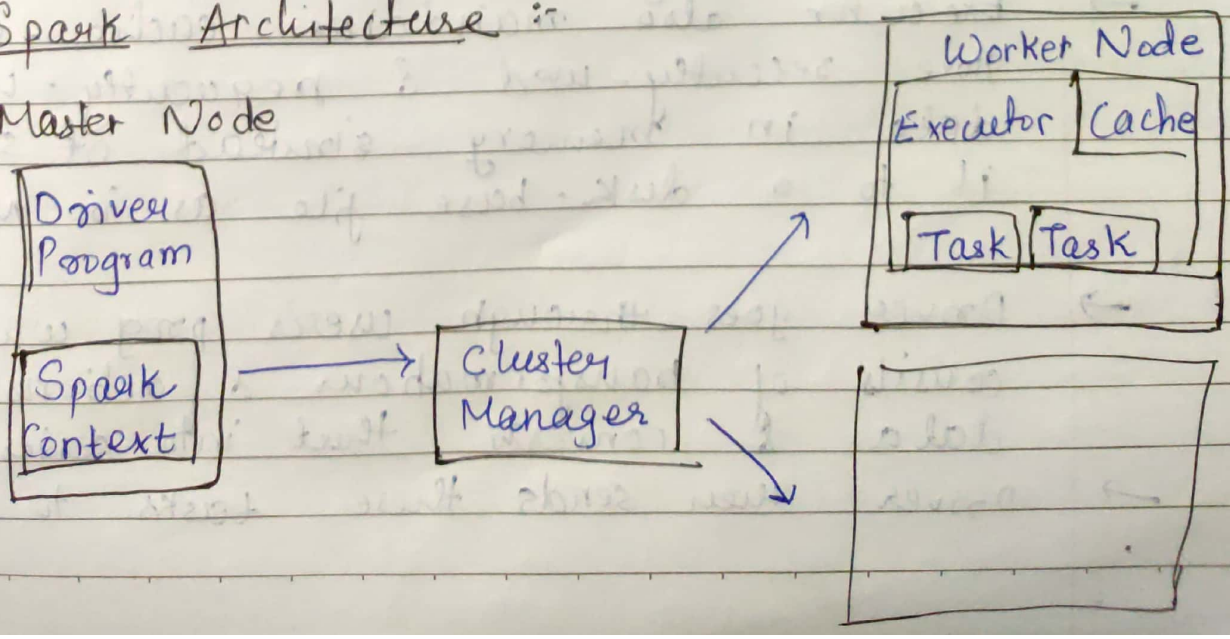
Spark

Many app had to run MR over multiple passes to process their data.
All intermediate data had to be stored back in file sys (HDFS), which tended to be slow since stored data was not just written to disks but also replicated.
Next MR phase could not start until the previous MR job completed fully.
MR was also restricted in what it could read its data from HDFS or some storage only.

Spark is an open source data processing engine to store & process data in real-time across various clusters of computers using simple programming constructs.
It stores its intermediate results in memory, providing for dramatically higher performance.
It has support for more map & reduce apes. so MR jobs can be implemented along with the SQL, graph proce, machine lear

Spark Architecture :-



→ Apache spark uses a master - slave archit that consits of driver, that runs on a master node & multiple executors which run across the worker nodes in the cluster

→ The spark code behaves as a driver prog and creates a SparkContext (sc), which is a gateway to all the Spark functionalities

→ The SC connects to Spark cluster manager which is responsible for allocating worker nodes, launching executors on them & keeping track of their status.

→ Each worker node runs one or more executors

→ An executor is a process that runs an instance of JVM.

→ Executor runs task on behalf of a specific Sp & keeps related data in memory

→ Executor remains running for duration of app which provides adv of performance over MR since new task can be started very quickly.

→ Executor also maintains cache, which store recently-used & frequently-used data in memory instead of storing it to a disk-base file as in MR.

→ Driver goes through user's prog which consits of transformations & actions on data & converts that into series of tasks

→ Driver then sends these tasks to executors

→ A task is app code that runs in executor on a JVM & can be written in Scala, Java, Python, etc.
→ It is transmitted as a jar file to executor which runs it.

* Resilient Distributed Datasets (RDD) :-

→ Data in Spark is a collection of RDDs.
→ Consider individual RDD as giant table in db or structured file.
→ Data is organized in RDD
→ RDD will be partitioned (sharded) across many computers so each task will work on only a part of dataset.

Created in 3 ways :

1) as a file in HDFS.
2) can be streaming sources
3) can be output of transformation fn.

Properties :

→ immutable : contents cannot be changed.
→ typed : Key-value pair (structured somehow)
→ partitioned :
→ Fault tolerance :
→ Lazy Evaluation
→ Persistence
→ In memory computation
→ Parallel

Operations :

1) Transformations :
→ read RDD & return new RDD.
Eg map, filter, groupByKey, etc.
→ evaluated lazily : computed only when some task wants their data.

2) Actions :
→ are opes that evaluates & return new value
→ when an action is requested on an RDD object, necessary transformations are computed and result is returned.
Eg reduce, write to file, grab samples, etc
count, first, etc.

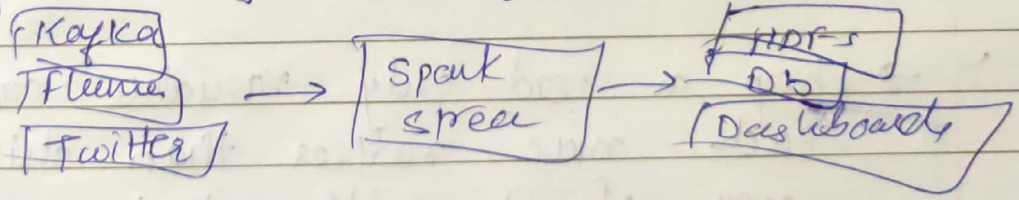
* Features & Adv of Spark :-

→ faster than MR jobs
→ High performance : 1) storing interme results in memory
2) allowing multiple transformations & actions read from the same dataset.
→ supports more ope than MR
→ no need to adapt a pb into series of MR opes, can define an arbitrary set of i/p, transfor, actions to produce results.
→ fault tolerant
→ dataset partitions can be regenerated if needed
→ tracks what transf was used to create the needed dataset & i/p transf req. With this, it can work backwards & recreate any missing RDD.

* Spark Core components :
↳ underlying general execution engine for spark platform that all other function are built upon.
↳ provides in-memory computing & referencing datasets in external storage sys.

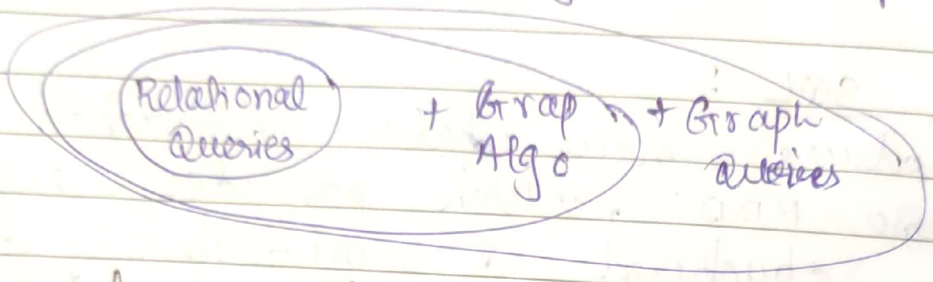
1) Spark SQL :
→ introduces new data abstraction called schema RDD, which provides support for structured & semi-struct data.
→ Blurs the line blw RDD & relational tables
→ intermix SQL command to query external data.
→ allow SQL extensions based on MLlib

2) Spark Streaming :
→ leverages Spark core's fast scheduling capability to perform streaming analytics.
→ It ingests data in mini batches & perform RDD trans on those mini bs
→ extends core API to allow high throughput, fault tolerant proces.



3) MLlib :
→ distributed ML framework becoz of distributed memory based spark archi
→ 9x faster than Hadoop disk based version of Mahi
→ goal → practical M. Learning scalable & easy.
→ includes : classifi, regre, cluster, etc.

4) GraphX :
→ distributed graph processing framework
→ provides API for expressing graph computation
→ " optimized runtime for this abstraction
→ recreate SQL queries in graphs to have better grasp of graph concepts.



* Shared variables :-

1) Broadcast variables :
→ used to efficiently distribute large values.

2) Accumulators :
→ used to aggregate the info of particular collection
→ can be "added" only through associative ops.
→ used to implement counters & sums
→ driver program can read accumulator's value not the task

→ Keep a read only variable cached on each m/c rather than shipping a copy of it with task.
→ eff broadc algo → to reduce comm cost.