7. Introduction to Big Data with Spark and Hadoop

- Hadoop Ecosystem
  - Ingest data
  - Store data
  - Analyze data
  - Access data
- HDFS
  - Blocks
  - Nodes
  - Replication
  - Read and write
  - HDFS architecture
- Hive
  - Hive architecture
- Apache HBASE
  - HBase architecture
  - Concepts
- Hands-on Lab: Hadoop MapReduce
- Summary & Highlights

Semana 3

- Why use Apache Spark?
- Parallel Programming using Resilient Distributed Datasets
  - RDDs
  - Creating an RDD in Spark
  - Parallel programming
- Scale out / Data Parallelism in Apache Spark
  - Components
  - Spark core
  - Scaling big data in Spark
- Dataframes and SparkSQL
- LAB → notebook sparkintro
- Summary & Highlights
- Practice quiz

Semana 4

- RDDs in Parallel Programming and Spark
  - Transformations
  - Actions
  - Directed Acyclyc Graph (DAG)
  - Transformations and actions
- Data-frames and Datasets
- ETL with DataFrames
  - Read the data
  - Analyze the data

Semana 1

Course introduction

The latest statistics report that the accumulated world’s data will grow from 4.4 zettabytes to 44 zettabytes, with much of that data classified as Big Data. Revenues based on Big Data analytics are projected to increase to $103 billion by 2027. Understandably, organizations across industries want to harness the competitive advantages of Big Data analytics. This course provides you with the foundational knowledge and hands-on lab experience you need to understand what Big Data is and learn how organizations use Apache Hadoop, Apache Spark, including Apache Spark SQL, and Kubernetes to expedite and optimize Big Data processing.

What is big data?

small data

- small enough for human inference
- accumulates slowly
- relatively consistent and structured data usually stored in known forms such as JSON and XML
- mostly located in storage systems within enterprises or data centers

big data

- generated in huge volumes and could be structured, semi-structured or unstructured
- needs processing to generate insights for human consumption
- arrives continously at enormous speed from multple sources
- comprises any form of data including video, photos, and more
- distributed on the cloud and server forms

Life cicle:

business case → data collection → data modeling → data processing → data visualization

1025 GB → 1 Terabyte

1024 TB → 1 Petabyte

1024 PB → 1 Exabyte

1024 EX → 1 Zettabyte

1024 ZB → 1 Yottabyte

The four V'2

- VELOCITY:
  - Description: speed at which data arrives
  - Attributes:
    - batch
    - close to real time

        - streaming
    - Drivers:
        - improved connectivity and hardware
        - rapid response times
- VOLUME:
    - Description: increase in the amount of data stored over time
    - Attribute:
        - petabytes
        - exa
        - zetta
    - Drivers:
        - increase in data sources
        - higher resolution sensors
        - scalable infrastructure
- VARIETY:
    - Description: many forms of data exist and need to be stored
    - Attributess:
        - structure
        - complexity
        - origin
    - Drivers:
        - mobile tech
        - scalable infrastructure
        - resilience
        - fault recovery
        - efficient storage and retrieval
- VERACITY:

- Description: the certainty of data
- Attributes:
  - consistency and completeness
  - integrity
  - amibuity
- Drivers:
  - cost and traceability
  - robus ingestion
  - ETL mechanisms

→ The fifth V is **VALUE ($)**

Parallel Processing, Scaling, and Data Parallelism

Parallel processing

Linear processing → instructions executed sequentially and if there's an error it starts again

→ for minor computing tasks

Parallel processing:

The problem is also devided into instructions but each instructions goes to a separate node with equal processing capacity and are executed in paralell.

Errors can be re-executed locally and it doesnt affect other instructions

- faster
- less memory and compute requirements
- flexibility. Execution nodes can be added and remove depending on the necessity

Data scaling

→ scaling UP : when the storage is full and you enlarge it

HORIZONTAL SCALING:

- adding additional nodes with same capacity
- This collections of nodes is known as **cluster capacity**
- if one process fails it doesnt affect the others and can be easily re-run

Fault tolerance:

> if node one has partitions P1, P2 and P3, and it fails, we can easily add a new node and recover these partitions from the copies they had in other nodes

Big Data Tools and Ecosystem

Key technologies include:

- hadoop
- HDFS
- spark
- mapReduce
- cloudera
- databricks

Analytics and visualization:

- tableau
- palantir
- SAS
- pentaho
- teradata

Business intelligence:

BI offers a range of tools that provide a quick and easy way to transform data into actionable insights

- cognos
- oracle
- powerBI
- business objects
- hyperion

Cloud providers

- ibm
- aws
- oracle

Programming tools

- R
- python
- scala
- julia

Open Source and Big Data

Hadoop plays a major role in open source Big Data projects.

Its three main components are:

- Hadoop MapReduce
    - framework that allows code to be written to run at scale on a hadoop cluster

    - less used than apache spark
- Hadoop File System (HDFS)
    - file system that stores and manages big data files
    - manages issues around large and distributed datasets
    - reilience and partitioning
    - still used in the industry → 70% of the world's big data resides on HDFS
    - S3 is also coming into use
- Yet Another Resource Negotiator (YARN)
    - resource manager
    - default resource manager for many big data apps, including hive and spark
    - kubernets is slowly becoming the new defacto standard but YARN is still very used

Beyond the Hype

FACT: more data has been created in the past two years than in the entire previous history of humankind

Where does big data come from?

- social data
    - social media
    - images
    - video
    - comments
- machine data
    - iot
    - sensors
- transactional data
    - invoices

    - payment orders
    - storage records
    - delivery receipts

Big Data Use Cases

- retail
    - price analytics
    - sentimental analysis
        - what consumers think of the product
- insurance
    - fraud analytics
    - risk assesment
- telecomm
    - improve network security
    - contextualized location-based promotions
    - real-time network analytics
    - optimized pricing
- manufacturing
    - predivtive maintenance
        - example: for machines
    - production optimization
- automotive industry
    - predictive support
    - connected self-driven cars
- finance
    - customer segmentation
    - algorithmic trading

Summary & Highlights

- Personal assistants like Siri, Alexa and Google Now, use Big Data and IoT to gather data and devise answers.
- Big Data Analytics helps companies gain insights from the data collected by IoT devices.
- Big Data requires parallel processing on account of massive volumes of data that are too large to fit on any one computer.
- "Embarrassingly parallel” calculations are the kinds of workloads that can easily be divided and run independently of one another. If any single process fails, that process has no impact on the other processes and can simply be re-run.
- Open-source projects, which are free and completely transparent, run the world of Big Data and include the Hadoop project and big data tools like Apache Hive and Apache Spark.
- The Big Data tool ecosystem includes the following six main tooling categories: data technologies, analytics and visualization, business intelligence, cloud providers, NoSQL databases, and programming tools.

Semana 2

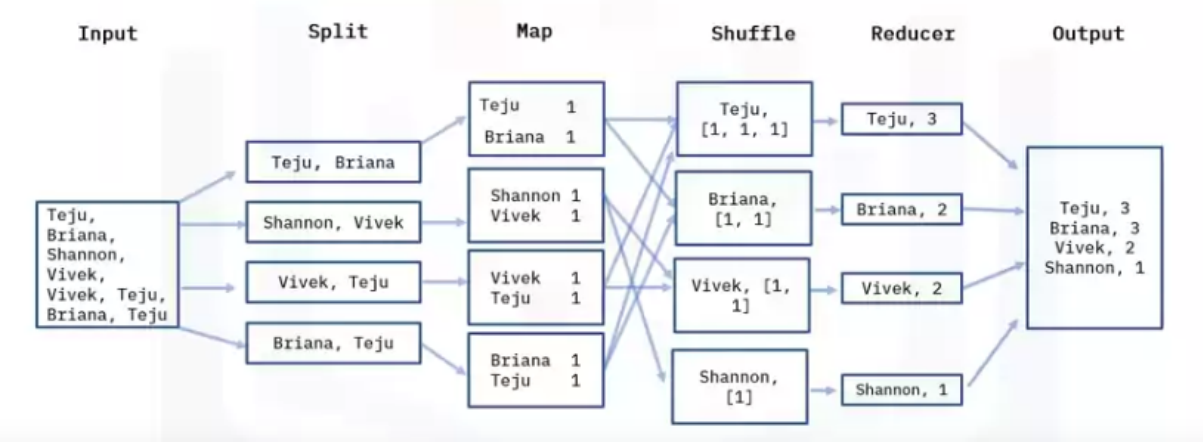
Introduction to Hadoop

- open-source program to process large data sets
- servers run applications on clusters
- handles parallel jobs or processes. Not a database but an ecosystem
- structured, unstructured and semi-structured data
- core components
    - **hadoop common:** essential part of the framework that refers to the collection of common utilities and libraries that support other hadoop modules
    - **HDFS:** handles large data sets running in comodity hardware, that is, low-specifications industry-grade hardware. HDFS scales a singles hadoop cluster into as much as thousand clusters

    - **MapReduce:** processes data bu splitting the data into smaller units. It was the first method to query data stored in HDFS
    - **YARN:** prepares RAM and CPU in Hadoop to run batch processes, stream, interactive and graph processing
- hadoop is NOT good for:
    - processing transactions (lack of random access)
    - when work cannot be parallelized
    - when there are dependencies in the data
    - low latency data access
    - processing lots of small files
    - intensive calculations with little data

Intro to MapReduce

- programming model used in hadoop for processing big data
- processing technique for distributed computing
- based on java
- two tasks
    - MAP
        - input file
        - processes data into key value pairs
        - further data sortin and organizing
    - REDUCE
        - aggregates and computes a set of result and produces a final output
    - MAPREDUCE
        - keeps track of its task by creating a unique key



Why mapReduce?

- parallel computing
  - devide → run tasks → done
- flexibility
  - process tabular and non tabular forms, such as videos
  - support for multiple languages
- platforn for analysis and data ware housing
- Common use cases:
  - social media
  - recommendatios (netflix recommendations algorithm)
  - financial industries
  - advertisment (google ads)

Hadoop Ecosystem

- made of components that suppot one another for big data processing

INGEST DATA (ex: Flume and Sqoop) → STORE DATA (ex: HDFS and HBase) → PROCESS AND ANALYZE DATA (ex: Pig and Hive) → ACCESS DATA (ex: Impala and Hue)

Ingest data

- Flume

    - collects, aggregates and transfers big data
    - has a simple and flexible architecture based on streaming data flows
    - uses a simple extensible data model that allows for online analytic application
- Sqoop
    - designed to transfer data between relational database systems and hadoop
    - accesses the database to understand the schema of the data
    - generates a MapReduce application to import or export the data

Store data

- HBase
    - non-relational database that runs on top of HDFS
    - provides real time wrangling on data
    - stores data as indexes to allow for random and faster access to data
- Cassandra
    - scalable, noSQL database designed to have no single point of failure

Analyze data

- Pig
    - analyzes large amounts of data
    - operates on the **client side** of a cluster
    - procedural data flow language that follows an order and a set of commands
- Hive
    - creating reports
    - operates on the **server side** of the cluster
    - declarative programming language, which means it allows users to express which data they wish to recieve

Access data

- Impala

    - scalable and easy to use platform for everyone
    - no programming skills required
- Hue
    - stands for hadoop user experience
    - allows you to upload, browse, and query data
    - runs pig jobs and workflow
    - provides editors for several SQL query languages like Hive and MySQL

HDFS

- Hadoop distributed file system
- storage layer of hadoop
- splits the files into blocks, creates replicas of the blocks, and stores them on different machines
- command line interface to interact with hadoop
- provides access to streaming data
    - this means that HDFS provides a constant bitrate when transferring data rather than having the data being transferred in waves

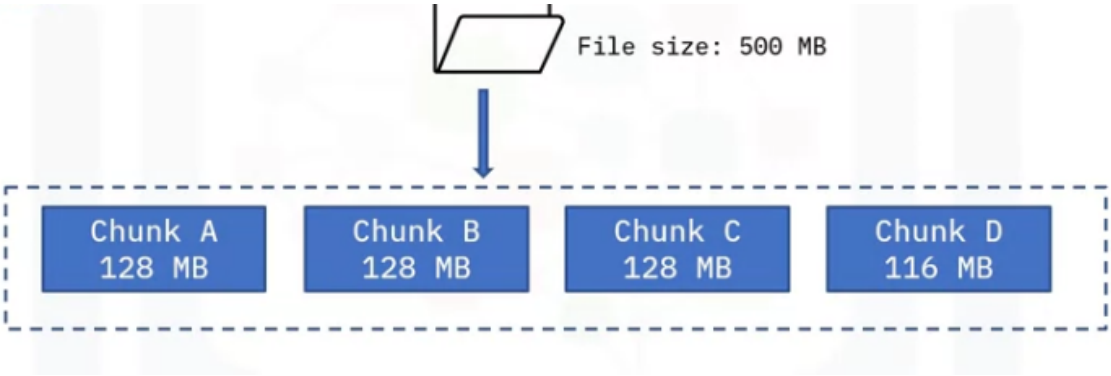
Key features

- cost efficient
- large amounts of data
- replication
- fault tolerant
- scalable
- portable

Blocks

- minimum amount of data that can be read or written

- provides fault tolerance
- default size is 64MB or 128 MB



- each file stored doesn't have to take up the configures space size

Nodes

- a node is a single system which is responisble to store and process data
- primary node = name node
  - regulates file access to the clients and mantains, manages and assigns tasks to the secondary node
- secondary node = data node
  - the actual workers
  - take instructions from the primary
- rack awareness in HDFS
  - choosing data node racks that are closest to each other
  - improves cluster preformance by reducing network traffic
  - name node keeps the rack ID information
  - replication can be done through rack awareness
  - (a rack is the collection of about 40 to 50 data nodes using the same network switch)

Replication

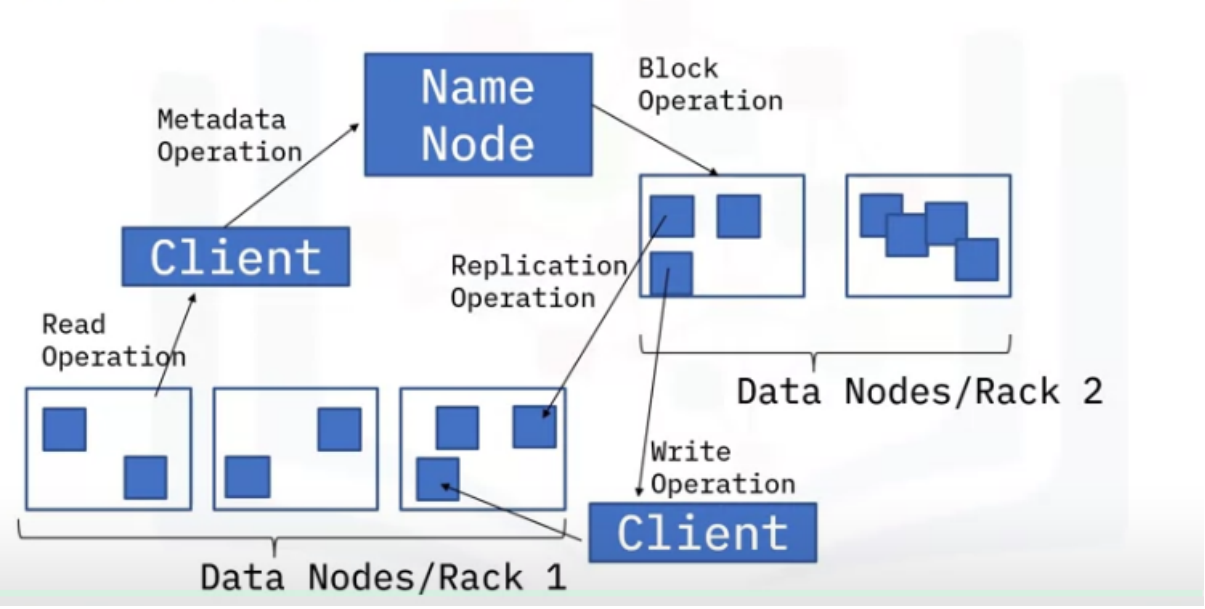
- creating a copy of the data block
- copies are created for backup purposes

- replication factor: number of times of the data block was copied

Read and write

- allows write once ready many operations
- read
  - client will send a request to the primary node to get the location of the data nodes containing blocks
  - client will read files closest to the data nodes
  - a client fulfills a user's request by interacting with the name and data nodes
- write
  - name node makes sure that te file doesnt exist
  - if the file exists client gets an IO Exception message
  - if the file doesnt exist, the client is given access to start writing files

HDFS architecture



Hive

- data warehouse software within hadoop that is designed for reading, writing and managing tabular-type datasets and data analysis
- scalable, fast and easy to use

- Hive query language (HiveQL) in inspired by SQL
- supports data cleansing and filtering depending on users' requirements
- file formats supported:
  - flat and text files
  - sequence file (binary key value pairs)
  - record columnar files (columns of a table stored in a columnar database)

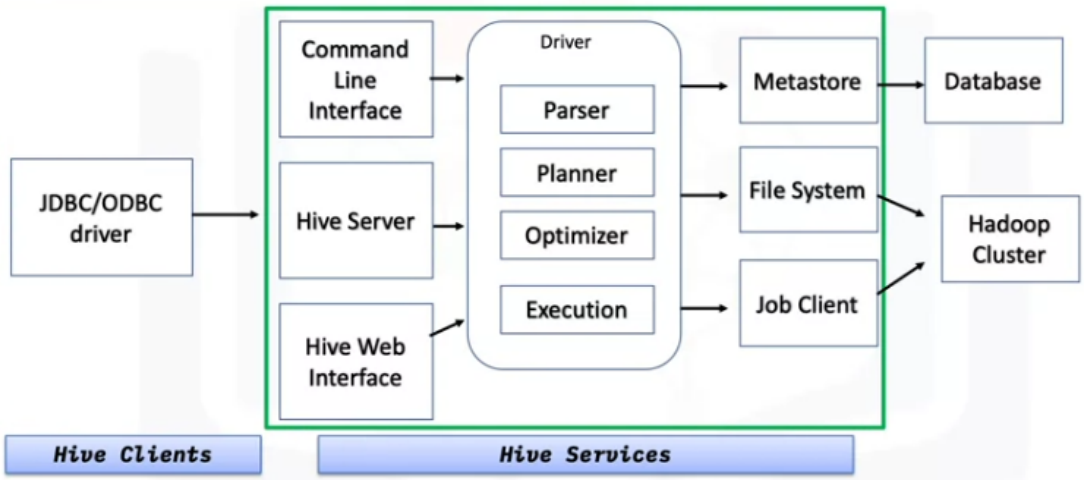
Traditional RDBMS

- used to mantain a database and uses SQL
- suited for real-time/dynamic data analysis like data from sensors
- designed to read and write as many times as it needs
- maximum data size it can handle is terabytes
- enforces that the schema must verify loading data before it can proceed
- may not always have built-in for support data partitioning

Hive

- used to mantain a data warehouse using hive query language
- suited for static data analysis like a text file containing names
- designed on the methodology of wwrite once, read many
- maximum data size it can handle is petabytes
- doesnt enforce the schema to verify loading data
- supports partitioning

Hive architecture



- Hive clients
  - JDBC client allows java apps to connect to hive
  - ODBC client allows apps based on ODBC protocol to connect to hive
- Hive services
  - hive server to enable queries
  - the driver recieves query statements
  - the optimizer is used to split tasks efficiently
  - the executor executes tasks after the optimizer
  - metastore stores the metadata information about the tables

Apache HBASE

- column-oriented non-relational database management system
- runs on top of hdfs
- provides a fault-tolerant way of storing sparse datasets
- works well with real-time data and random read and write access to big data
- used for write-heavy applications
- linearly and modularly scalable
- backup support for MapReduce
- provides consistent reads and writes
- no fixed column schema

- easy-to-use java api for client access
- provides data replication across clusters
- predefine table schema and specify column families
- new columns can be added to column families at any time
- schema is very flexible
- has a master node to manage the cluster and region servers to perform the work

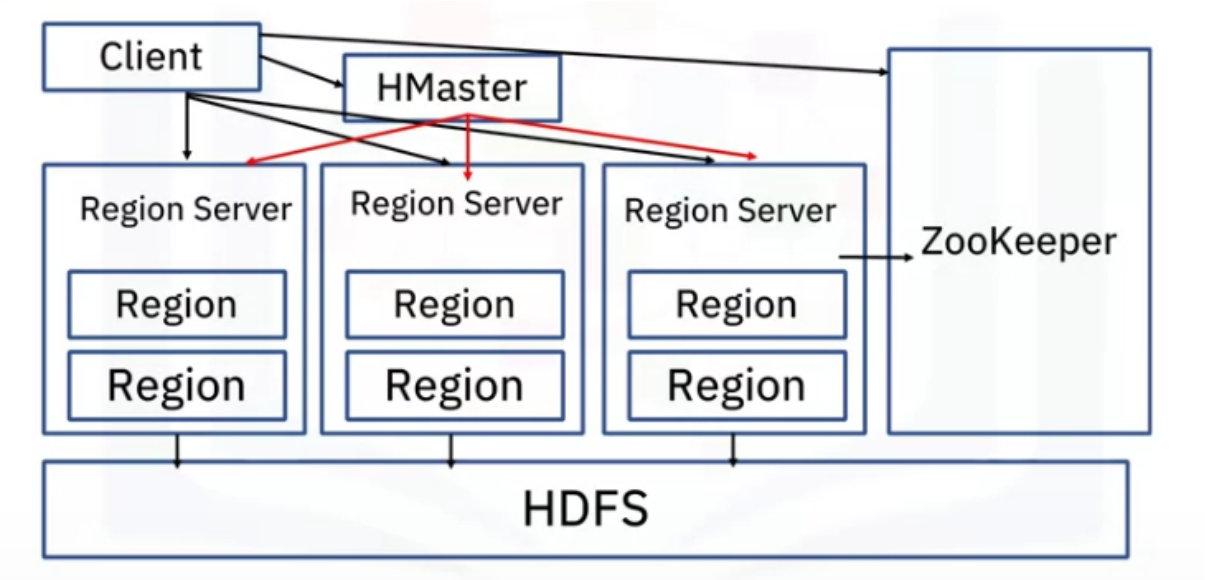
HBase

- stores data in the form of columns and rows in a table
- allows dynamic changes
- suitable for random writes and reads of data stored in HDFS
- allows for storing and processing of big data

HDFS

- stores data in a distributed manner across different nodes on that network
- has a rigid architecture that doesnt allow changes
- suited for write once and read many times
- for storing only

HBase architecture



Concepts

- HMaster
  - monitors the region server instances
  - assigns regions to region servers
  - manages any changes that are made to the schema
- Region Servers
  - communicates directly with the client
  - recieves and assigns requests to regions
  - responsible for managing regions
  - communicates directly with the client
- Region
  - smallest unit of HBase cluster
  - contains multiple stores
  - two componentes:
    - HFile
    - Memstore
- Zookeeper
  - mantains healthy links between nodes
  - provides distributed sync
  - tracks server failure

Hands-on Lab: Hadoop MapReduce

The steps outlined in this lab use the Dockerized single-node Hadoop Version 3.2.1. **Hadoop** is most useful when deployed in a fully distributed mode on a large cluster of networked servers sharing a large volume of data. However, for basic understanding, we will configure Hadoop on a single node.

```
#clone the repository
git clone https://github.com/ibm-developer-skills-network/ooxwv-docker_hadoop.git

#Compose the docker application.<
#Compose is a tool for defining and running multi-container Docker applications. It us
es the YAML file to configure the serives and enables us to create and start all the s
```

```
ervices from just one configurtation file.
docker-compose up -d


#Run the namenode as a mounted drive on bash.
docker exec -it namenode /bin/bash
```

Hadoop enviroment is configured by editing a set of configuration files:

- hadoop-env.sh: serves as a master file to configure YARN, HDFS, MapReduce, and Hadoop-related project settings
- core-site.xml: defines HDFS and Hadoop core properties
- hdfs-site.xml: governs the location for storing node metadata, fsimage file and log file
- mapred-site.xml: lists the parameters to MapReduce configuration
- yarn-site.xml: defines settings relevant toYARN. It contains configurations for the Node Manager, Resource Manager, Containers, and Application Master.

```
#For the docker image, these xml files have been configured already. You can see these
in the directory /opt/hadoop-3.2.1/etc/hadoop/ by running
ls /opt/hadoop-3.2.1/etc/hadoop/*.xml
```

Set up for MapReduce

```
#In the HDFS, create a directory named user.
hdfs dfs -mkdir /user

#In the HDFS, under user, create a directory named root.
hdfs dfs -mkdir /user/root

#Under /user/root, create an input directory.
hdfs dfs -mkdir /user/root/input

#Copy all the hadoop configuration xml files into the input directory.
hdfs dfs -put $HADOOP_HOME/etc/hadoop/*.xml /user/root/input

#Create a data.txt file in the current directory.
curl https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-BD0225EN-S
killsNetwork/labs/data/data.txt > data.txt

#Copy the data.txt into the /user/root directory to pass it into the wordcount proble
m.
hdfs dfs -put data.txt /user/root

#Check if the file has been copied into the HDFS by viewing its content.
hdfs dfs -cat /user/root/data.txt
```

MapReduce word count

```
#Run the Map reduce application for wordcount on data.txt and store the output in /use
r/root/output
hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.2.1.jar wor
dcount data.txt /user/root/output

#Once the word count runs successfully, you can run the following command to see the o
utput file it has generated.
hdfs dfs -ls /user/root/output/
#While it is still processing, you may only see '_temporary' listed in the output dire
ctory. Wait for a couple of minutes and run the command again till you see output as s
hown in the image above.

#Run the following command to see the word count output.
hdfs dfs -cat  /user/root/output/part-r-00000
```

Summary & Highlights

- Hadoop is an open-source framework for Big Data that faced challenges when encountering dependencies and low-level latency.
- MapReduce, a parallel computing framework used in parallel computing, is flexible for all data types, addresses parallel processing needs for multiple industries and contains two major tasks, “map” and “reduce.”
- The four main stages of the Hadoop Ecosystem are Ingest, Store, Process and Analyze, and Access.
- Key HDFS benefits include its cost efficiency, scalability, data storage expansion and data replication capabilities. Rack awareness helps reduce the network traffic and improve cluster performance. HDFS enables “write once, read many” operations.
- Suited for static data analysis and built to handle petabytes of data, Hive is a data warehouse software for reading, writing, and managing datasets. Hive is based on the “write once, read many” methodology, doesn’t enforce the schema to verify loading data and has built-in partitioning support.
- Linearly scalable and highly efficient, HBase is a column-oriented non-relational database management system that runs on HDFS and provides an easy-to-use Java API for client access. HBase architecture consists of HMaster, Region servers, Region, Zookeeper and HDFS. A key difference between HDFS and

HBase is that HBase allows dynamic changes compared to the rigid architecture of HDFS.

Semana 3

Why use Apache Spark?

- open source in-memory application framework for distributed data processing and iterative analysis on massive data volumes
- written in Scala
- runs in java virtual machines
- distributed computing
- easy-to-use python, scala and java APIs

Different!!

→ PARALLEL COMPUTING processors access shared memory

→ DISTRIBUTED COMPUTING processors usually have their own private or distributed memory

Distributed computing benefits

- scalability and modular growth
- fault tolerance and redundancy

Apache Spark

- create MapReduce jobs for complex jobs, interactive query, and online event-hub processing involves lots of (slow) disk I/O

MapReduce

- keep more data i**n-memory** with a new distributed execution engine

Data engineering

- core spark engine
- clusters and executors
- cluster managment
- sparkSQL
- catalyst tungsten dataFrames

Data science and machine learning

- sparkML
- dataFrames
- streaming

Parallel Programming using Resilient Distributed Datasets

- spark parallelizes computations using the lambda calculus (functional programming)

RDDs

A resilient distributed dataset is:

- spark's primary data abstraction
- a fault-tolerant collection of elements
- partitioned across the nodes of the cluster
- capable of accepting parallel operations
- immutable

Supported file types:

- text
- sequenceFiles
- Avro
- Parquet
- Hadoop input formats

Supported file formats:

- local
- cassandra

- HBase
- amazon S3
- others
- sql and nosql databases

→ Spark applications consist of a **driver program** that runs the user's main functions and multiple **parallel operations** on a cluster

Creating an RDD in Spark

Option 1: use an external or local file from hadoop-supported file system such as

- HDFS
- cassandra
- HBase
- amazon s3

Option 2: create from a list in Scala or Python using the APIs

```
data = [1, 2, 3, 4]
distData = sc.parallelize(data)
```

Option 3: apply a transformation on an existing RDD to create a new RDD

Parallel programming

- simultaneous use of multiple compute resources to solve a computational problem
- breaks problems into discrete parts that can be solved concurrently
- runs simultaneous instructions on multiple processors
- employs an overall control/coordination mechanism

- you can create an RDD by parallelizing an array of objects, or by splitting a dataset into partitions

- spark runs one task for each partition of the cluster

Resilient distributed datasets

- are always recoverable as they are immutable
- can persist or cache datasets in memory across operations, which speeds iterative operations
- persistence and cache

Scale out / Data Parallelism in Apache Spark

Components

DATA STORAGE HDFS and other formats

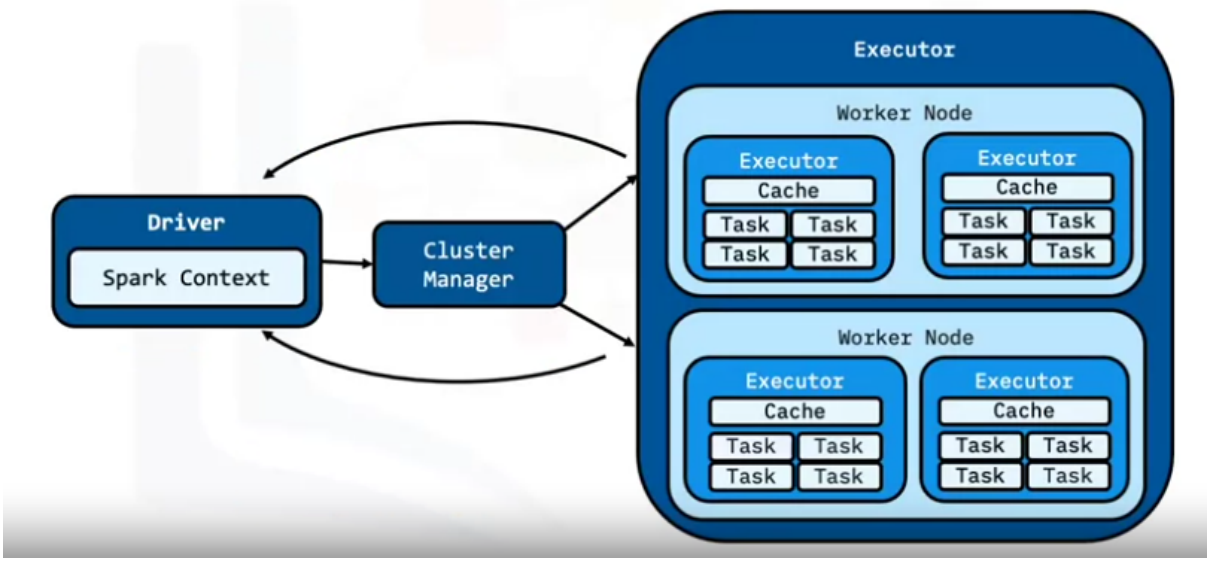
COMPUTE INTERFACE. APIs: scala, java and python

MANAGEMENT. Distributed computing Standalone, Mesos, YARN and Kubernetes

Spark core

- is a base engine
- is fault-tolerant
- performs large scale parallel and distributed data processing
- manages memory
- sschedules tasks
- houses APIs that define RDDs
- contains a distributed collection of elements that are parallelized across the cluster

Scaling big data in Spark



- The Spark Application consists of the driver program and the executor program. Executor programs run on worker nodes.
- Spark can start additional executor processes on a worker if there is enough memory and cores available. Similarly, executors can also take multiple cores for multithreaded calculations. Spark distributes RDDs among executors.
- Communication occurs among the driver and the executors.
- The driver contains the Spark jobs that the application needs to run and splits the jobs into tasks submitted to the executors. The driver receives the task results when the executors complete the tasks. If Apache Spark were a large organization or company, the driver code would be the executive management of that company that makes decisions about allocating work, obtaining capital, and more. The junior employees are the executors who do the jobs assigned to them with the resources provided.
- The worker nodes correspond to the physical office space that the employees occupy. You can add additional worker nodes to scale big data processing incrementally.

Dataframes and SparkSQL

- sparksql is a spark module for **STRUCTURED** DATA PROCESSING
- used to query structured data inside spark programs, using either sql or a familiar DataFrame API
- usable in java, scala, python and R

- runs SQL queries over imported data and existing RDDs independently of API or programming language

```
#example sql query in python

results = spark.sql("SELECT * FROM people")

names = results.map(lambda p: p.name)
```

Benefits of SparkSQL:

- includes a cost-based optimizer, columnar storage, and code generation to make queries fast
- scales to thousands of nodes and multi-hour queries using the spark engine, which provides full mid-query fault tolerance
- provides a programming abstraction called **DataFrames** and can also act as a distributed SQL query engine

DataFrames:

- Distributed collection of data organized into named columns
- conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations
- built on top of the RDD API
- uses RDD
- performs relational queries
- benefits
    - scale from KBs on a single laptop to petabytes on a large cluster
    - support for a wide array of data formats and storage systems
    - state-of-the-art optimization and code generation through the spark sql catalyst optimizer
    - seamless integration with all big data tooling and infrastrcture via spark
    - APIs for python, java, scala, and R, which is in development via spark R

```
#read from json file and create dataframe
df = spark.read.json("people.json")
df.show()
df.printSchema()

#register the dataframe as SQL temporary vire
df.createTempView("people")
```

```
#sql query
spark.sql("SELECT * FROM people").show()

#same with dataframe
df.select("name").show()
df.select(df["name"]).show()
```

```
#sql query
spark.sql("SELECT age, name FROM people WHERE age > 21").show()

#same with dataframe python api
df.filter(df["age"]>21).show()
```

LAB → notebook sparkintro

When running the second cell I had this error:

> "Couldn't find Spark, make sure SPARK_HOME env is set"
> " or Spark is in an expected location (e.g. from homebrew installation)."

This line should get everything done and installed but it doesnt:

```
import findspark
findspark.init()
```

To solve it:

1. install spark from the web page: https://www.apache.org/dyn/closer.lua/spark/spark-3.2.0/spark-3.2.0-bin-hadoop3.2.tgz

2. Set enviroment variable. In this case for Linux (fedora)
   Theres a hidden file in home directory called .bashrc
   There you need to add this line:

   SPARK_HOME="/path/spark-3.2.0-bin-hadoop3.2"

4. Reboot
5. Check the enviroment variable has been created:

```
printenv SPARK_HOME
```

5. Problem solved

Summary & Highlights

- Spark is an open source in-memory application framework for distributed data processing and iterative analysis on massive data volumes. Both distributed systems and Apache Spark are inherently scalable and fault tolerant. Apache Spark solves the problems encountered with MapReduce by keeping a substantial portion of the data required in-memory, avoiding expensive and time-consuming disk I/O.
- Functional programming follows a declarative programming model that emphasizes “what” instead of “how to” and uses expressions.
- Lambda functions or operators are anonymous functions that enable functional programming. Spark parallelizes computations using the lambda calculus and all functional Spark programs are inherently parallel.
- Resilient distributed datasets, or RDDs, are Spark’s primary data abstraction consisting of a fault-tolerant collection of elements partitioned across the nodes of the cluster, capable of accepting parallel operations.You can create an RDD using an external or local Hadoop-supported file, from a collection, or from another RDD. RDDs are immutable and always recoverable, providing resilience in Apache Spark RDDs can persist or cache datasets in memory across operations, which speeds iterative operations in Spark.
- Apache Spark architecture consists of components data, compute input, and management. The fault-tolerant Spark Core base engine performs large-scale Big Data worthy parallel and distributed data processing jobs, manages memory, schedules tasks, and houses APIs that define RDDs.

- Spark SQL provides a programming abstraction called DataFrames and can also act as a distributed SQL query engine. Spark DataFrames are conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations.

Practice quiz

1. Benefits of working with Spark:
   it is an open-source, in-memory application framework for distributed data processing
2. What are the features or characteristics of a functional programming language such as Scala?
   - it treats functions as first-class citizens, for example functions can be passed as arguments to other functions
   - it follows a declarative programming model
   - the emphasis is in the "what" not in the "how to"
3. Which of the following statements are true of Resilient Distributed Datasets (RDDs)?
   - An RDD is a distributed collection of elements parallelized across the cluster
   - RDDs are persistent and speed up interactions because they can reuse the same patition in other actions on a dataset
   - RDDs enable Apache Spark to reconstruct transformations
4. How is the Spark Application Architecture configured to scale big data?
   - spark application consists of the driver program and the executor program
   - executor programs run on worker nodes and you can add additional worker nodes to scale big data processing incrementally.
   - apache spark architecture consists of three main pieces. components data, compute input and management
5. Which SQL query options would display the names column from this DataFrame example?
   - df.select(df["name"]).show()
   - spark.sql("SELECT name FROM people").show()
   - df.select("name").show()

Semana 4

RDDs in Parallel Programming and Spark

- RDDs are spark's primary data abstraction
- they are partitioned across the nodes of the cluster
  - dataset is partitioned
  - partitions are stored in worker memory

Transformations

- create a new RDD from existing one
- are "lazy" because the results are only computed when evaluated by actions
  - the map transformation passes each element of a dataset through a function and returns a new RDD

Actions

- actions return a value to driver program after running a computation
  - reduce()
    - an action that aggregates all RDD elements

Directed Acyclyc Graph (DAG)

How do transformations and actions happen?

Spark uses a unique data structure called DAG and an associated DAG Schedular to perform RDD operations.

- it's a graphical data structure with edges and vertices
- a new edge is generated only from an existing vertex
- in apache spark DAG, the vertices represent RDDs and the edges represent operations such as transformations or actions
- if a noed goes down, spark replicated the DAG and restores the node

Transformations and actions

1. Spark creates the DAG when creating an RDD
2. Spark enables the DAG Schedular to perform a transformation and updates the DAG

3. The DAG now points to the new RDD
4. The pointer that transforms RDD is returned to the Spark driver program
5. If there is an action, the driver program that calls the action evaluates the DAG only after Spark completes the action

Transformation	Description
map (*func*)	Returns a new distributed dataset formed by passing each element of the source through a function *func*
filter (*func*)	Returns a new dataset formed by selecting those
distinct ([numTasks]))	Returns a new dataset that contains the distinct elements of the source dataset
flatmap (*func*)	Similar to map (*func*) Can map each input item to zero or more output items *Func* should return a Seq rather than a single item

Action	Description
reduce(*func*) aggregates dataset elements using the function *func*	*func* takes two arguments and returns one • Is commutative • Is associative • Can be computed correctly in parallel
take(*n*)	Returns an array with the first *n* element
collect()	Returns all the elements as an array WARNING: Make sure that ? will fit in driver program
takeOrdered (*n*,key=*func*)	Returns *n* elements ordered in ascending order or as specified by the optional key function

- A transformation is only a map operation. We need actions to return computed values to the driver program

Data-frames and Datasets

- DATASETS
  - provide an API to access a distributed data collection

- collection of strongly typed Java Virtual Machine objects
- provides the combined benefits of both RDDs and Spark SQL
- features:
  - immutable, cannot be deleted or lost
  - encoder that converts JVM objects to a tabular representation
  - extend DataFrame type-safe and object-orientes API capabilities
  - work with both Scala and Java APIs
- dynamically typed languages, such as Python and R, do NOT support dataset APIs
- benefits
  - provide compile-time type safety
  - compute faster than RDDs
  - offer the benefits of Spark SQL and DataFrames
  - optimize queries using Catalyst and Tungsten
  - enable improved memory usage and caching
  - use dataset API functions for aggregate operations including sum, average, join and group by.
- creating a dataset in Scala

```
val ds = Seq("Alpha", "Beta", "Gamma").toDS()

//from text file
val ds = spark.read.text("/text_folder/file.txt").as[String]

case class Customer(name: String, id: Int, phone: Double)
val ds_cust = spark.read.json("/customer.json").as[Customer]
```

-

DataFrames

- not typesafe
- use APIs in Java, Scala, Python and R

- built on top of RDDs and added in earlier Spark versions

Datasets

- strongly-typed
- use unified Java and Scala APIs
- built on top of DataFrames and the latest data abstraction added to Spark

Catalyst and Tungsten

Goals of Spark SQL optimization:

- reduce query time
- reduce memory consumption
- save organizations time and money

Catalyst

- spark SQL buil-in rule-based query optimizer
- based on functional programming constructs in Scala
- supports the addition of new optimization techniques and features
- enables developers to add data source-specific rules and support new data types
- tree data structure and a set of rules
    - four major phases of query execution
        - analysis
            - catalyst analizes the query, the DataFrame, the unresolved logicical plan and the Catalog to create a logical plan
        - logistical optimization
            - the logical plan evolves into an Optimized Logical Plan. This is the rule-based optimization step of Spark SQL and rules such as folding, pushdown and pruning are applied here
        - physical planning

        - describes computation on datasets with specific definitions on how to conduct the computation. A cost model then chooses the physical plan with the least cost. This is the cost-based optimization step.
    - code generation
        - Catalyst applies the selected physical plan and generates Java nytecode ro run on each node.

Rule-based optimization → defines how to run the query

is the table indexed?
does the query contain only the required columns?

Cost-based optimizaion → equals time + memory a query consumes

what are the best paths for multiple datasets to use when querying data?

Tungsten

- Spark's cost-based optimizer that maximizes CPU and memory performance
- features
    - manages memory explicitly and does not rely on the JVM object model or garbage collection
    - enables cache-friendly computation of alorithms and data structures using both STRIDE-based memory access
    - supports on-demand JVM byte code generation
    - does not generate virtual function dispatches
    - places intermediate data in CPU registers

ETL with DataFrames

- basic DF operations
    - read the data

    - analyze the data
    - transform the data
    - load data into database
    - write data back to disk
- iThere's also ELT
    - here the data resides in data lakes
    - some companies use a mixture between both ETL and ELT

Read the data

- create a dataframe
- create a dataframe from an exissting dataframe

```
import pandas as pd
mtcars = pd.read_csv('mtcars.csv')
sdf = spark.createDataFrame(mtcars)
```

Analyze the data

- view the schema and take note of data types

```
sdf.printSchema()

sdf.show(5)

sdf.select('mpg').show(5)
```

Transform the data

- keep only relevant data
- apply filters, joins, sources and tables, column operations, grouping and aggregations and other functions
- apply domain-specific data augmentation processes

```
sdf.filter( sdf['mpg'] < 18 ).show(5) #mpg is a column

car_counts = sdf.groupby(['cyl']).agg({"wt": "count"})\.sort("count(wt)", ascending=Fa
lse).show(5)
```

Loading or exporting the data

- final step of ETL pipeline
- export to another database
- export to disk as JSON files
- save the data to a Postgres database
- use an API to export data

Real-world usage of SparkSQL

SparkSQL:

- spark module for structured data processing
- runs SQL queries on Spark DataFrames
- usable in java, scala, python and R

First step: create a table view

- creating a table view is required to run SQL queries programmatically on a DataFrame
- a view is a temporary table to run SQL queries
  - a tempoerary view provides local scope within the current spark session.
  - a global temporary view provides global scope within the spark application. Useful for sharing

```
df = spark.read.json("people.json")

df.createTempView("people")
spark.sql("SELECT * FROM people").show()

df.createGlobalTempView("people")
spark.sql("SELECT * FROM global_temp.people").show()
```

Aggregating data

- DataFrames contain inbuilt common aggregation functions
  - count()
  - countDistinct()
  - avg()
  - max()
  - min()
  - others
- alternatively, aggregate using SQL queries and tableviews

```
import pandas as pd

mtcars = pd.read_csv("mtcars.csv")
sdf = spark.createDataFrame(mtcars)

sdf.select('mpg').show(5)

#option 1
car_counts = sdf.groupby(['cyl']).agg({"wt": "count"})\.sort("count(wt)", ascending=Fa
lse).show(5)
#option 2
sdf.createTempView("cars")
sql("SELECT cyl, COUNT(*) FROM cars GROUPBY cyl ORDER BY 2 DESC")
```

Data sources supported

- parquet files
  - supports reading and writing, and preserving data schema
  - spark sql can also run queries without loading the file
- JSON datasets
  - spark infers the schema and loads the dataset as a DataFrame
- Hive tables
  - spark supports reading and writing data stored in Apache Hive

Summary & Highlights

- RDDs are Spark's primary data abstraction partitioned across the nodes of the cluster. Transformations leave existing RDDs intact and create new RDDs based on the transformation function. With a variety of available options, apply functions to transformations perform operations. Next, actions return computed values to the driver program. Transformations undergo lazy evaluation, meaning they are only evaluated when the driver function calls an action.
- A dataset is a distributed collection of data that provides the combined benefits of both RDDs and SparkSQL. Consisting of strongly typed JVM objects, datasets make use of DataFrame typesafe capabilities and extend object-oriented API capabilities. Datasets work with both Scala and Java APIs. DataFrames are not typesafe. You can use APIs in Java, Scala, Python. Datasets are Spark's latest data abstraction.
- The primary goal of Spark SQL Optimization is to improve the run-time performance of a SQL query, by reducing the query’s time and memory consumption, saving organizations time and money. Catalyst is the Spark SQL built-in rule-based *query* optimizer. Catalyst performs analysis, logical optimization, physical planning, and code generation. Tungsten is the Spark built-in cost-based optimizer for CPU and memory usage that enables cache-friendly computation of algorithms and data structures.
- Basic DataFrame operations are reading, analysis, transformation, loading, and writing. You can use a Pandas DataFrame in Python to load a dataset and apply the print schema, select function, or show function for data analysis. For transform tasks, keep only relevant data and apply functions such as filters, joins, column operations, grouping and aggregations, and other functions.
- Spark SQL consists of Spark modules for structured data processing that can run SQL queries on Spark DataFrames and are usable in Java, Scala, Python and R. Spark SQL supports both temporary views and global temporary views. Use a DataFrame function or an SQL Query + Table View for data aggregation. Spark SQL supports Parquet files, JSON datasets and Hive tables.

Semana 5

Apache Spark Architecture

- **two main processes**

- driver program
  - The driver program runs as one process per application.
  - The driver process can be run on a cluster node or another machine as a client to the cluster. The driver runs the application’s user code,
  - creates work and sends it to the cluster.
- executors
  - work independently
  - There can be many throughout a cluster and one or more per node, depending on configuration.

- **Spark context**
  - The Spark Context starts when the application launches and must be created in the driver

  before DataFrames or RDDs. Any DataFrames or RDDs created under the context are tied to it and the context must remain active for the life of them. The driver program creates work from the user code called “Jobs” (or computations that can be performed in parallel). The Spark Context in the driver divides the jobs into tasks to be executed on the cluster. Tasks from a given job operate on different data subsets, called Partitions. This means tasks can run in parallel in the Executors. A Spark Worker is a cluster node that performs work.

- **Spark executor**
  - A Spark Executor utilizes a set portion of local resources as memory and compute cores, running one task per available core. Each executor manages its data caching as dictated by the driver. In general, increasing executors and available cores increases the cluster’s parallelism. Tasks run in separate threads until all cores are used. When a task finishes, the executor puts the results in a new RDD partition or transfers them back to the driver. Ideally, limit utilized cores to total cores available per node. For instance, an 8-core node could have 1 executor process using 8 cores.

- **Stage**
  - A “stage” in a Spark job represents a set of tasks an executor can complete on the

  current data partition. When a task requires other data partitions, Spark must perform a “shuffle." A shuffle marks the boundary between stages. Subsequent

tasks in later stages must wait for that stage to be completed before beginning execution, creating a dependency from one stage to the next. Shuffles are costly as they require data serialization, disk and network I/O. This is because they enable tasks to “pass over” other dataset partitions outside the current partition. An example would be a “groupby” with a given key that requires scanning each partition to find matching records. When Spark performs a shuffle, it redistributes the dataset across the cluster. This example shows two stages separated by a shuffle. In Stage 1, a ransformation (such as a map) is applied on dataset “a” which has 2 partitions (“1a” and “2b”). This creates data set “b”. The next operation requires a shuffle (such as a “groupby”). Key values could exist in any other partition, so to group keys of equal value together, tasks must scan each partition to pick out the matching records. Transformation results are placed in Stage 2. Here results have the same number of partitions, but this depends on the operation.

Final results are sent to the driver program as an action, such as collect. NOTE: It is not advised to perform a collection to the driver on a large data set as it could easily use up the driver process memory. If the data set is large, apply a reduction before collection.

Overview of Apache Spark Cluster Modes

The Spark Cluster Manager communicates with a cluster to acquire resources for an application to run. It runs as a service outside the application and abstracts the cluster type. While an application is running, the Spark Context creates tasks and communicates to the cluster manager what resources are needed. Then the cluster manager reserves executor cores and memory resources. Once the resources are reserved, tasks can be transferred to the executor processes to run

Spark has built-in support for several cluster managers:

Standalone manager is included

- no additional dependencies required
- two main componentes:
    - workers
        - run on cluster nodes. They start an executor process with one or more reserved nores

    - master
        - There must be one master available which can run on any cluster node. It connects workers to the cluster and keeps track of them with heartbeat polling. However, if the master is together with a worker, do not reserve all the node’s cores and memory for the worker.

To manually set up a Spark Standalone cluster, start the Master. The Master is assigned a URL with a hostname and port number. After the master is up, you can use the Master URL to start workers on any node using bi-directional communication with the master. Once the master and the workers are running, you can launch a Spark application on the cluster by specifying the master URL as an argument to connect them.

Apache Hadoop YARN

- general-purpose cluster manager
- supports other frameworks besides spark
- have their own dependencies
- To run Spark on an existing YARN cluster, use the ‘--master’ option with the keyword YARN.

Apache mesos

- general-purpose cluster manager
- dynamic partitioning between Spark and other big data frameworks and scalable partitioning between multiple Spark instances
- may require additional set up

Kubernetes

- runs containerized applications
- This makes Spark applications more portable and helps automate deployment, simplify dependency management and scale the cluster as needed.
- launch application on kubernetes
  ./bin/spark-submit \

- -master k8s://https://<k8s-apiserver-host>:<k8s-apiserver-port> \
<additional configuration>

Local mode

- spark can also be run in local mode
- not connect to cluster, easy to get started
- runs in the same process that calls sparl-submit
- uses threads for running executor tasks
- useful for testing
- runs locally within a single process which can limit performance
- to run:
  #launch spark in local mode with 8 cores
  ./bin/spark-submit \
  - -master local[8]
  <additional configuration>

How to Run an Apache Spark Application

Spark submit

- Unified interface for submitting applications
- found in the bin/ directory
- easily switches from local to cluster

1. parse command line arguments and options
2. read additional configuration specified in 'conf/spark-defaults.conf'
3. connect to the cluster manager specified with the '- -master' argument or run in local mode
4. transfer applications (JARs or python files) and any additional files specified to be distributed and run in the cluster

Option/Setting	Form	Mandatory ?
Tell Spark what cluster manager to connect with	'--master'	Yes
Specify the program entry point if using Java or Scala application	`--class <full-class-name>`	Yes
Set how the driver is deployed (Client or Cluster). Default is Client mode	'deploy-mode'	No
Set CPU core and memory usage in executors	`--executor-cores` and `--executor-memory`	No
See available options by cluster manager	`./bin/spark-submit --help`	N/A

Example launch python SparkPi to a Spark Standalone cluster. Estimate Pi with 1000 samples:

```
./bin/spark-submit \
--master spark://207.184.161.138:7077 \
examples/src/main/python/pi.py
1000
```

Spark shell

- simple way to learn spark api
- powerful tool to analyze data interactively
- use in local mode or with a cluster
- can initiate in scala and python
- enviroment
  - SparkContext is automaticallt initialized and is available as 'sc'
  - SparkSession is automatically available as 'spark'
  - expressions are entered in the shell and then evaluated in the driver to become jobds that are scheduled as tasks for the cluster

```
#spark shell example in scala
val df = spark.range(10)
df: org.apache.spark.sql.Dataset[Long] = [id: bigint]

df.withColumn("mod", expr("id % 2")).show(4)
```

Summary & Highlights

- Spark Architecture has driver and executor processes, coordinated by the Spark Context in the Driver.
- The Driver creates jobs and the Spark Context splits jobs into tasks which can be run in parallel in the executors on the cluster. Stages are a set of tasks that are separated by a data shuffle. Shuffles are costly, as they require data serialization, disk and network I/O. The driver program can be run in either client Mode (connecting the driver outside the cluster) or cluster mode (running the driver in the cluster).
- Cluster managers acquire resources and run as an abstracted service outside the application. Spark can run on Spark Standalone, Apache Hadoop YARN, Apache Mesos or Kubernetes cluster managers, with specific set-up requirements. Choosing a cluster manager depends on your data ecosystem and factors such as ease of configuration, portability, deployment, or data partitioning needs. Spark can also run using local mode, which is useful for testing or debugging an application.
- 'spark-submit' is a unified interface to submit the Spark application, no matter the cluster manager or application language. Mandatory options include telling Spark which cluster manager to connect to; other options set driver deploy mode or executor resourcing. To manage dependencies, application projects or libraries must be accessible for driver and executor processes, for example by creating a Java or Scala uber-JAR. Spark Shell simplifies working with data by automatically initializing the SparkContext and SparkSession variables and providing Spark API access.

Using Apache Spark on IBM Cloud

Why use Spark on IBM Cloud

- cloud benefits
  - streamline deployment with less configuration
  - easily scale up to increase compute power
  - enterprise grade security

- tie into existing IBM big data solutions for AIOps and apps for IBM Watson and IBM Analytics Engine

What is AIOps

- the application of artificial intelligence to automate or enhance IT operations
- helps collect, aggregate and work with large volymes of operations data
- helps identify events and patterns in infrastructure systems
- diagnose root causes and report or fix them automatically

IBM Spectrum Conductor

- run multiple spark apps and versions together, on a single large cluster
- manage and share cluster resources as needed
- provide enterprise grade security

Setting Apache Spark Configuration

Configuration types:

- Properties: adjust and control app behavior
- Enviroment variables: adjust settings on a per-machine basis
- Logging: control how logging is output using 'conf/log4j-dafaults.properties'

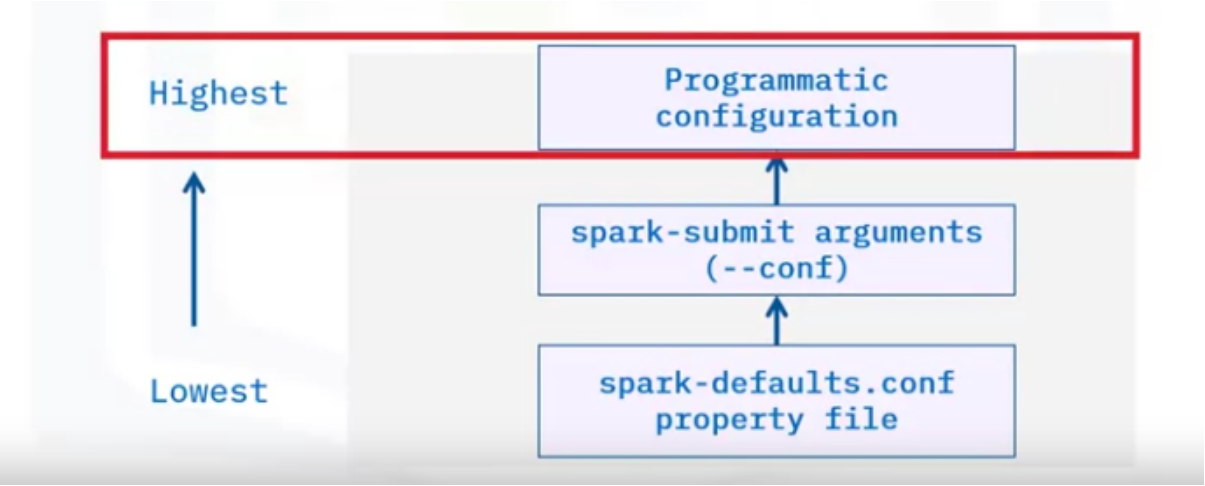
Configuration Type	Template File	Actual File
Spark properties	spark-defaults.conf.template	spark-defaults.conf
Environment variables	spark-env.sh.template	spark-env.sh
Logging properties	log4j.properties.template	log4j.properties

You can set properties:

1. Programmatically when creating SparkSession or using a SparkConf object
2. In the file `conf/spark-defaults.conf`
3. When launching `spark-submit` with arguments `--master`, or `--conf <key>=<value>`

```
# Set the master and additional conf when creating session
spark = SparkSession\
    .builder\
    .master("spark://<master-url>:7077")\
    .config("<key>", "<value>")\
    .getOrCreate()
```

Property precedence: (the order is from highest to lowest)



Static configuration usually doesnt change because it would be needed to change the app

Dynamic configuration avoids hard-coding values such as number of cored or reserved memory

Enviromental variables loaded from: conf/spark-env.sh

- common usage: ensure all cluster nodes use the same python version PYSPARK_PYTHON enviroment variable

Running Spark on Kubernetes

Also abreviated as **k8s** runs containerized applications on a cluster and is

- open source
- highly scalable
- provides flexible, automated deployments
- protable, so can be run in the same way wether in the cloud or on-premises

Use to manage containers that run distributed systems in a more resilient and flexible way, with benefits including:

- network service discovery
- clsuter load balancing
- automated scale up and down
- orchestrating storage

Local Kubernets cluster on a machine using tools as **minikube**

Running Spark on Kubernetes

- containerization
- better resource sharing
    - multiple spark apps can run concurrently and in isloation

```
./bin/spark-submit \
--master k8s://https://<k82-apiserver-host>:<k82-apiserver-port> \
--deploy-mode client \
--class <application-main-class>
--conf spark.kubernetes.container.image=<spark-image> \
--conf spark.kubernetes.dirver.pod.name=<pod-name> \
local:///path/to/application.jar
```

Summary & Highlights

- Running Spark on IBM Cloud provides enterprise security and easily ties in IBM big data solutions for AIOps, IBM Watson and IBM Analytics Engine. Spark’s big

data processing capabilities work well with AIOps tools, using machine learning to identify events or patterns and help report or fix issues. IBM Spectrum Conductor manages and deploys Spark resources dynamically on a single cluster and provides enterprise security. IBM Watson helps you focus on Spark's machine learning capabilities by creating automated production-ready environments for AI. IBM Analytics Engine separates storage and compute to create a scalable analytics solution alongside Spark's data processing capabilities.

- You can set Spark configuration using properties (to control application behavior), environment variables (to adjust settings on a per-machine basis) or logging properties (to control logging outputs). Spark property configuration follows a precedence order, with the highest being configuration set programmatically, then spark-submit configuration and lastly configuration set in the spark-defaults.conf file. Use Static configuration options for values that don't change from run to run or properties related to the application, such as the application name. Use Dynamic configuration options for values that change or need tuning when deployed, such as master location, executor memory or core settings.
- Use Kubernetes to run containerized applications on a cluster, to manage distributed systems such as Spark with more flexibility and resilience. You can run Kubernetes as a deployment environment, which is useful for trying out changes before deploying to clusters in the cloud. Kubernetes can be hosted on private or hybrid clouds, and set up using existing tools to bootstrap clusters, or using turnkey options from certified providers. While you can use Kubernetes with Spark launched either in client or cluster mode, when using Client mode, executors must be able to connect with the driver and pod cleanup settings are required.

Semana 6

The Apache Spark User Interface

Connect to the UI with the URL:

http://<driver-node>:4040

- The Jobs tab displays the application's jobs, including job status
- the Stages tab reports the state of tasks within a stage.
- the Storage tab shows the size of RDDs or DataFrames that persisted to memory or disk.
- the Environment tab information includes any environment variables and system properties for Spark or the JVM.
- the Executors tab displays a summary that shows memory and disk usage for any executors in use
- If the application runs SQL queries, select the SQL tab and the Description hyperlink to display the query's details.

Monitoring Application Progress

Benefits:

- quickly identify failed jobs and tasks
- fast access to locate inefficient operations

Multiple related jobs:

- from different sources

- one or more DataFrames
- actions applied to the DataFrames

Workflows can include:

- jobs created by the SparkContext in the driver program
- jobs in progress running as tasks in the executors
- completed jobs transferring results back to the driver or writing to disk

How do jobs progress?

1. Spark jobs divide into stages, which connect as a Directed Acyclic Graph, or DAG
2. Tasks for the current stage are scheduled on the cluster
3. When the stage completes all of its tasks, the next dependent stage in the DAG begins.
4. The job progresses through the DAG all stages are completed

→ If any of the tasks within a stage fail, after several attempts, Spark marks the task, stage, and job as failed and stops the application

History server:

```
spark.eventLog.enabled true
spark.eventLog.dir <path-for-log-files>

#command to connect to the spark app UI history server
http://<host-url>:18080

#start history server
./sbin/start-history-server.sh
```

Debugging Apache Spark Application Issues

Common app issues:

- user code
    - driver programm

- configuration
- app dependencies
  - app files
    - source files: python script, java JAR, required data files
  - app libraries
  - dependencies must be available for all nodes of the cluster
- resource allocation
  - CPU and memory resources must be available for all tasks to run
  - any worker with free resources can start processes
  - spark retries until worker is free
- network communication

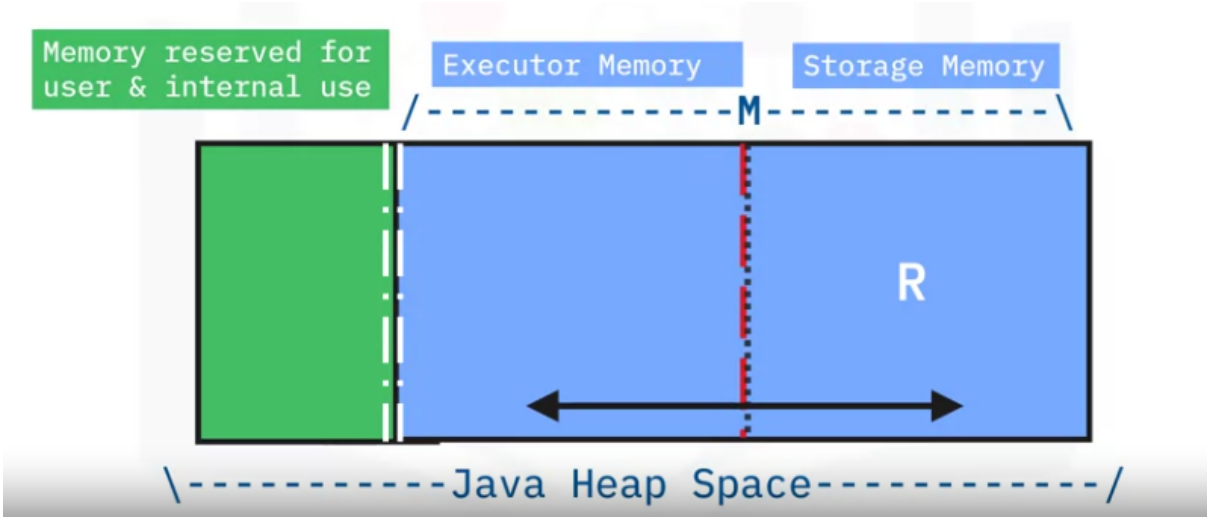
Understanding Memory Resources

Executor memory:

- processing
- caching
- excessive caching leads to issues

Driver memory:

- loads data, broadcasts variables
- handles results, such as collections



Data persistence or cache:

- store intermediate calculations
- persist to memory/disk
- less computation

Understanding Processor Resources

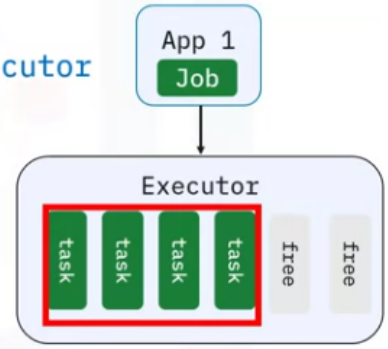
- Spark assigns CPU cores to driver and executor processes
- parallelism is limited by the number of cores available
- executors process tasks in parallel up to the number of cores assigned to the applications
- after processing, CPU cores become available for future tasks
- workers in the cluster contain a limited number of cores
- if no cores are available to an app, the application must wait for currently running tasks to finish
- spark queues tasks and waits for available executors and cores for maximized parallel processing
- parallel processing tasks mainly depend on the number of data partitions and operations
- app settings will override default bahaviour

Core utilization example:

A Spark standalone cluster with one worker node and six cores

- Submit App 1
- The application requests 4 cores per executor

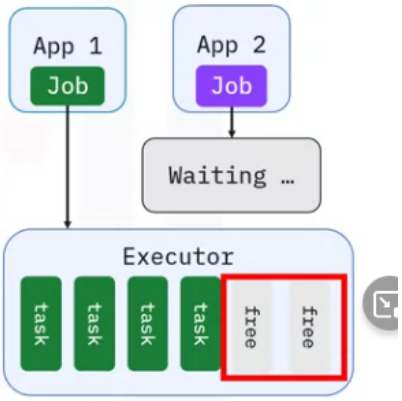
```
$ ./bin/spark-submit \
  --master spark://<spark-master-URL>:7077 \
  --executor-cores 4 \
  examples/src/main/python/pi.py \
  1000
```



- App 1 occupies four cores

- Submit App 2 using the same cluster
- The application requests 4 cores per executor

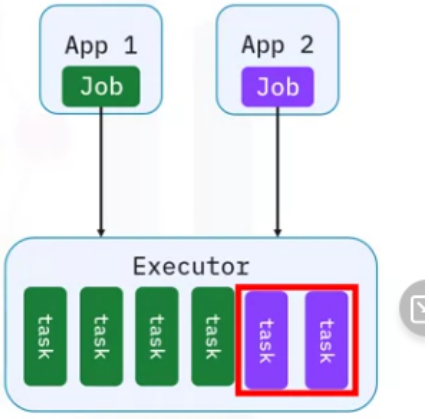
```
$ ./bin/spark-submit \
  --master spark://<spark-master-URL>:7077 \
  --executor-cores 4 \
  examples/src/main/python/pi.py \
  1000
```



- Two cores are available
- App 2 must wait until two additional cores are available

- Start App 2 and request two cores

```
$ ./bin/spark-submit \
  --master spark://<spark-master-URL>:7077 \
  --executor-cores 2 \
  examples/src/main/python/pi.py \
  1000
```



- App 1 and App 2 now run simultaneously

Summary & Highlights

- To connect to the Apache Spark user interface web server, start your application and connect to the application UI using the following URL: http://<driver-node>:4040
- The Spark application UI centralizes critical information, including status information into the **Jobs**, **Stages**, **Storage**, **Environment** and **Executors** tabbed regions. You can quickly identify failures, then drill down to the lowest levels of the application to discover their root causes. If the application runs SQL queries, select the **SQL** tab and the **Description** hyperlink to display the query's details.
- The Spark application workflow includes jobs created by the Spark Context in the driver program, jobs in progress running as tasks in the executors, and completed jobs transferring results back to the driver or writing to disk.
- Common reasons for application failure on a cluster include user code, system and application configurations, missing dependencies, improper resource allocation, and network communications. Application log files, located in the Spark installation directory, will often show the complete details of a failure.
- User code specific errors include syntax, serialization, data validation. Related errors can happen outside the code If a task fails due to an error, Spark can attempt to rerun tasks for a set number of retries. If all attempts to run a task fail, Spark reports an error to the driver and terminates the application. The cause of an application failure can usually be found in the driver event log.
- Spark enables configurable memory for executor and driver processes. Executor memory and Storage memory share a region that can be tuned.
- Setting data persistence by caching data is one technique used to improve application performance.
- The following code example illustrates configuration of executor memory on submit for a Spark Standalone cluster:
  $ ./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master
  spark://<spark-master-URL>:7077 \
  --executor-memory 10G \
  /path/to/examples.jar \1000
- The following code example illustrates setting Spark Standalone worker memory and core parameters:

```
# Start standalone worker with MAX 10Gb memory, 8 cores
$
./sbin/start-worker.sh \
spark://<spark-master-URL> \
–-memory 10G –-cores 8
```

- Spark assigns processor cores to driver and executor processes during application processing. Executors process tasks in parallel according to the number of cores available or as assigned by the application.
- You can apply the argument '--executor-cores 8 \' to set executor cores on submit *per executor*. This example specifies eight cores.
- You can specify the executor cores for a Spark standalone cluster *for the application* using the argument "--total-executor-cores 50' followed by the number of cores for the application. This example specifies 50 cores.
- When starting a worker manually in a Spark standalone cluster, you can specify the number of cores the application uses by using the argument '--cores' followed by the number of cores. Spark's default behavior is to use all available cores.

Course Final Exam

Which of the following Apache Spark benefits helps manage big data processing?

Unified Framework

The three Apache Spark components are data storage, compute interface, and cluster management framework. Which order does the data flow through these components?

the data from a Hadoop file system flows into the compute interface or API, which then flows into different nodes to perform distributed/parallel tasks.

Select the characteristics of datasets.

Strongly-typed; use unified Scala and Java APIs; built on top of DataFrames; are the latest data abstraction added to spark

Which of the following features belong to Tungsten?

- Manages memory explicitly and does not rely on the JVM object model or garbage collector
- Places intermidiate data in CPU registers

How does IBM Spectrum Conductor help avoid downtime when running Spark?

Cluster resources divided dynamically

Spark dependencies require driver and cluster executor processes to be able to access the application project. Java and Scala applications provide this access with what?

uber-JAR

Which command specifies the number of executor cores for a Spark standalone cluster *for the application*?

```
--total-executor-cores
```

Identify common areas where Spark application issues can happen.

User code, configuration, app dependencies, resource allocation, network comm

Select the answer that identifies the main components that describe the dimensions of Big Data.

velocity, volume, variety, veracity

What is Data Scaling?

technique to manage, store and process the overflow of data

What is the current projected yearly growth rate for data?

40%

Which of the following Hadoop core components prepares the RAM and CPU for Hadoop to run data in batch, stream, interactive, and graph processing?

YARN

What happens when Spark performs a shuffle? Select all that apply.

- boundaries between stages are marked
- datasets redistributed across cluster

What is the Spark property configuration that follows a precedence order, with the highest being configuration set programmatically, then spark-submit configuration and lastly configuration set in the spark-defaults.conf file?

Setting how many cores are used → this task configuration could change so dynamic configuration handles it well

What are the required additional considerations when deploying Spark applications on top Kubernetes using client mode? Select all that apply.

- the executors must be able to communicate and connect with the driver programm
- use the driver's pod name to set spark.kubernetes.driver.pod.name

Select the answer that identifies the licensing types available for open-source software.

Public domain, Copyleft, Permissive, Lesser General Public License

How does MapReduce keep track of its tasks?

unique keys

Which of the following characteristics are part of Hive rather than a traditional relational database?

- designed on the methodology of write once, read many
- can handle petabytes of data

Select the option that most closely matches the steps associated with the Spark Application Workflow.

> The application creates a job. Spark devides the job into one or more stages. The first stage starts tasks. The tasks run and as one stages completes, the next stage starts. When tasks and stages complete the next job can begin.