ACDS Lecture Series

Lecture - 16

Big Data and Hadoop

G. N. Sastry and Team

ADVANCED COMPUTATION AND DATA SCIENCES (ACDS) DIVISION

CSIR-North East Institute of Science and Technology, Jorhat, Assam, India

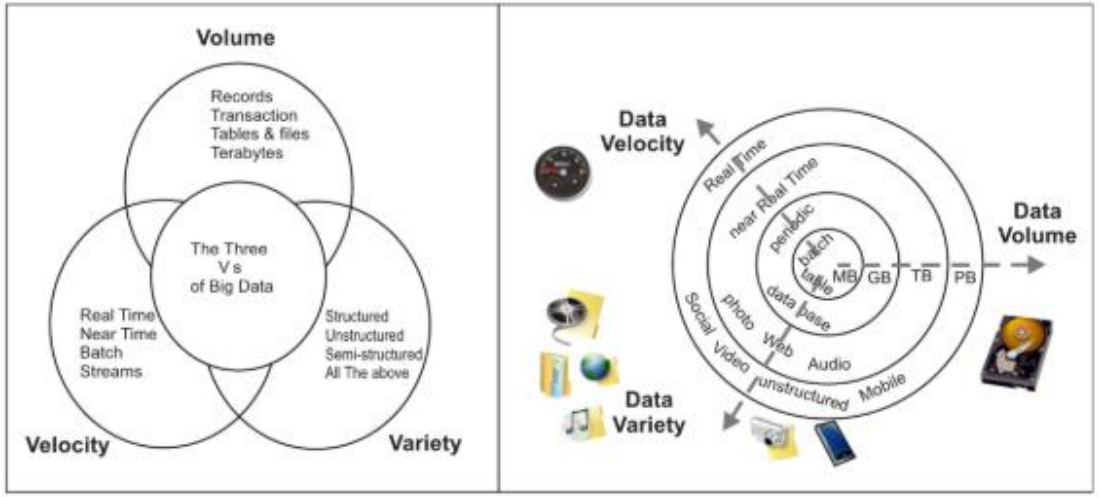
16. Big data and hadoop

16.1 Overview of big data

- Big Data: Term for massive data sets having large, more varied and complex structures.
- Center of modern science and business.
- 90% of all data has been created in the last two years.
- Today it would take a person approximately 181 million years to download all the data from the internet.
- Social media accounts for 33% of the total time spent online.
- In 2019, internet users spent 1.2 billion years online.
- 97.2% of organizations are investing in big data and AI.
- Technologies and initiatives that involve data that is too diverse, fast-changing or massive for conventional technologies.

16.1.1 The 4 or more Vs of Big Data

- Variety: This refers to the evolving types and growing sources of data, including semi-structured and structured data.
- Volume: This signifies large amount of data.
- Velocity: This refers to the speed of the data acquisition and processing
- Value : The 4th "V" is an effort to distinguish the business values of big data.
- Veracity: This signifies the messiness or trust worthiness of the data



16.1.2 Structured Data

- It concerns all data which can be stored in database SQL in table with rows and columns.
- Often managed using Structured Query Language (SQL).
- Characteristics:
  - Depends on creating a model.
  - Data can be stored as numeric, currency, alphabetic, name, date, address etc.
  - Data can be easily entered, stored, queried and analyzed.
- Structured data technology standards: SQL is a standard of the American National Standards Institute and managed by INCITS.

16.1.3 Semi-structured Data

- Information that doesn't reside in a relational database but that does have some organizational properties that make it easier to analyze.
- Their structure are irregular, implicit, flexible and often nested hierarchically.
- They have a reasonably consistent set of fields and the data are tagged.
- Semi-structured exist to ease space, clarity or compute.
- Examples – CSV, XML, JSON, NoSQL.

16.1.4 Unstructured Data

- Information that either does not have a pre-defined data model or is not organized in a pre-determined manner.
- Characteristics:
  - Typically text-heavy.
  - Fastest growing data.
  - Includes- books, journals, documents, audio, video, images etc.
- Unstructured data standards: OASIS has published UIMA standards.

16.1.5 Operational Big Data

- NoSQL data systems such as document databases are used for operational big data workloads.
- NoSQL uses new cloud computing architectures and are inexpensive and efficient.
- Some NoSQL systems can provide insights into patterns and trends based on real-time data with minimal coding.
- Example: In a multi-user game or financial application, aggregates for user activities or instrument performances are displayed to users to inform their next actions.

16.1.6 Analytical Big Data

- Analytical big data workloads are addressed by MPP database systems and MapReduce.
- MapReduce provides a new method of analyzing data that are complementary to the capabilities provided by SQL.
- MapReduce has emerged as the first choice for big data analytics.

	Operational	Analytical
Latency	1ms- 100 ms	1 min – 100 min
Concurrency	1000 – 100,000	1 – 10
Access Pattern	Writes and Reads	Reads
Queries	Selective	Unselective
Data Scope	Operational	Retrospective
End User	Customer	Data Scientist
Technology	NoSQL, MapReduce	MPP Database

16.1.7 Advantages, Issues and, challenges of Big Data

Issues:

- Issues related to the characteristics
- Storage and transport issues
- Processing issues

Challenges:

- Privacy and security
- Data access and sharing of information
- Analytical challenges
- Human resources and manpower
- Technical challenges

Advantages:

- Understanding and targeting customers.
- Understanding and optimizing business process.
- Improving science and research.
- Improving health and public health.
- Optimizing machine and device performance.
- Improving security and law enforcement.

16.2 Industry examples of Big Data

16.2.1 Fraud detection

16.2.2 Social media analysis

16.2.3 Improving health care and public health

16.2.4 The role of Big Data in medicine

16.2.1 Fraud detection

- Big data platforms can change the fraud detection game by analyzing claims and transactions in real-time, identifying large-scale patterns across many transactions or detecting anomalous behavior from an individual user.

16.2.2 Social media analysis

- Facebook has 955 million monthly active accounts using 70 languages, 140 billion photos, uploaded, 125 billion friend connections, every day 30 billion pieces of content and 2.7 billion likes and comments have been posted.
- Every minute, 72 hours of video are uploaded and every day, 4 billion views performed on YouTube.
- 1 billion Tweets every 72 hours from more than 140 million active users on Twitter
- 571 new websites are created every minute of the day.
- Social media can provide real-time insights into how the market is responding to products and campaigns. With those insights, companies can adjust their pricing, promotion, and campaign placement on the fly for optimal purpose.

  Example: IBM's Cognos Consumer Insights, a point solution running on IBM's BigInsights Big Data Platform, conducts such activities.

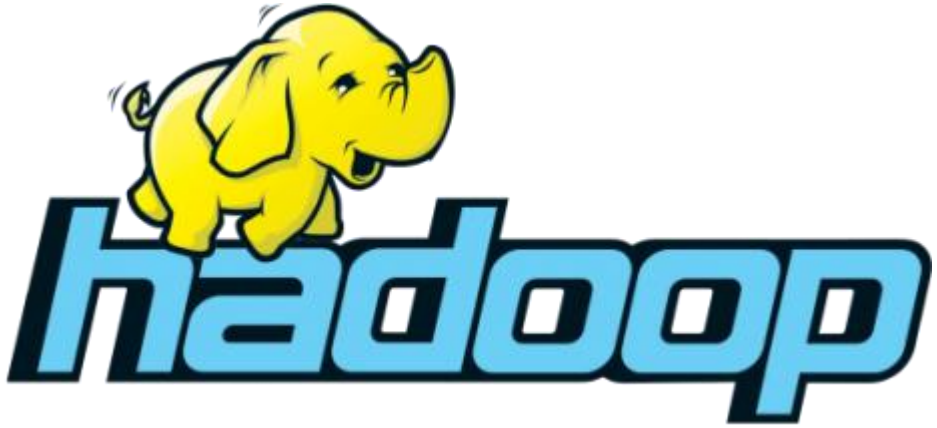
16.2.3 Improving health care and public health

- The computing power of big data analytics enables us to decode entire DNA strings in minutes and will allow us to find new cures and better understand and predict disease pattern.
- Integrating data from medical records with social media analytics enables us to monitor flu outbreaks in real-time, simply by listening to what people are saying, i.e. “Feeling homesick today- in bed with a cold”.
- Healthcare: clinical decision support systems, individual analytics applied for patient profile, personalized medicine, performance based pricing for personnel, analyze disease patterns, improve public health

16.2.4 The role of Big Data in medicine

- Our ability to advance medical care and efficiently translate science into modern medicine is bounded by our capacity to access and process these big data.
- The role of big data in medicine is one where we can build better health profiles and better predictive models around individual patients so that we can better diagnose and treat disease.

16.3 Basics of Hadoop



- Hadoop was created by Doug Cutting and Mike Cafarella in 2005.
- It is an open source software framework written in java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware.
- It is designed to scale up from a single server to thousands of machines.
- Hadoop allows to store and process big data in a distributed environment .

16.3.1 Big Data and Hadoop

- Big data is among the hottest trends in It right now, and Hadoop stands at front and center in the discussion of how to implement a big data strategy.

THE ORIGIN OF THE NAME "HADOOP"

The name Hadoop is not an acronym; it's a made-up name. The project's creator, Doug Cutting, explains how the name came about: The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are my naming criteria Kids are good at generating such. Googol is a kid's term. Projects in the Hadoop ecosystem also tend to have names that are unrelated to their function, often with an elephant or other animal theme ("Pig," for example). Smaller components are given more descriptive (and therefore more mundane) names. This is a good principle, as it means you can generally work out what something does from its name. For example, the name node manages the file system namespace.

16.3.2 Hadoop architecture

Apache Hadoop composes of the following modules-

1. Hadoop common: contains libraries and utilities needed by other Hadoop modules.
2. Hadoop Distributed File System (HDFS): a distributed file-system that stores data on commodity machines, providing very aggregate bandwidth across the cluster.
3. Hadoop YARN: a resource-management platform responsible for managing computer resources in clusters and using them for scheduling of users' application.
4. Hadoop MapReduce: a programming module for large scale data processing.

16.3.3 Main components of Hadoop framework

1. HDFS: Hadoop's own rack-aware file-system, a UNIX-based data storage layer. Data files are replicated as sequences of blocks in the cluster.

   Characteristics:

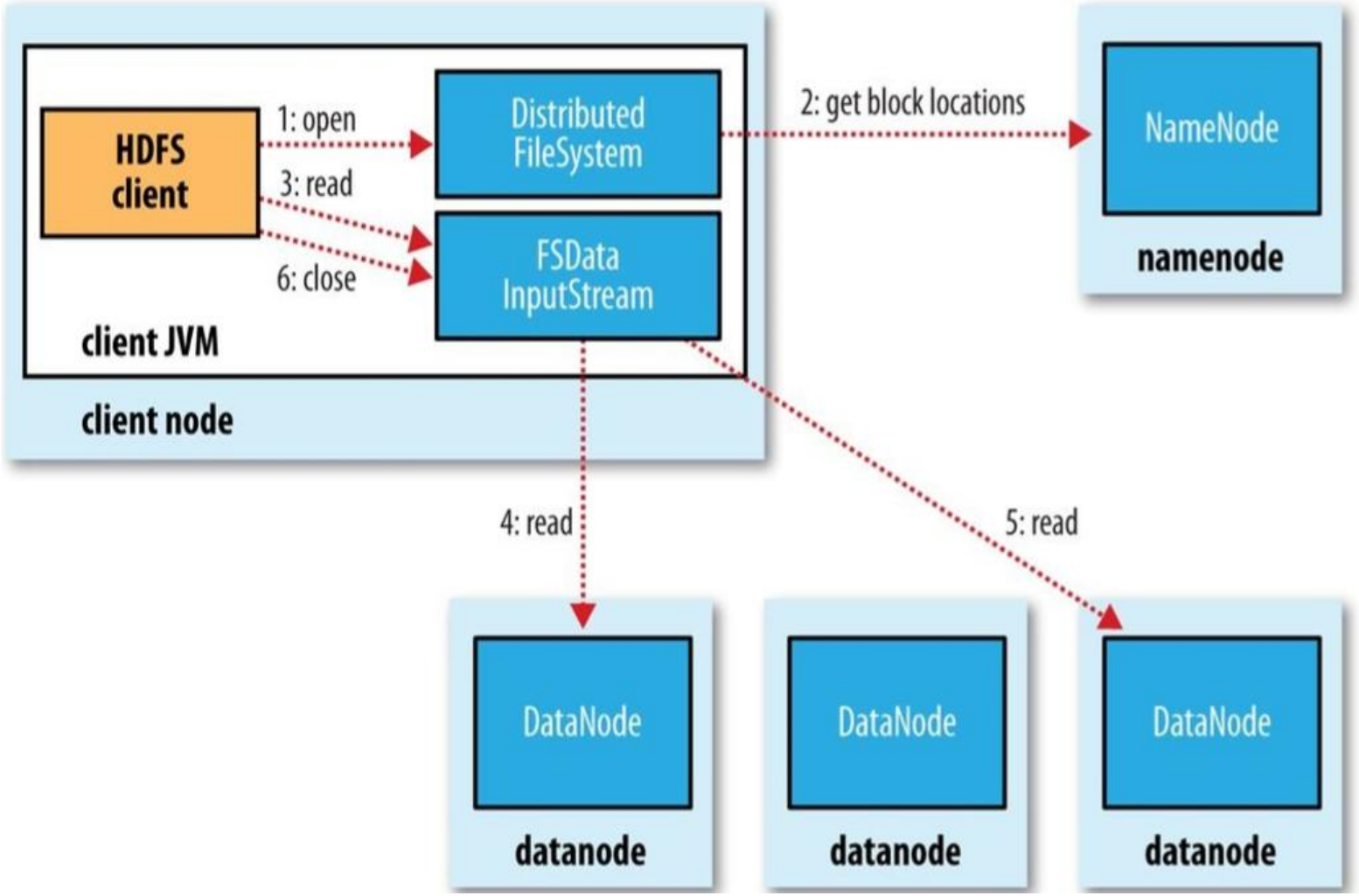
   - Fault tolerant
   - Runs with commodity hardware
   - Able to handle large datasets
   - Master slave paradigm
   - Write once file access only

2. MapReduce: Programming model for processing large datasets distributed on a large cluster. There are two phases- Map and Reduce.
   - Map phase: Once divided, datasets are assigned to the task tracker to perform the map phase. The data functional operation will be performed over the data, emitting the mapped key and value pairs as the output of the Map Phase.
   - Reduce Phase: The master node the collects the answer to all the sub-problems and combines them in some way to from the output.

16.3.3 Main components of Hadoop framework

3. HBase: A scalable, distributed database for random read/write access.
4. Pig: A high level data processing system for analyzing data sets that occur a high level language.
5. Hive: A data warehousing application that provides a SQL like interface and relational model.
6. Sqoop: A project for transferring data between relational databases and Hadoop.
7. Avro: A system of data serialization.
8. Oozie: A workflow for dependent Hadoop jobs.
9. Chukwa: A Hadoop subproject as data accumulation system for monitoring distributed systems.
10. Flume: A reliable and distributed streaming log collection.
11. ZooKeeper: A centralized service for providing distributed synchronization and group services

16.3.4 How does Hadoop work?



Step 1: The client opens the file it wishes to read by calling open() on the FileSystem object.

Step 2: DistributedFileSystem calls the namenode, using remote procedure calls (RPCs), to determine the locations of the first few blocks in the file.

Step 3: The client then calls read() on the stream

Step 4: Data is streamed from the datanode back to the client, which calls read() repeatedly on the stream.

Step 5: When the end of the block is reached, DFSInputStream will close the connection to the datanode, then find the best datanode for the next block.

Step 6: When the client has finished reading, it calls close() on the FSDataInputStream

16.3.5 Uses and Advantages of Hadoop

Advantages

- Low cost- runs on commodity hardware.
- Storage flexibility.
- Open source community.
- Fault tolerant.
- Complex data analytics.

Uses:

- Searching and text mining.
- Log processing.
- Recommendation system.
- Pattern recognition.
- Risk assessment.
- Sentiment analysis.
- Video and image analysis.
- Business intelligence/ data warehousing.
- Archiving
- Graph creation and analysis.

16.3.6 Hadoop streaming

- Hadoop streaming uses Unix standard streams as the interface between Hadoop and your program.
- Streaming is naturally suited for text processing.
- Map input datais passed over standard input to your map function, which processes it line by line and writes lines to standard output.
- A map output key-value pair is written as a single tab-delimited line.
- Input to the reduce function is in the same format- a tab-separated key-value pair, passed over standard input.
- The reduce function reads lines from standard input, which the framework guarantees are sorted by key, and writes its results to standard output.

16.4 Learning the HDFS (Hadoop Distributed File System) architecture

- HDFS is a filesystem designed for storing **very large files** with **streaming data access** patterns, running on clusters of **commodity hardware**.
  - **very large files:** files that are in hundreds of megabytes, gigabytes, or terabytes.
  - **streaming data access:** most efficient data processing pattern is a write-once, read-many-times pattern.
  - **commodity hardware:** Hadoop doesn't require expensive, highly reliable hardware. It's designed to run on clusters of commodity hardware.
- Applications for which using HDFS does not work so well:
  - Low-latency data access.
  - Lots of small files.
  - Multiple writers, arbitrary file modifications.

16.4.1 NameNode

- Master of the HDFS system.
- Maintains the directories, files, and manages filesystem namespace.
- Information stored persistently on the local disk in the form of two files: the namespace image and the edit log.
- The filesystem cannot be used without the namenode. Therefore it is important to make the namenode resilient to failure.
- Hadoop provides two mechanisms for this:
    1. Back up the files that make up the persistent state of the filesystem metadata.
    2. Run a secondary namenode. Its main role is to periodically merge the namespace image with the edit log to prevent the edit log from becoming too large. Usually runs on a separate physical machine

16.4.2 DataNode

- Slaves that are deployed on each machine and provide actual storage.
- Responsible for serving read-and-write data request for the clients.
- They are the workhouse of the filesystem.
- They store and retrieve blocks when they are told to, and they report back to the namenode periodically with lists of blocks that they are storing.

16.4.3 HDFS file blocks

- Data in filesystem is dealt in blocks, which are an integral multiple of the disk block size. Block size is the minimum amount of data that can be read or written.
- HDFS has a large block size, 128 MB by default (Normal block size = 512 bytes).
- Files in HDFS are broken into block sized chunks, which are stored as independent units.
- HDFS's fsck command understands blocks. For example, running:

```
% hdfs fsck / -files -blocks
```

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

16.4.4 HDFS filesystems

Filesystem	URI scheme	Java implementation (all under org.apache.hadoop)	Description
Local	file	fs.LocalFileSystem	A filesystem for a locally connected disk with client-side chechsums. Use RawLocalSystem for a local filesystems with no checksums.
HDFS	hdfs	hdfs.DistributedFileSystem.	Hadoop's distributed filesystem. HDFS is designed to work efficiently in conjunction with MapReduce.
WebHDFS	webhdfs	hdfs.web.WebHdfsFileSystem	A filesystem providing authenticated read/write access to HDFS over HTTP
Secure WebHDFS	swebhdfs.	hdfs.web.SWebHdfsFileSystem.	The HTTPS version of WebHDFS.
HAR	har	fs.HarFileSystem	A filesystem layered on another filesystem for archiving files. Hadoop Archives are used for packing lots of files in HDFS into a single archive file to reduce the namenode's memory usage. Use the hadoop archive command to create HAR files.
View	viewfs	viewfs.ViewFileSystem	A client-side mount table for other Hadoop filesystems. Commonly used to create mount points for federated namenodes.
FTP	ftp	fs.ftp.FTPFileSystem	A filesystem backed by an FTP server.
S3	s3a	fs.s3a.S3AFileSystem	A filesystem backed by Amazon S3. Replaces the older s3n (S3 native) implementation.
Azure	wasb	fs.azure.NativeAzureFileSystem	A filesystem backed by Microsoft Azure.
Swift	swift	fs.swift.snative.SwiftNativeFileSystem	A filesystem backed by OpenStack Swift

16.4.5 HDFS Commands

- The Hadoop command line environment is Linux-like.
- The syntax of Hadoop fs shell command is as follows:

hadoop fs <args>

1. Create a directory in HDFS at the given path(s):
   - Usage:

     ```
     hadoop fs -mkdir <paths>
     ```

   - Example:

     ```
     hadoop fs -mkdir usershiva/dir1 usershiva/dir2
     ```

2. List the contents of a directory:
   - Usage:

     ```
     hadoop fs -ls <args>
     ```

   - Example:

     ```
     hadoop fs -ls usershiva
     ```

3. Put and Get a file in HDFS:
   - Usage(Put):

     ```
     hadoop fs -put <localsrc> ... <HDFS_dest_Path>
     ```

   - Example:

     ```
     hadoop fs -put homeshiva/Samplefile.txt  usershiva/dir3/
     ```

   - Usage(Get):

     ```
     hadoop fs -get <hdfs_src> <localdst>
     ```

   - Example:

     ```
     hadoop fs -get usershiva/dir3/Samplefile.txt home
     ```

4. See contents of a file:
   - Usage:

     ```
     hadoop fs -cat <path[filename]>
     ```

   - Example:

     ```
     hadoop fs -cat usershiva/dir1/abc.txt
     ```

5. Copy a file from source to destination:
   - Usage:

     ```
     hadoop fs -cp <source> <dest>
     ```

   - Example:

     ```
     hadoop fs -cp usershiva/dir1/abc.txt usershiva/dir2
     ```

16.4.5 HDFS Commands

6. Copy a file from/To Local filesystem to HDFS:
   - Usage of copyFromLocal:

     ```
     hadoop fs -copyFromLocal <localsrc> URI
     ```

   - Example:

     ```
     hadoop fs -copyFromLocal homeshiva/abc.txt
     usershiva/abc.txt
     ```

   - Usage of copyToLocal

     ```
     hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>
     ```

7. Move file from source to destination:
   - Usage:

     ```
     hadoop fs -mv <src> <dest>
     ```

   - Example:

     ```
     hadoop fs -mv usershiva/dir1/abc.txt usershiva/dir2
     ```

8. Remove a file or directory in HDFS:
   - Usage:

     ```
     hadoop fs -rm <arg>
     ```

   - Example:

     ```
     hadoop fs -rm usershiva/dir1/abc.txt
     ```

   - Usage of the recursive version of delete:

     ```
     hadoop fs -rmr <arg>
     ```

   - Example:

     ```
     hadoop fs -rmr usershiva/
     ```

9. Display the last few lines of a file:
   - Usage:

     ```
     hadoop fs -tail <path[filename]>
     ```

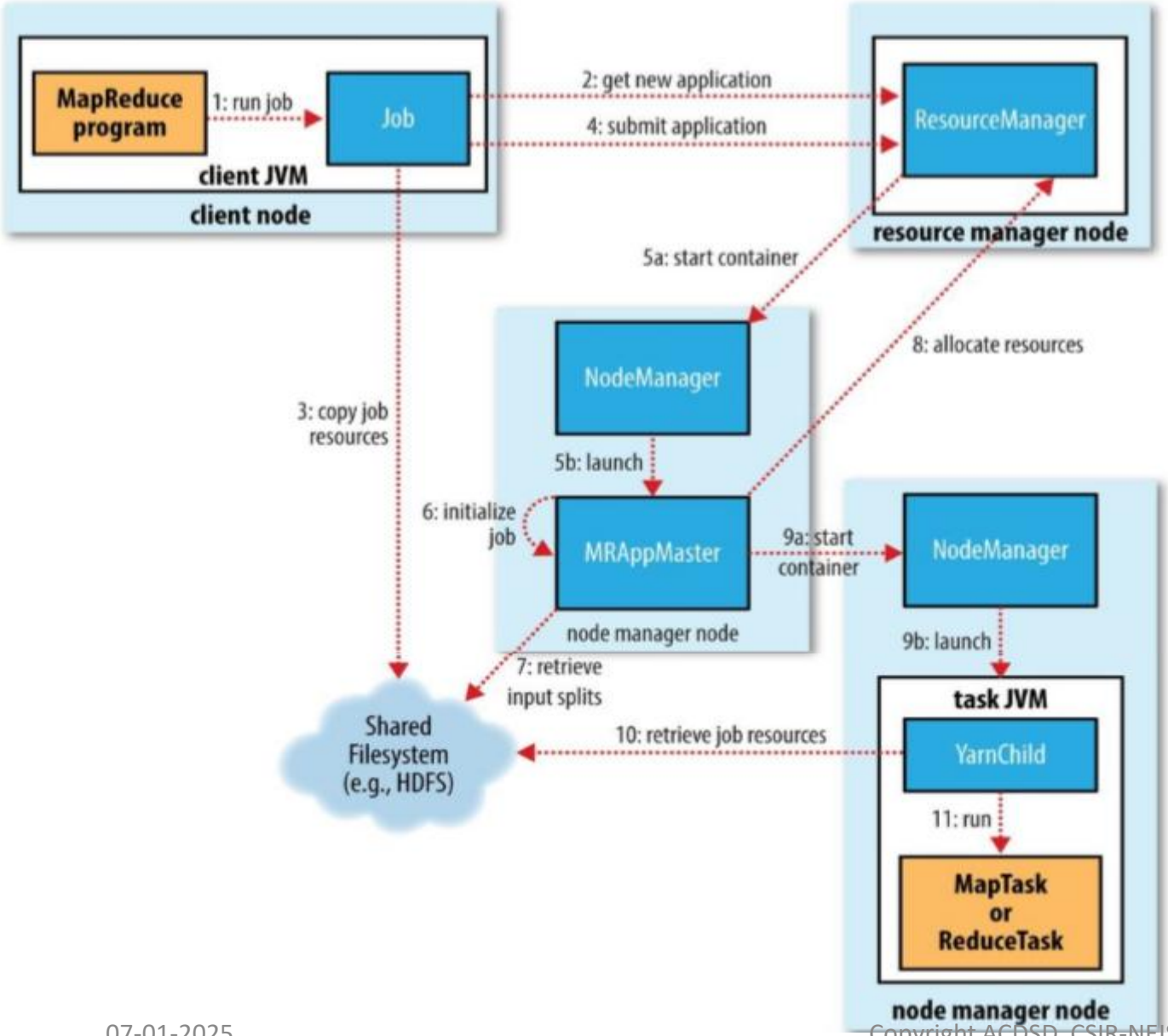
   - Example:

     ```
     hadoop fs -tail usershiva/dir1/abc.txt
     ```

16.5 The MapReduce's engines: JobTracker and TaskTracker

- Classic MapReduce contains job submission, job initialization, task assignment, task execution, progress and status update.
- Created by Google using the divide and conquer method to break down complex big data problems into small units of work and process them in parallel
- MapReduce is managed with master-slave architecture with two components.
  - JobTracker: Master coordinator daemon, responsible for coordinating and completing a MapReduce job in Hadoop, resource management, tracking resource availability, and task process cycle.
  - TaskTracker: Slaves daemon that performs task assigned by JobTracker. TaskTracker sends heartbeat messages to JobTracker periodically to notify about free slots and sends the status to JobTracker about the task.

16.5.1 How MapReduce works?



- The client, which submits the MapReduce job.
- The YARN resource manager, which coordinates the allocation of compute resources on the cluster.
- The YARN node managers, which launch and monitor the compute containers on machines in the cluster.
- The MapReduce applicationmaster, which coordinates the tasks running the MapReduce job. The application master and the MapReduce tasks run in containers that are scheduled by the resource manager and managed by the node managers.
- The distributed filesystem (normally HDFS), which is used for sharing job files between the other entities.

16.5.2 MapReduce example

A Weather Dataset

For our example, we will write a program that mines weather data. Weather sensors collect data every hour at many locations across the globe and gather a large volume of log data, which is a good candidate for analysis with MapReduce because we want to process all the data, and the data is semi-structured and record-oriented.

Example: Map function for maximum temperature in Python

```
#!/usr/bin/env python

import re
import sys

for line in sys.stdin:
  val = line.strip()
  (year, temp, q) = (val[15:19], val[87:92], val[92:93])
  if (temp != "+9999" and re.match("[01459]", q)):
    print "%s\t%s" % (year, temp)
```

Example: Reduce function for maximum temperature in Python

```
#!/usr/bin/env python

import sys

(last_key, max_val) = (None, -sys.maxint)
for line in sys.stdin:
  (key, val) = line.strip().split("\t")
  if last_key and last_key != key:
    print "%s\t%s" % (last_key, max_val)
    (last_key, max_val) = (key, int(val))
  else:
    (last_key, max_val) = (key, max(max_val, int(val)))

if last_key:
  print "%s\t%s" % (last_key, max_val)
```

16.5.2 MapReduce example

Application to calculate the proportion of records with missing fields

```
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.util.*;

public class MissingTemperatureFields extends Configured implements Tool {

  @Override
  public int run(String[] args) throws Exception {
    if (args.length != 1) {
      JobBuilder.printUsage(this, "<job ID>");
      return -1;
    }
    String jobID = args[0];
    Cluster cluster = new Cluster(getConf());
    Job job = cluster.getJob(JobID.forName(jobID));
    if (job == null) {
      System.err.printf("No job with ID %s found.\n", jobID);
      return -1;
    }
    if (!job.isComplete()) {
      System.err.printf("Job %s is not complete.\n", jobID);
      return -1;
    }

      Counters counters = job.getCounters();
      long missing = counters.findCounter(
          MaxTemperatureWithCounters.Temperature.MISSING).getValue();
      long total = counters.findCounter(TaskCounter.MAP_INPUT_RECORDS).getValue();

      System.out.printf("Records with missing temperature fields: %.2f%%\n",
          100.0 * missing / total);
      return 0;
    }
    public static void main(String[] args) throws Exception {
      int exitCode = ToolRunner.run(new MissingTemperatureFields(), args);
      System.exit(exitCode);
    }
}
```

16.5.3 Hadoop's MapReduce

The Writable interface

• The Writable interface is used for values for serialization and deserialization.

• Some of the classes that implement the Writable interface are ArrayWritable, BooleanWritable, ByteWritable, DoubleWritable, FloatWritable, IntWritable, LongWritable, MapWritable, NullWritable, ObjectWritable, ShortWritable, TupleWritable, VIntWritable, and VLongWritable.

• We can create our own custom Writable class that can be used in MapReduce. For creating a class, we have to implement the Writable class and implement the following two methods:

- void write (DataOutput out): This serializes the object.
- void readFields (DataInput in): This reads the input stream and converts it to an object.

WritableComparable interface

• WritableComparable is used for keys, which is inherited from the Writable interface and implements a comparable interface to provide comparison of value Objects.

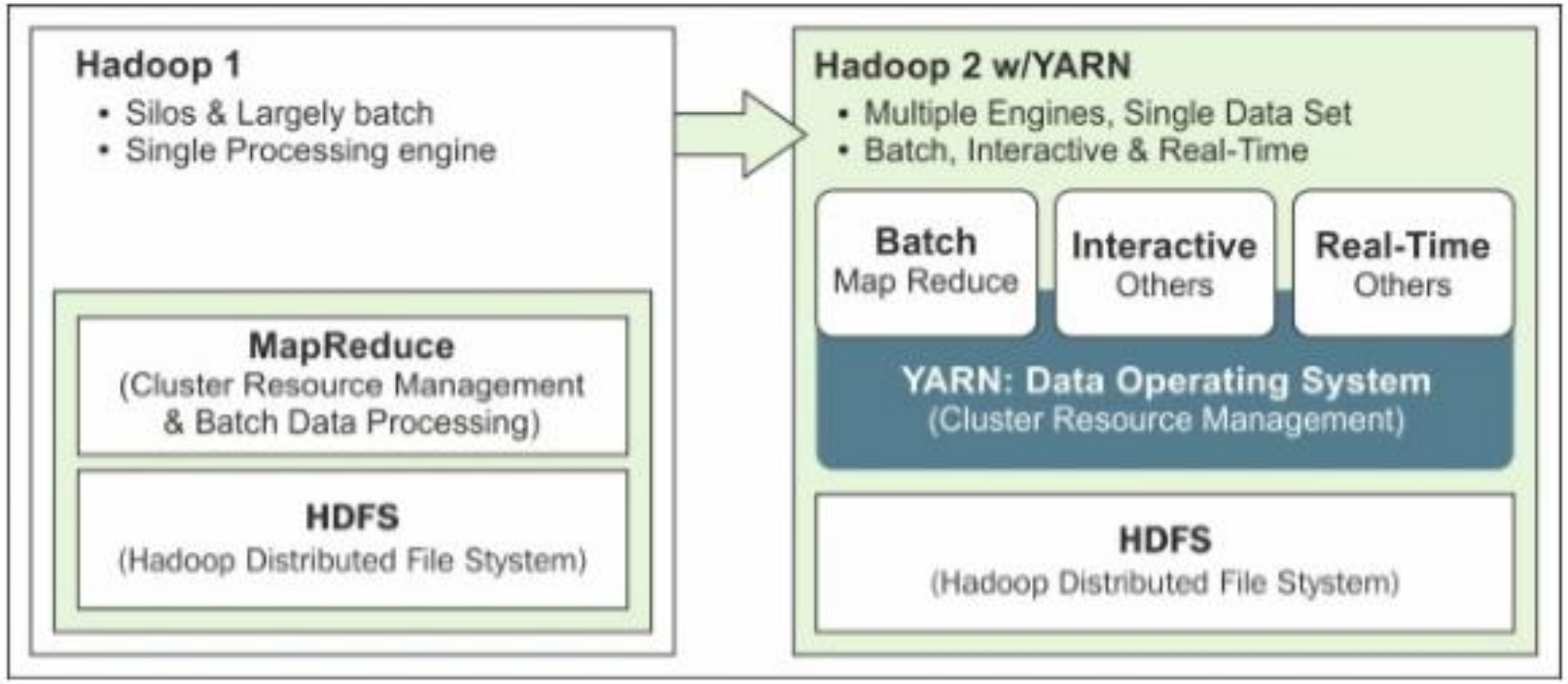
• Some of the implementations are BooleanWritable, BytesWritable, ByteWritable, DoubleWritable, FloatWritable, IntWritable, LongWritable, NullWritable, ShortWritable, Text, VIntWritable, and VLongWritable.

• We can create our own custom WritableComparable class that can be used in MapReduce. For creating a class, we have to implement WritableComparable class and implement the following three methods:

- void write (DataPutput out): This serializes the object
- void readFields (DataInput in): This reads the input stream and converts it to an object
- Int compareTo (Object obj): Compare the values required to sort the key

16.6 YARN (Yet Another Resource Negotiator)

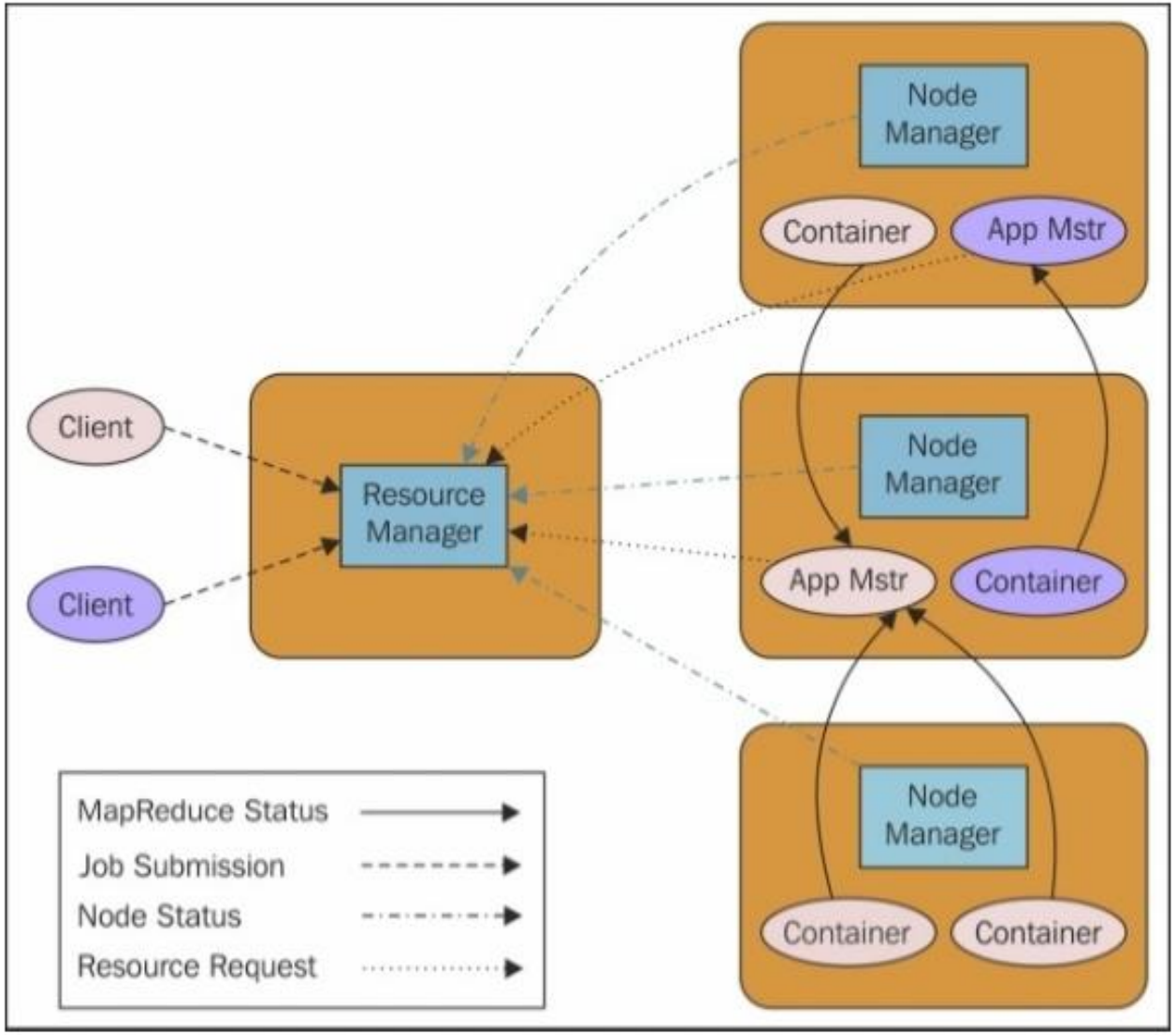
- Next generation compute and cluster management technology.
- Provides platform to build/run multiple distributed apllications in Hadoop.
- Released in the Hadoop 2.0 version in 2012.
- YARN delegates and splits up the responsibility into separate daemons and achieves better performance and fault tolerance.



16.6.1 YARN Architecture

YARN architecture is extremely scalable, fault tolerant, and processes data faster as compared to MapReduce. YARN architecture has the following three components:

- ResourceManager (RM)
- NodeManager (NM)
- ApplicationMaster (AM)



16.6.2 Application powered by YARN

- Apache Giraph: Graph processing
- Apache Hama: Advanced Analytics
- Apache Hadoop MapReduce: Batch processing
- Apache Tez: Interactive/Batch on top of Hive
- Apache S4: Stream processing
- Apache Samza: Stream processing
- Apache Storm: Stream processing
- Apache Spark: Realtime Iterativeprocessing
- Hoya:Hbase on YARN

16.7.1 PIG architecture

- Pig is a component which has the abstraction wrapper of Pig Latin language on top of MapReduce. Pig was developed by Yahoo! around 2006 and was contributed to Apache as an open source project.
- Pig can be used in a much easier way for structured and semi-structured data analysis.
- The Pig data flow architecture is layered for transforming Pig Latin statements to MapReduce steps.
- There are three main phases in compiling and executing a Pig script, which are as follows:

Logical plan	Physical plan	MapReduce plan
• parsed for syntax errors and validation of the input files and input data structures. • has a one-to-one correspondence with the operators.	• translation of each operator into the physical form of execution • most operators have a one-to-one correspondence with the physical plan.	• The final stage of Pig compilation is to compile the physical plan to actual MapReduce jobs • The compiler also looks for opportunities to put in Combiners wherever possible.

16.7.2 PIG modes

APACHE PIG

The user can run Pig in two modes

- Local Mode
- MapReduce Mode

Three modes of execution

- Interactive mode or grunt mode
- Batch mode or script mode
- Embedded mode

16.7.2 PIG Commands

- We will be using the movies_data.csv file as a dataset for exploring Pig commands.
- Loading data: Use LOAD command

Example:

```
grunt> movies = LOAD 'userbiadmin/shiva/movies_data.csv' USING
PigStorage(',') as (id,name,year,rating,duration);
```

We can use schemas to assign types to fields:

```
A = LOAD 'data' AS (name, age, gpa); // name, age, gpa default to
bytearrays
A = LOAD 'data' AS (name:chararray, age:int, gpa:float); // name is
now a String (chararray), age is integer and gpa is float
```

- Dump: The dump command is very useful to interactively view the values stored in the relation and writes the output to the console.

Example:

```
grunt> DUMP movies;
INFO  [JobControl]
org.apache.hadoop.mapreduce.lib.input.FileInputFormat     - Total
input paths to process : 1
INFO  [main] org.apache.hadoop.mapreduce.lib.input.FileInputFormat
- Total input paths to process : 1
(1,The Nightmare Before Christmas,1993,3.9,4568)
(2,The Mummy,1932,3.5,4388)
(3,Orphans of the Storm,1921,3.2,9062)
(4,The Object of Beauty,1991,2.8,6150)
(5,Night Tide,1963,2.8,5126)
(6,One Magic Christmas,1985,3.8,5333)
(7,Muriel's Wedding,1994,3.5,6323)
```

16.7.2 PIG Commands

- **Store**

The store command is used to write or continue with the data. Pig starts a job only when a DUMP or STORE is encountered

Example:

```
grunt> STORE movies INTO 'temp' USING PigStorage(','); /This will
write contents of movies to HDFS in /temp location
```

- **FOREACH generate**

A FOREACH operation is used to apply a column-level expression in each record of the relation.

Example:

```
grunt> movie_duration = FOREACH movies GENERATE name, (double)
(duration/60);
```

- **Filter**

Filter is used to get rows matching the expression criteria.

Example:

```
grunt> movies_greater_than_four = FILTER movies BY
(float)rating>4.0;
grunt> DUMP movies_greater_than_four;
```

We can use multiple conditions with filters and Boolean operators (AND, OR, NOT):

```
grunt> movies_greater_than_four_and_2012 = FILTER movies BY
(float)rating>4.0 AND year > 2012;
grunt> DUMP movies_greater_than_four_and_2012;
INFO  [JobControl]
org.apache.hadoop.mapreduce.lib.input.FileInputFormat     - Total
input paths to process : 1
WARN  [main] org.apache.pig.data.SchemaTupleBackend     -
SchemaTupleBackend has already been initialized
INFO  [main] org.apache.hadoop.mapreduce.lib.input.FileInputFormat
- Total input paths to process : 1
(22148,House of Cards: Season 1,2013,4.4,)
(22403,House of Cards,2013,4.4,)
(37138,Orange Is the New Black: Season 1,2013,4.5,)
(37141,Orange Is the New Black,2013,4.5,)
(37174,The Following: Season 1,2013,4.1,)
(37239,The Following,2013,4.1,)
(37318,The Carrie Diaries,2013,4.3,)
(37320,The Carrie Diaries: Season 1,2013,4.3,)
(37589,Safe Haven,2013,4.2,6936)
```

16.7.3 HIVE architecture

- Hive provides a data warehouse environment in Hadoop with a SQL-like wrapper and also translates the SQL commands in MapReduce jobs for processing.
- Hive architecture has different components.

Driver	Metastore	Query Compiler	Execution Engine	HiveServer2	Client components
• manages the lifecycle of a HiveQL • also maintains a session handle for session statistics	• stores the system catalog and metadata about tables, columns, partitions, and so on.	• compiles HiveQL into a DAG of optimized map/reduce tasks.	• executes the tasks produced by the compiler in a proper dependency order.	• provides a thrift interface and a JDBC/ODBC server • provides a way of integrating Hive with other applications	• Command Line Interface (CLI • the web UI, and drivers

16.7.4. HIVE data types

TYPES:

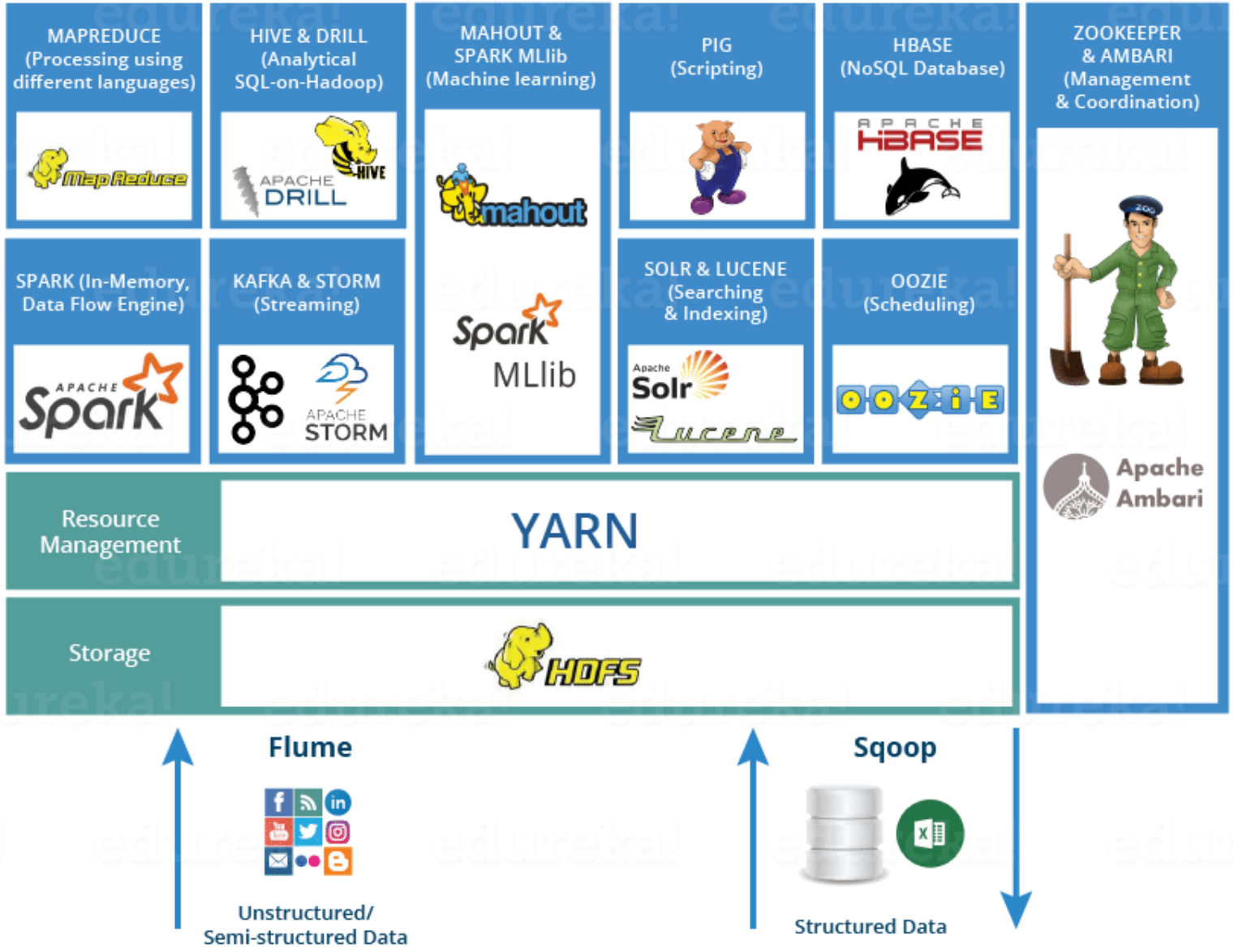
• STRUCT: These are groupings of data elements similar to a C-struct
Syntax: STRUCT<field_name : data_type>

• MAP: These are key value data types.
Syntax: MAP<primitive_type, data_type>

• ARRAY: These are lists that can be randomly accessed through their position.
Syntax: ARRAY<data_type>

• UNION: It can hold an element of one of the data types specified in the union.
Syntax: UNIONTYPE<data_type1, data_type2…>

16.8 Hadoop Ecosystem



16.9 Summary

16.11 References: Books

1. Jain, V. K.; *Big Data and Hadoop*, Khanna Book Publishing, **2017.**
2. Prajapati, V.; *Big Data Analytics with R and Hadoop*, Packt Publishing, **2013.**
3. Russom, P.; *Big Data Analytics*, TDWI Best Practices Report, **2011**.
4. Achari, S.; *Hadoop Essentials*, Packt Publishing, **2015.**
5. White, T.; *Hadoop: The Definitive Guide*, O'Reilly Publishing, **2015.**

16.11 References: Research articles

1. Sagiroglu, S.; Sinanc, D. *Big data: A review*, IEEE, International Conference on Collaboration Technologies and Systems (CTS), (**2013**), DOI**:** 10.1109/CTS.2013.6567202.
2. Chen, M.; Mao, S.; Liu, Y. *Big Data: A survey*, Mobile Network Application, (**2014**) 19:171–209, DOI: 10.1007/s11036-013-0489-0.
3. Labrinidis, A.; Jagadish, H,V. *Challenges and opportunities with Big Data*, Proceedings of the VLDB Endowment, The 38th International Conference on Very Large Data Bases, **(2012)** 5: 12, DOI: 10.14778/2367502.2367572.
4. Satyanarayana, L, V. *A survey on challenges and advantages in Big Data*, International Journal of Computer Science and Technology, **(2015)** 6:2**,** 115- 119.

THANK YOU