHDFS for HDFS Encryption

### Encryption in Wire

RPC Encryption → Clients interacting with NameNode

Data Transfer Protocol → Clients interacting with DataNode to Read and Write

HTTPS Encryption → NameNode UI to listen for HTTP

Encryption during Shuffle → Data movement from mappers to reducers

JDBC → Data moving between a HiveServer2 over JDBC and ODBC

# Chapter 5 : Data Encryption In Hadoop

## Key Management Service (KMS) For Data Encryption

- The Ranger Key Management Service is an open source, scalable cryptographic key management service supporting HDFS data at rest encryption.
- Ranger KMS is based on the Hadoop KMS originally developed by the Apache community.
- The Hadoop KMS stores keys in a file-based Java keystore by default.
- Ranger extends the native Hadoop KMS functionality and stores keys in a secure database.
- Ranger provides centralized administration of the key management server through the Ranger admin portal.
- There are three main functions of Ranger KMS are :
    - **Key management** : Ranger admin provides the ability to create, update or delete keys using the Web UI or REST APIs. All Hadoop KMS APIs work with Ranger KMS using the keyadmin username and password.
    - **Access control policies** : Ranger admin also provides the ability to manage access control policies within Ranger KMS. The access policies control permissions to generate or manage keys, adding another layer of security for data encrypted in Hadoop.
    - **Audit** : Ranger provides full audit trace of all actions performed by Ranger KMS.

# Chapter 5 : Data Encryption In Hadoop

## Encryption Of Data At Rest

- HDFS data at rest encryption implements end-to-end encryption of data read from and written to HDFS.
- End-to-end encryption means that data is encrypted and decrypted only by the client.
- HDFS does not have access to unencrypted data or keys.
- HDFS encryption involves several elements.
- It provides protection in addition to the standard HDFS permissions in case of data theft or un-authorized access.
- It is enabled by an key with controlled access. There is a key for each encryption zone.
- Encryptions zones are created in HDFS within which all data is encrypted upon write and decrypted upon read.
- Every file in an encryption zone has a unique encryption key, called the data encryption key (DEK).

# Chapter 5 : Data Encryption In Hadoop

## Encryption In Wire

### RPC Encryption

The most common way for a client to interact with a Hadoop cluster is through RPC. A client  connects to a NameNode over RPC protocol to read or write a file. RPC connections in Hadoop use Java's Simple Authentication & Security Layer (SASL) which supports encryption.

### Data Transfer Protocol

The NameNode gives the client the address of the first DataNode to read or write the block. The actual data transfer between the client and a DataNode is over Data Transfer Protocol. To encrypt data transfer we need to set dfs.encryt.data.transfer=true on NameNode and all DataNode.

### HTTPS Encryption

Encryption over the HTTP protocol is implemented with the support for SSL across a Hadoop cluster.

### Encryption during Shuffle

The data moves between the Mappers and the Reducers over the HTTP protocol, this step is called shuffle. Reducer initiates the connection to the Mapper to ask for data and acts as SSL client.

### JDBC

HiveServer2 implements encryption with Java SASL which helps to encrypt the data moving between a HiveServer2 and a JDBC client.

# Chapter 5 : Data Governance In Hadoop

## Data Governance Using Cloudera Navigator

- Cloudera Navigator is a data governance solution for Hadoop, offering capabilities such as data discovery, continuous optimization, audit, lineage, metadata management and policy enforcement.
- As part of Cloudera Enterprise, Cloudera Navigator enables high-performance agile analytics, supporting continuous data architecture optimization and meeting regulatory compliance requirements.
- It is integrated with leading data preparation, quality and profiling tools.
- Cloudera Navigator's unique policy engine automates critical data stewardship and curation activities.
- It helps to identify and act on opportunities for data model optimization.
- It gives comprehensive visibility into how existing queries, workloads, and schemas are accessed and performing.
- Cloudera Navigator enables users to effortlessly explore and tag data through an intuitive search-based interface.
- It offers integrated backup and disaster recovery.

# Chapter 5 : Data Governance In Hadoop

## Data Governance Using Apache Atlas

- Atlas solves data governance problems across a wide range of industries that use Hadoop. It enables platform-agnostic governance controls that effectively address compliance requirements.
- Atlas is designed to exchange metadata with other tools and processes within and outside of the Hadoop stack.
- Atlas provides technical and operational tracking enriched by business taxonomical metadata.
- Atlas facilitates easy exchange of metadata by enabling any metadata consumer to share a common metadata store that facilitates interoperability across many metadata producers.

Apache Atlas empowers enterprises to effectively and efficiently address their compliance requirements through a scalable set of core governance services. These services include:

- **Data Lineage** : Captures lineage across Hadoop components at platform level.
- **Agile Data Modeling** : Type system allows custom metadata structures in a hierarchy taxonomy.
- **REST API** : Modern, flexible access to Atlas services, HDP components, UI & external tools.
- **Metadata Exchange** : Leverage existing metadata / models by importing it from current tools. Export metadata to downstream systems.

MOTT MACDONALD

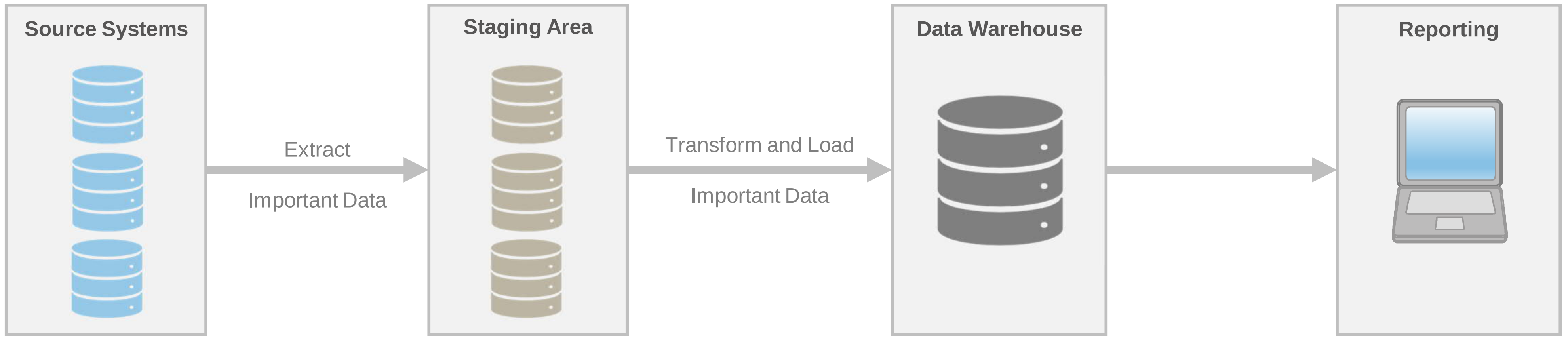
6

## ELT or ETL?

# Chapter 6 : Contents

# Chapter 6 : What is ETL?

- ETL stands for **E**xtract, **T**ransform and **L**oad.
- The ETL process typically extracts data from the source / transactional systems, transforms it to fit the model of data warehouse and finally loads it to the data warehouse.
- The transformation process involves cleansing, enriching and applying transformations to create the desired output.
- Data is usually dumped to a staging area after extraction.



# Chapter 6 : Advantages and Disadvantages Of ETL

## Advantages

**Ease of development** : The process usually involves loading just the relevant data, it reduces complexity and time involved in development.

**Process maturity** : The ETL process is quite mature with multiple production implementations and well defined best practices and processes.

**Tools availability** : A number of tools are available for ETL. This provides flexibility in choosing the most appropriate tool.

**Availability of expertise** : The decades of existence and extensive adoption of ETL process across the board have ensured abundant availability of ETL experts.

## Disadvantages

**Flexibility** : Any changes on the source system would involve updating and re-engineering the entire ETL routine. This adds to time and cost involved in development and maintenance of ETL process.

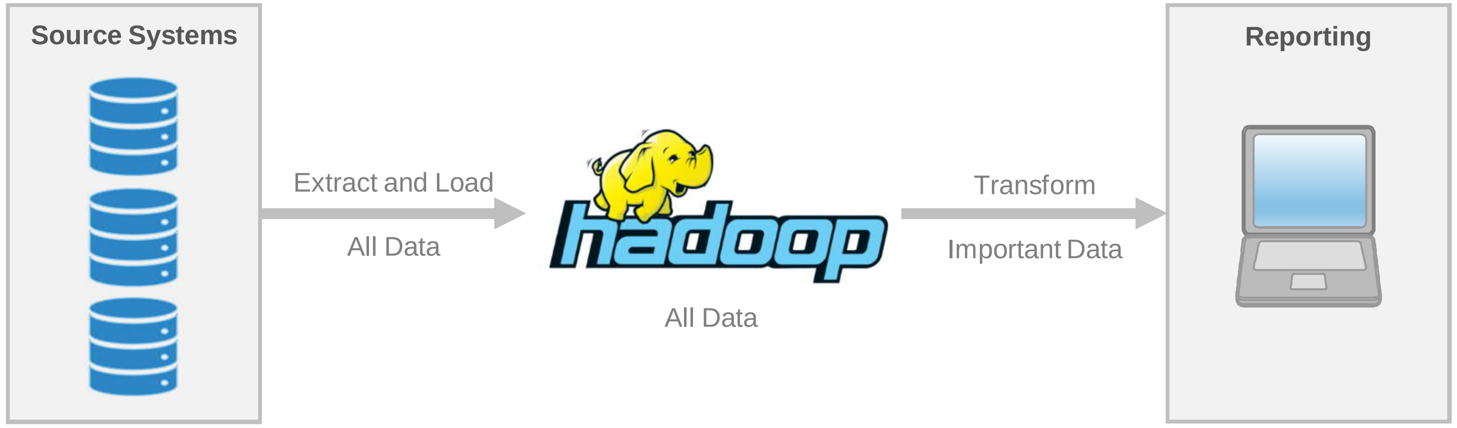
**Hardware** : Most ETL tools come with their own hardware requirements. They have proprietary execution engines which do not use the existing data warehouse hardware. This leads to additional costs.

**Cost** : The maintenance, hardware and licensing costs of the ETL tools add up to the total cost of operating and maintaining the ETL process.

**Limited to relational data** : They are unable to process semi-structured and unstructured data like social media feeds, log files, etc.

# Chapter 6 : What is ELT?

- ELT stands for **E**xtract, **L**oad and **T**ransform.
- ELT process loads the entire data into the data lake.
- Load process can also perform some basic validations and data cleansing rules.
- The data is then transformed for analytical reporting as per demand.



# Chapter 6 : Advantages and Disadvantages Of ELT

## Advantages

**Separation of concerns** : The ELT process separates the loading and transformation tasks into independent blocks and thereby minimizes the interdependencies.

**Flexible and future-proof** : In ELT implementation, entire data from the source systems is already available in the data lake which ensures that future requirements can easily be incorporated into the warehouse structure.

**Utilizes existing hardware** : Hadoop uses the same hardware for storage as well as for processing. This helps in cutting down additional hardware cost.

**Cost effective** : Open source Hadoop framework cuts considerable cost of operating and maintaining the ELT process.

**Not limited to relational data** : With Hadoop, the ELT processes can process semi-structured and unstructured data.

## Disadvantages

**Process maturity** : Though the ELT process has been there for a while, it has not been widely adopted.

**Tools availability** : As a result of limited adoption, the number of tools available to implement ELT processes on Hadoop is currently limited.

**Availability of expertise** : The limited adoption of ELT technology again has an impact on the availability of experts on ELT. The experts for ELT on Hadoop are currently scarce.

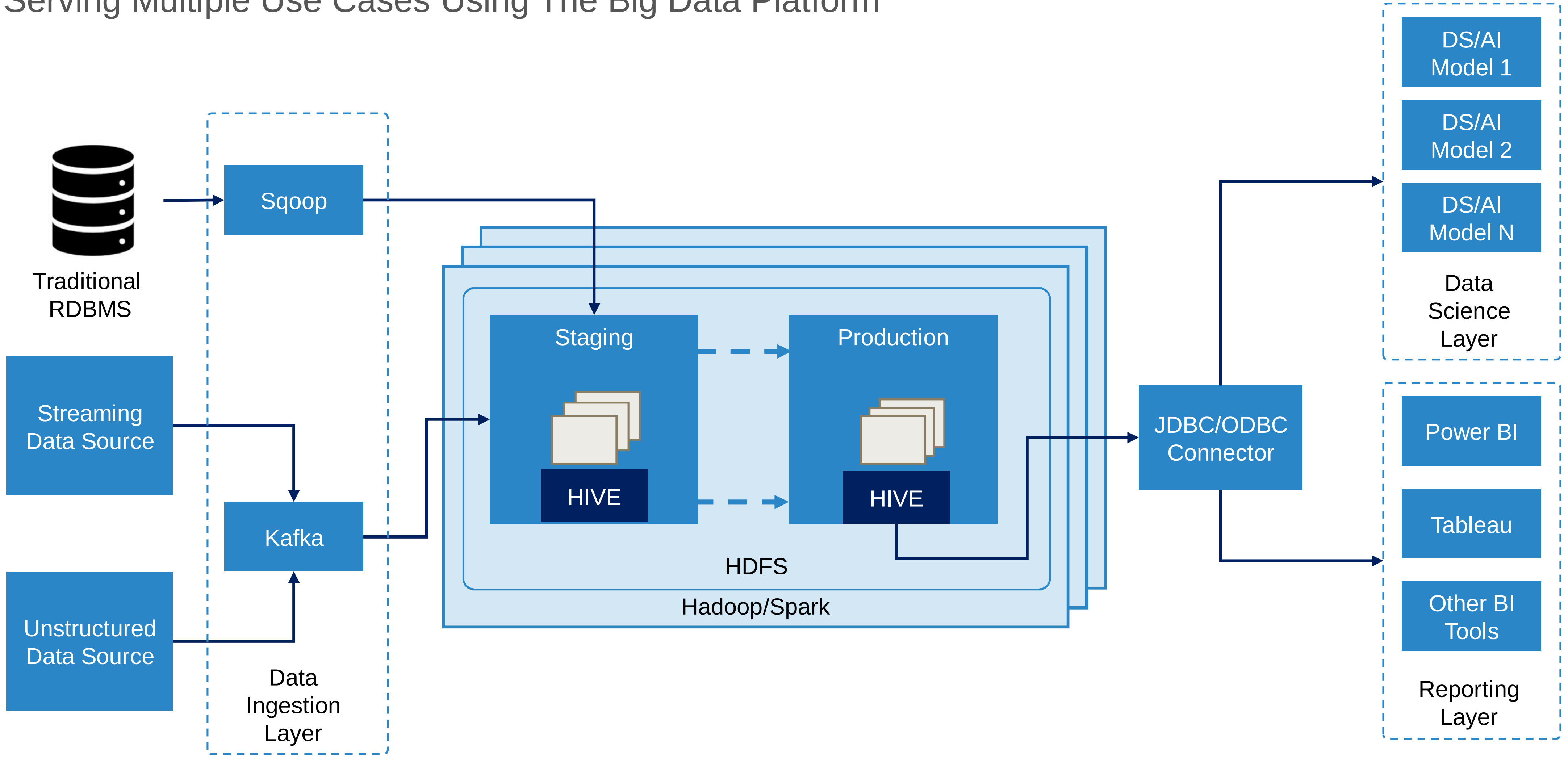
7

## Applications Of Big Data

# Chapter 7 : Contents

# Chapter 7 : An Overview

## Serving Multiple Use Cases Using The Big Data Platform



# Chapter 7 : Consumption By Reporting Layer

## Reporting Tools Consuming High Velocity, High Volume Data

- Apache Hive is a data warehouse built on top of Apache Hadoop for providing data query and analysis.
- Hive provides a SQL-like interface to query data stored in HDFS.
- HiveServer2 has many clients and can be easily accessed using JDBC, Python or Ruby clients. This makes Hive compatible with most reporting tools such as Tableau, Microsoft Power BI, Qlik, Zeppelin etc.
- There are in-memory tools such as Presto and Impala which are used for quick querying of data present in Hive tables.
- HiveServer2 is easily scalable and can handle thousands of concurrent queries if configured properly.
- The data consumption layer is agnostic of the data volume or locality, thus allowing for concurrent consumption of data spread across multiple nodes.

# Chapter 7 : Consumption By Reporting Layer

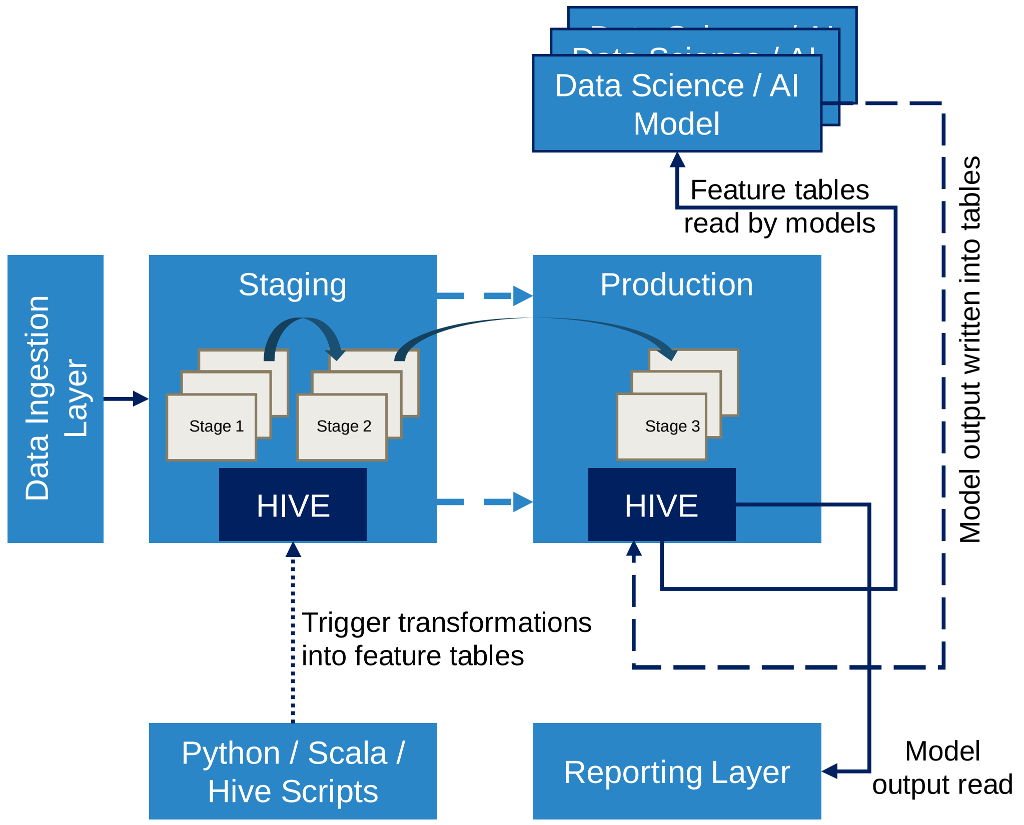
## Simplified Data Consumption Through Interactive Tools : Apache Zeppelin

- Apache Zeppelin is a new and upcoming web-based notebook which brings data exploration, visualization, sharing and collaboration features to Spark.
- It Integrates with many different open source, big data tools such as Apache projects Spark, Flink, Hive, Ignite, Lens and Tajo.
- Zeppelin supports many interpreters such as Apache Spark, Python, JDBC, Markdown, and Shell.
- Zeppelin notebooks are 100% opensource.
- Apache Zeppelin with Spark integration provides a number of great features including automatic SparkContext and SqlContext.
- Zeppelin aggregates values and displays them in a pivot chart with simple drag and drop.
- It helps to easily create charts with multiple aggregated values.

# Chapter 7 : Consumption By Machine Learning Models

## Productionized Data Science / AI Models Consuming Heterogenous Data

- Data stored in the staging layer often maps the source system, i.e. similar number of columns, similar data types, etc.
- A key phase in the development of Data Science / AI models is feature engineering, a process of using domain knowledge to extract features from raw data via data mining techniques. These features are then used to improve the performance of machine learning algorithms.
- Feature engineering requires several data transformation steps. These are often achieved through a combination of Hive scripts, Python/Scala scripts, which rea data from one or more source tables, transform into the required form and write to custom tables.
- The engineered feature tables are read by the Data Science / AI models to make predictions, which are then written back to other tables, which may be (depending on the permissions) read by the reporting layer.


Data Science / AI Model
Feature tables read by models
Model output written into tables
Data Ingestion Layer
Staging
Stage 1
Stage 2
HIVE
Production
Stage 3
HIVE
Trigger transformations into feature tables
Python / Scala / Hive Scripts
Reporting Layer
Model output read

MOTT MACDONALD

8

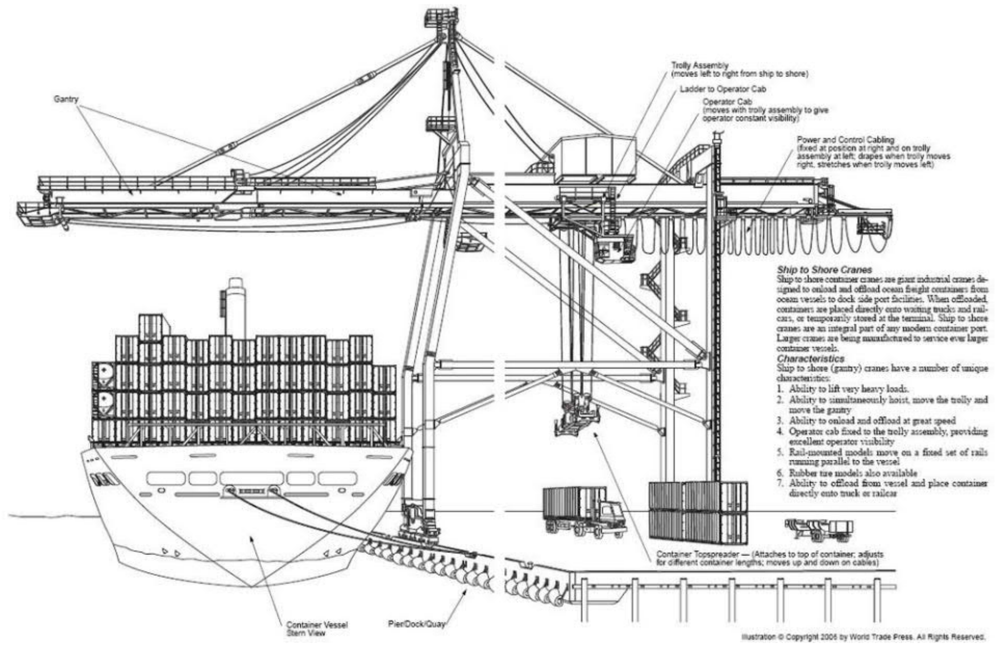
# BMA’s Big Data Ecosystem

# Chapter 8 : Contents

# Chapter 8 : Use Case, Identifying Fraud In Telecommunications Network

- Fraud in any industry creates moderate to severe financial loss. This is especially true for telecommunication companies. Traditional approaches to detect telecommunication fraud relied on blacklisting fraudulent telephone numbers. The attackers evade such detection by changing their numbers, which is very easy to achieve through VoIP.
- A solution to identifying (and reducing) network fraud required access to the signaling data generated at the start of as well as during each call.
- The solution developed ingested signalling data using Kafka and stored it into HDFS. The system was capable of ingesting several terabytes of data per day. Spark was used to process huge volumes of data with a short span (approximately 20 GB every 15 Minutes) – this was made possible because of its in-memory data processing capabilities. The solution was able to report on fraudulent calls in near-real time, thus allowing for timely corrective action to be taken by the telecommunication provider.
- Big Data Technologies Used :
  - Data Ingestion is done by using Sqoop, Flume and Kafka. These tools were used to ingest the data into HDFS.
  - After ingestion in to HDFS, a combination of Pig and Spark jobs were used to process the ingested data.
  - Hadoop supports development of custom user defined functions (UDFs). Several UDFs were developed to modularize the required complex logic of the fraud identification rules.
  - PySpark jobs were developed to write the results (of fraud identification logic) to hourly tables in Hive. These jobs were scheduled using the Oozie workflow scheduler.
  - Tableau consumed the data written into the Hive tables, thus allowing for corrective actions to be taken (when required).

# Chapter 8 : Use Case, Predictive Maintenance


Gantry
Trolly Assembly (moves left to right from ship to shore)
Ladder to Operator Cab
Operator Cab (moves with trolly assembly to give operator constant visibility)
Power and Control Cabling (fixed at position at right and on trolly assembly at left; drapes when trolly moves right, stretches when trolly moves left)
Container Topspreader — (Attaches to top of container; adjusts for different container lengths; moves up and down on cables)
Container Vessel Stern View
Pier/Dock/Quay

*Ship to Shore Cranes*
Ship to shore container cranes are giant industrial cranes designed to onload and offload ocean freight containers from ocean vessels to dock side port facilities. When offloaded, containers are placed directly onto waiting trucks and railcars, or temporarily stored at the terminal. Ship to shore cranes are an integral part of any modern container port. Larger cranes are being manufactured to service ever larger container vessels.

*Characteristics*
Ship to shore (gantry) cranes have a number of unique characteristics:

1. Ability to lift very heavy loads.
2. Ability to simultaneously hoist, move the trolly and move the gantry
3. Ability to onload and offload at great speed
4. Operator cab fixed to the trolly assembly, providing excellent operator visibility
5. Rail-mounted models move on a fixed set of rails running parallel to the vessel
6. Rubber tire models also available
7. Ability to offload from vessel and place container directly onto truck or railcar

- Most modern large scale ports with container shipping terminals operate a wide variety of complex machinery : quay-side cranes, gantry cranes, yard cranes, etc.
- Unplanned maintenance on such heavy usage machinery creates severe impact on the ports, in terms of both financial and added operational complexity.
- A major port operator required a solution to predict when a quay-side crane serving the container vessels is most likely to encounter an unexpected downtime due to an unexpected failure of one of its equipment.
- The solution developed ingested telemetry data from the cranes via a client-proprietary system, stored it into HDFS, transformed the data in near-real time (approximately 1 GB every hour) and deployed predictive models trained on historical subsets of this data.
- Upon ingestion into HDFS, the telemetry data was transformed and a downstream process used to generate the features required to feed in to the predictive models developed to predict the likelihood of failure over a pre-defined timeframe in the future. The models' predictions were saved back into HDFS and dashboards were developed to visualize model predictions on each crane's performance.

# Thank you