|  |  |  |
| --- | --- | --- |
| **#** | **title** | **Description/Explanation** |
| **1** | **Core Components**  A picture containing table  Description automatically generated | **Spark Core:**   Underlying execution engine for spark platform and all other functionalities have been build on top of it. It contains the functionalities like, fault recovery, memory management, task scheduling and interacting with external storage. Also it contains the APIs for RDD.  **Spark SQL:** Gives support for structured and semi-structured data. It allows querying data via SQL as well as the Apache Hive variant of SQL—called the Hive Query Language (HQL). It supports many sources of data, including Hive tables, Parquet, and JSON.  **Spark Streaming :** Spark Streaming enables processing of live streams of data. It ingests data in mini-batches and performs RDD (Resilient Distributed Datasets) transformations on those mini-batches of data. e.g data streams include logfiles generated in PROD servers or queues of messages contains status update etc.  **Mlib (machine learning):** MLib contains the common machine learning functionalities. It contains multiples types of ML algorithms, including classification, regression, clustering and collaborative filtering.  **GraphX (graph):** GraphX is a library for manipulating graphs (e.g., a social network’s friend graph) and performing graph-parallel computations. GraphX extends the Spark RDD API, allowing us to create a directed graph with arbitrary properties attached to each vertex and edge. |
| **2** | **Interactive Operations on MapReduce**  Diagram  Description automatically generated | While doing iterative operations on MapReduce, intermediate results will be stored on Disk and the same value will be read from the disk later. This contributes overhead due to replication, disk I/O and serialization. eventually it makes system slow. Most of the hadoop applications spend over 90% of time doing HDFS read/write operations. |
| **3** | **Interactive Operations on Spark**  Diagram  Description automatically generated | Spark stores intermediate results in a distributed memory instead of Stable storage (Disk) and make the system faster. It run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.      If the Distributed memory (RAM) is sufficient to store intermediate results (State of the JOB), then it will store those results on the disk. |
| **4** | **RDD vs DataFrame vs Dataset** | **RDD** stands for Resilient Distributed Datasets and is the fundamental data structure of Spark. It contains Read-only/immutable partitions collection of records that spread across many machines in the cluster. It allows a programmer to perform in-memory computations on large clusters in a fault-tolerant manner. So it speed up the task. It can easily and efficiently process data which is structured as well as unstructured but user need to ingest the schema. Also it allows data come from any source, e.g. text file, database via JDBC, etc ( no predefined structure). However, no inbuild optimization engine is available in RDD (e.g. catalyst optimizer and Tungsten execution )  **DataFrame** is a distributed collection of immutable data, organized into named columns. It is conceptually equal to a table in a relational database. It works only on structured and semi-structured data, and it allow the Spark to manage schema. It allows Data processing in different formats (AVRO, CSV, JSON, and storage system HDFS, HIVE tables, MySQL). Optimization takes place using catalyst optimizer in four phases, namely (i) Analysing logical plan, (ii) logical plan optimization (iii) physical plan and (iv) code generation. DataFrame API is very easy to use and faster for exploratory analysis and creating aggregated statistics.  **Datasets** are an extension of DataFrame API which provides type-safe, object-oriented programming interface. It also efficiently processes structured and unstructured data and supports data come from different sources. It auto discover the schema of the files because of using Spark SQL engine. Dataset it is faster to perform aggregation operation on plenty of data sets. |
| **5** | **Internal Spark Execution**  Graphical user interface, diagram  Description automatically generated  Diagram  Description automatically generated | **Sequence of tasks:**  (1) user submits a job using “spark-submit”.  (2) “spark-submit” will in-turn launch the Driver which will execute the main() method of our code.  (3) Further, RDDs will be created for the data source.  (4) Spark creates DAG with logical plans (i.e. mainly code and syntax is correct) and optimization techniques applied .Upon the validations, it submits the DAG to the DAG scheduler.  (5) The DAG scheduler divides he operations into stages of tasks and creates physical plan (eg where the data exist, type of operation) . Each stage is comprised of tasks based on partitions of input data.  (6) The stages are passed on to the Task Scheduler.  (7) Driver contacts the cluster manager(Spark Standalone/Yarn/Mesos) and requests for resources to launch the Executors.  (8) The cluster manager launches the Executors on behalf of the Driver.  (9) Once the Executors are launched, they establish a direct connection with the Driver.  (10) The driver determines the total number of Tasks by checking the Lineage.  (11)Task runs on Executor and each Task upon completion returns the result to the Driver.  (12)Finally, when all Task is completed, the main() method running in the Driver exits, i.e. main() method invokes sparkContext.stop().  (13) Finally, Spark releases all the resources from the Cluster Manager. |
| **6** | **Transformations/Actions** | Transformations: which create a new dataset from an existing one.  Actions: which return a value to the driver program after running a computation on the dataset. |
| **7** | **Narrow Transformation**  Chart, box and whisker chart  Description automatically generated  **Wide Transformation**    Diagram  Description automatically generated | all the elements that are required to compute the records in single partition live in the single partition of parent RDD. A limited subset of partition is used to calculate the result. *Narrow transformations* are the result of *map(), filter().*              all the elements that are required to compute the records in the single partition may live in many partitions of parent RDD. The partition may live in many partitions of parent RDD. *Wide transformations* are the result of *groupbyKey()* and *reducebyKey()*. |
| **8** | **Spark vs Hive** | |  |  |  | | --- | --- | --- | | **#** | **Hive** | **Spark** | | **1** | Hive is a data warehousing tool built on top of hadoop to enable querying and analytical purpose. | Apache Spark is a data processing framework that can quickly perform processing tasks on very large data sets, and can also distribute data processing tasks across multiple computers. | | **2** | Slower than Spark as it relay on MapReduce which uses too many Disk IO processing. | Faster in execution times (as it uses inmemory, RDD and DAG scheduler) compare to Hadoop. | | **3** | User can easily access the terminal and query the data. | The purpose of Spark is quickly processing tasks. No terminal access to users. | | **4** | Used to perform data-warehousing (data come from many sources) operations and its meant for batch jobs. | Suited for big data applications performing huge data processing and expect results faster than mapreduce. | |
| **9** | **Hive vs Impala** | |  |  |  | | --- | --- | --- | | **#** | **Hive** | **Impala** | | **1** | Implemented in Java and reply on Mapreduce. So performance is poor. | Implemented in C++ and uses in memory based computation. So its fast. | | **2** | Hive performs well for complex queries. | Impala struggles as query complexity increases but Impala perform well with less complex queries. | | **3** | Hive is fault-tolerant. | If any nodes goes down, it has to restart. No fault tolerance. | | **4** | Meant for batch processing. | Gives instant result. | |
| **10** | **Scalable Spark Application** | **Hardware side - adding new nodes/cores**  **Parameters - executors/cores/threads**  **Cluster/client**  **<<few more >>** |
| **11** | **Two modes: Client vs Cluster mode**  <https://gankrin.org/difference-between-spark-cluster-client-deploy-modes/>  Major difference between yarn-cluster mode and yarn-client mode is the location where driver program is run.          <https://blog.knoldus.com/cluster-vs-client-execution-modes-for-a-spark-application/> | A spark application gets executed within the cluster in two different modes – one is cluster mode and the second is client mode:  **In client mode**, the node where the spark-submit is invoked and it will act as the Spark driver, and where the SparkContext will live for the lifetime of the app. But this node WILL NOT execute the DAG as this it is designated JUST as a driver for the spark cluster. However all the other nodes will act as executors for running the job. We can track the execution of the jobs through the Web UI. In the client mode, the client who is submitting the spark application will start the driver and it will maintain the spark context. So, till the particular job execution gets over, the management of the task will be done by the driver. Also, the client should be in touch with the cluster. The client will have to be online until that particular job gets completed.    **In cluster deploy mode** , all the slave or worker-nodes act as an Executor. But one of them will act as Spark Driver too. Means which is where the SparkContext will live for the lifetime of the app.  The node which submit the job also act as executor.  In the cluster mode, the Spark driver or spark application master will get started in any of the worker machines. So, the client who is submitting the application can submit the application and the client can go away after initiating the application or can continue with some other work. So, it works with the concept of Fire and Forgets.    **When to use Client/Cluster mode?**  If we submit an application from a machine that is far from the worker machines, for instance, submitting locally from our laptop, then it is common to use cluster mode to minimize network latency between the drivers and the executors. In any case, if the job is going to run for a long period time and we don’t want to wait for the result then we can submit the job using cluster mode so once the job submitted client doesn’t need to be online.  Client mode :we can monitor the activities closely. |
| **12** | **Spark-submit**  ./bin/spark-submit --class org.apache.spark.examples.SparkPi --master spark://<<hostname/ipaddress>>:portnumber --deploy-mode cluster ./examples/jars/spark-examples\_2.11-2.3.1.jar 5(number of partitions) | **Parameters :**  class -> main clas of the job  deploy-mode > cluster/client  driver-memory - memory allocated to the driver  num-executors - number of executors to be created  executor-memory - amount of memory to be allocaed for executor.  executor-core - number of threadds used by each executor. |
| **13** | **YARN**  **(**<https://www.edureka.co/blog/hadoop-yarn-tutorial/>)  Diagram  Description automatically generated | Yarn (Yet Another Resource Negotiator) is a generic resource-management and job scheduling framework for distributing/balancing workloads in a distributed environment. Although it is part of the Hadoop ecosystem, YARN can support other compute-frameworks (such as Tez, and Spark) in addition to MapReduce. Yarn keeps track of available resources (memory, CPU, storage) across the cluster (where Hadoop is running). Then each application (e.g., Hadoop) asks the resource manager what resources are available and other resource related queries. YARN enabled the users to perform operations as per requirement by using a variety of tools like Spark for real-time processing, Hive for SQL, HBase for NoSQL and others. YARN performs all your processing activities by allocating resources and scheduling tasks.  YARN Components:   * 1. **Resource Manager:**Runs on a master daemon and manages the resource allocation in the cluster.   2. **Node Manager:**They run on the slave daemons and are responsible for the execution of a task on every single Data Node.   3. **Application Master:**Manages the user job lifecycle and resource needs of individual applications. It works along with the Node Manager and monitors the execution of tasks.   4. **Container:** Package of resources including RAM, CPU, Network, HDD etc on a single node |
| **14** | **Using Yarn in a Hadoop cluster**    **(**<https://www.bmc.com/blogs/hadoop-apache-yarn/>) | * + 1. Spin up at least 2 virtual machines. Three would be better.     2. Create the Hadoop user and do the following Hadoop installation steps and run the Hadoop commands as that user.     3. Put hostnames of all 3 machines and their IPs in /etc/hosts     4. Create a folder on the master to store Hadoop data. The name you put here must correspond with step 7.     5. Install Hadoop on the machine you designate as the master.     6. Install Java on all 3 VMs.     7. Look at Hadoop’s instructions for configuring core-site.xml and hdfs.xml. The instructions vary slightly by Hadoop version.     8. Edit $HADOOP\_HOME/etc/hadoop/slaves and add the hostnames of the slaves.     9. Add JAVA\_HOME to hadoop-env.sh.     10. Do not install Hadoop on the slaves. Rather copy the entire Hadoop installation (i.e., the $HADOOP\_HOME directory) to the slaves.     11. Set the .bashrc config file or the Hadoop user as shown below.     12. Run start\_dfs.sh to start Hadoop and start\_yarn.sh. You do not need to restart Hadoop as you work through installation problems with Hadoop and restart that. (It will probably take a few times to get it correct.)     13. Run jps on the slaves and make sure the dataNode process is running.     14. Run jps on the master and make sure the nameNode process is running.     15. Make sure you can open this URL on the master [http://localhost:50070/dfshealth.html#tab-overview](https://_localhost_50070/dfshealth.html#tab-overview)     16. Format the Hadoop file system using: hadoop namenode -format.     17. Install Apache Pig and make sure you can run the example shown below. |
|  |  |  |

* 1. **Spark**
  2. **Hive**

|  |  |  |
| --- | --- | --- |
| **#** | **Component** | **Description** |
| **1** | **Architecture** | Diagram  Description automatically generated |
| **2** | **Request flow** | Diagram  Description automatically generated |
| **3** | Managed/External tables |  |
| **4** | Hive Partitions |  |

* 1. **Impala**
  2. **Ozzie**

Apache Oozie is a workflow scheduler for Hadoop. It is a system which runs the workflow of dependent jobs. Here, users are permitted to create Directed

Acyclic Graphs of workflows, which can be run in parallel and sequentially in Hadoop.

* 1. **Kafka**
  + Definition
  + Producer/Consumer/Kafka Brokers/Kafka Cluster/Topic/Partitions/Partition Offset/Leader/Followers
  + Consumer group
  + Replication factor
  + Zoo keeper
  + Pub-Sub messaging workflow
  + Messaging and consumer group
  + Spark Integration (is it required? ) i.e. Kafka -> Spark Streaming -> Spark Engine -->DB & HDFS & Dashboard
  + Pub-Sub vs P2P
  + Deciding number of partitions per topic . Min no of service to run Kafka cluster

Diagram

Description automatically generated

Diagram

Description automatically generated

* + **Java**

Diagram

Description automatically generated

* + **Other Questions**
  + How to improve the performance of MapReduce (Ans : use spark )
  + **Big data file formats**

When processing Big data, the cost required to store such data is more (Hadoop stores data redundantly to achieve fault tolerance). Choosing an appropriate file format

can have some significant benefits.

|  |  |  |
| --- | --- | --- |
| # | Name | Description |
| 1 | **AVRO** | Avro is a row-based storage format for Hadoop which is widely used as a serialization platform. Avro stores the schema in JSON format making it easy to read and interpret by any program. |
| 2 | **PARQUET** | Parquet, an open-source file format for Hadoop stores nested data structures in a flat columnar format.  Compared to a traditional approach where data is stored in a row-oriented approach, parquet is more efficient in terms of storage and performance. |
| 3 | **ORC** | The Optimized Row Columnar ([ORC](https://orc.apache.org/)) file format provides a highly efficient way to store data. It was designed to overcome the limitations of other file formats. It ideally stores data compact and enables skipping over irrelevant parts without the need for large, complex, or manually maintained indices. |

* + **Compression techniques**

|  |  |  |
| --- | --- | --- |
| # | Compression Technique | Description |
| 1 | gzip | provides lossless compression that is not splittable. It is often used for HTTP compression. |
| 2 | Snappy | provides lossless compression that is not splittable. It is integrated into Hadoop Common and often used for database compression. |
| 3 | LZ4 | provides lossless compression that is not splittable unless combined with the 4MC library. It is used for general-purpose analysis. |
| 4 | Zstd | provides lossless compression that is splittable. It is not data type-specific and is designed for real-time compression. |

* + Data warehouse vs Data Mart

|  |  |  |
| --- | --- | --- |
| # | component | Description |
| 1 | Diagram  Description automatically generated | data warehouses are built to serve as the central store of data for the entire business.  data mart fulfills the request of a specific division or business function. |

* + Direct data from data lake will be viewed by Hue (hadoop user experience)
  + **File Transfer Mechanisms**

Connect:Direct is a managed file transfer product that transfers files between, and within enterprises.

SFTP (SSH File Transfer Protocol) : File transfer protocol that's perfect for businesses who require privacy/security capabilities. SFTP runs on SSH

* + Edge, Master and Data Nodes

An **edge node** is a computer that acts as an end user portal for communication with other [nodes](https://searchnetworking.techtarget.com/definition/node) in [cluster computing](https://whatis.techtarget.com/definition/distributed-computing). Edge nodes are also sometimes called [gateway](https://internetofthingsagenda.techtarget.com/definition/gateway) nodes or edge communication nodes. **Master nodes** control which nodes perform which tasks and what processes run on what nodes. The majority of work is assigned to **worker nodes**. Worker node store most of the data and perform most of the calculations  Edge nodes facilitate communications from end users to master and worker nodes.

* + Load data into datalake

Prepare a simple file transfer program or framework that accepts (source, target paths, file types ) and run as batch.

* + Loading Json file in hive

-> store the json file in hdfs

-> create external table and "ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.JsonSerDe'

-> query the content

* + ODBC vs JDBC

ODBC is an SQL-based Application Programming Interface (API) created by Microsoft that is used by Windows software applications to access databases via SQL. JDBC is an SQL-based API created by Sun Microsystems to enable Java applications to use SQL for database access.

* + ETL background (NiFi) and Stored procedure knowledge in sybase
  + Batch layer work
  + Implemented log collector framework
  + Intro about myself (including research)
  + Explain EqEdting/iRFQ and why switched to three tier ( performance + separating UI and busines logic + maintenance + modern technologies (i.e HTML + Java microservices)
  + Already on Java and Bigdata
  + TODO : correct the design
  + Two tier to three tier