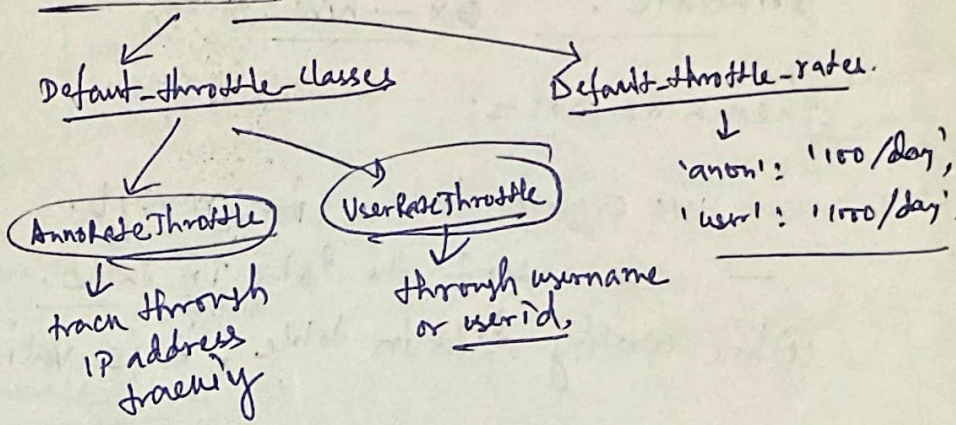
Throttling

similar to permissions.

to restrict no of requests to your API.

two policy



Hive Indexing vs Materialize view :

> Hive version 3.0

↳ there is no indexing in Hive.

to accelerate query processing. in Data Warehouse, we should precompute relevent summaries or materialized view.

- Query rewriting → it automatically rewrites incoming queries using materialize views & hence, accelerate query execution.

By default value → true.

hive.materializedview.rewriting in property

Maintanence of Materialize view:

we can Rebuild the view when source table data changes or new data inserted.

By default Hive supports incremental rebuild ⊙ means only new data will be refreshed but to refresh updated or delete operation → go with full rebuild.

** what do you mean by schema on read & schema on write?

schema on read :

we have the data in the form of files & we create table structure on top of it to see the tabular structure. Ex — hive.

schema on write:

- we create table in mysql.
- we insert the data in table.

While writing data in table, it is validating the things.

** create schema in hive —

```
Create database mydatabase
Comment "New Database"
Location '/path to db'
WITH DBPROPERTIES (property_name2 value);
```

** Utilizing hive partitioning with some settings. ✓

set hive.mapred.mode = strict.

it restrict 3 varieties of queries —

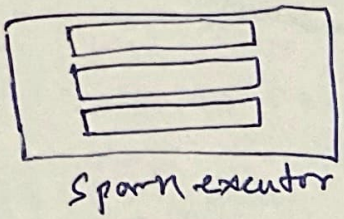
1. queries on partitioned table not permitted unless you include a partition filter in WHERE clause.
2. NO ORDER BY without LIMIT clause..
3. prevent cartesian product.

→ all are permitted if you set it to "nonstrict".

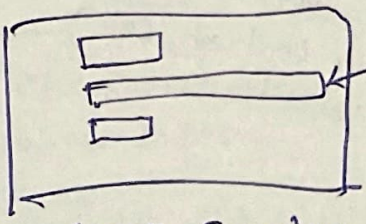
How to handle data skewness in spark —

When data is unevenly distributed among partitions in the cluster.

→ it downgrades performance of query, specially joins.



(evenly distributed)



(unevenly distributed)

How to handle?

1. salt the skewed column.
2. Apply skew-hint.
3. use Broadcast join.
4. Enable Adaptive query Execution (Spark 3.0)

↓ it provides three features:

→ Dynamically coalescing shuffle partitions.

→ Dynamically switching join strategies.

→ Dynamically optimizing skew joins.

to enable —
spark.sql.adaptive.enabled = true.

modes in spark:

three main modes of operation:

1. **Local mode:** single machine using all available core. default mode & suitable for development & testing.
2. **standalone:** when your program uses spark's resource manager. here also spark runs on cluster of machines. (for small scale production)
3. **cluster mode:** spark runs on a cluster manager such as YARN, Mesos, etc. (for large scale production).

SparkContext vs SparkSession:

SparkContext

↓

Since spark 1.x

↓

you need to create separate spark context for every thing like (Hive Context, sql Context).

SparkSession:

↓

Since spark 2.0

↓

provides unified entry point of a spark application.

→ all these encapsulated in SparkSession.

Sqoop performance tuning techniques:

HDFS ⟷ RDBMS ✓

(1) --num-mappers or -m (no of map tasks to run in parallel). ✓

by default 4 mappers works.

no of map tasks < max no of connections the database supports.

(2) --split-by : clm of the table used to split work units across the mappers.

Sqoop use this clm to split the data across the executor.

if no primary key present, use --split-by argument.

(3) --boundary-query : use for creating splits.

(4) --fetch-size : No of entries to read from database at once.

by default 1000 records at a time.

(5) --direct : fast import doesn't run any map or reduce.

only supported in mysql & postgresql.

How to handle sqoop data loss while transfering huge data.

what to do? check why failed → get app url & see errors.

Create a staging table ——

sqoop export --

--staging-table tablename \

export will happen to staging table so, staging table will be impacted if job failed.

--clear-staging-table ⇒ clear the staging table data before starting the sqoop job.

Q multiple small files? How to deal with it in spark?

Sol (1) use Coalesce & Repartition:

optimal no of files = Total file size / Configured block size.

(2) spark.sql.shuffle.partitions
↓
Set optimal value (By default 200).

Q Why output of the map writes to local Disk, not to HDFS?

if written to HDFS → will replicate blocks of data.
↳ overhead for namenode to hold metadata.
↳ if job failed in between, will create huge intermediate output residing in HDFS & cleaning up would be difficult.

Q What is speculative execution in spark.

speculative task → is the task runs slower than the rest of the task in a job.
it runs slower than median successfully completed tasks in the task sheet.
↳ it will not stop the slow running task but it launch new task in parallel.

spark.speculation → false (default)
spark.speculation.interval → 100 ms → time interval to use before checking for speculative tasks.
spark.speculation.multiplier → 1.5 → How many times slower than the median.
spark.speculation.quantile → .75 → percentage of the tasks that has not finished yet.

Q Where you can stored data of hive tables?
~~in hive metastore~~ HDFS, S3 or any other Hadoop compatible file system.
default → /user/hive/warehouse.

Q1 Why hdfs is not used by hive metastore for storage?

RDBMS is being used, schema has to be updated frequently & quick access to this schema is required.

Q2 Difference between external & managed tables in hive?

- external → after drop, file will be intact.
- managed → after drop, files will also be deleted.

Q3 order by:

In strict mode, order by should provide "limit" clause.

hive.mapred.mode = strict.

SORT BY:

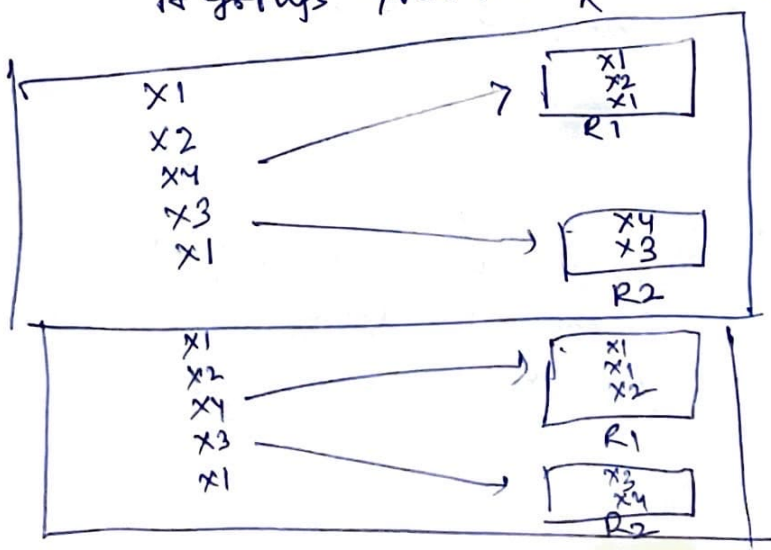
Sort the data per reducer.

order by vs sort By

- order by → provides guarantee ordering of rows in output.
- sort By → only guarantee ordering in single reducer.
  if more than one reducer then Sort By gives partially ordered final result.

Distribute By:

distribute rows among reducer in a query.
it groups your data in a same reducer.



Distribute By (doesn't sort in same reducer)

cluster By (also does sorting)

Distribute By + Sort By

** What is dynamic partitioning in Hive?

allows for automatic creation of partition.
every time it is being taken care at runtime & no need to be aware of the data well in advance.

Steps:

1. create normal table without any partition & have the data loaded in it.
2. create a partitioned table with partition on a column name.
3. transfer the data from normal table to partitioned table.

** How Bucketing works in hive?

in Hive 3.2 → murmurhash algorithm.
you can write your custom hash fun also.

set hive.enforce.bucketing=true;

*** Spark vs MR

100x times than MR	Faster than traditional
Scala	Java
Batch/real time processing	Batch processing
Caches the data in-memory & enhances the system performance	Don't support caching.

** RDD

Resilient — track data lineage information to recover lost data, automatically on failure.
Distributed — RDD resides on multiple nodes. it is distributed across different nodes of a cluster.
Datasets

** How spark achieve fault tolerance?

using RDD's lineage & checkpointing.
↓
you should cache your RDD before checkpointing.

** transformation & actions

two types of RDD operations.

transformation — it is a function that produces new RDD from existing RDD. (map(), flatMap(), filter(),)

action — it actually works on dataset.
(count(), collect(), take(), reduce()).

Narrow vs wide transformation

- No shuffling is involved (map, filter, flatmap)
- shuffling is involved reduceByKey / groupByKey

RDD vs lineage?

each RDD will have a lineage.

RDD lineage is logical execution plan & expanded every time you apply a transformation on any RDD.

DAG → combination of vertices & Edges.

- vertices → represents RDDs
- Edges → operations to be applied on RDD.

upon calling an action → DAG submitted to DAG scheduler & further splits the graph into the stages of the task.

Spark partition:

to reduce data shuffle around the network in preparation for subsequent RDD processing. When the data is key-value oriented, partitioning is essential. Because the range of keys or comparable keys is in the same partition.

adaptive query execution (enabled in spark 3.2.1)

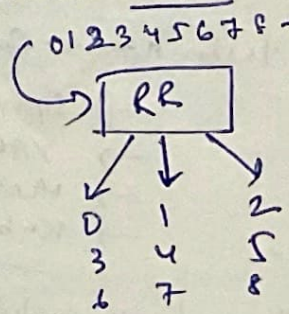
- → it reduces the no of partitions used in shuffle operation in runtime.
- → it also determined join ~~partition~~ strategies in real-time & optimizes skewed join.

Still we need to do some static configuration, —

we can also dynamically repartition over dataset, —

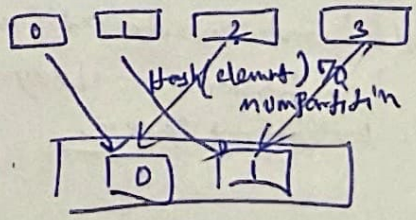
① Round Robin partitioning:

df.repartition(10)



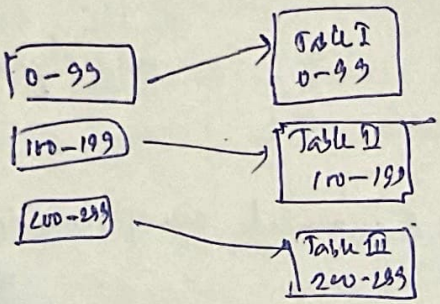
② Hash partitioning:

df.repartition(10, 'class_id')



③ Range partitioning:

df.repartitionByRange(10, 'grade')



How to determine optimal no of partition?

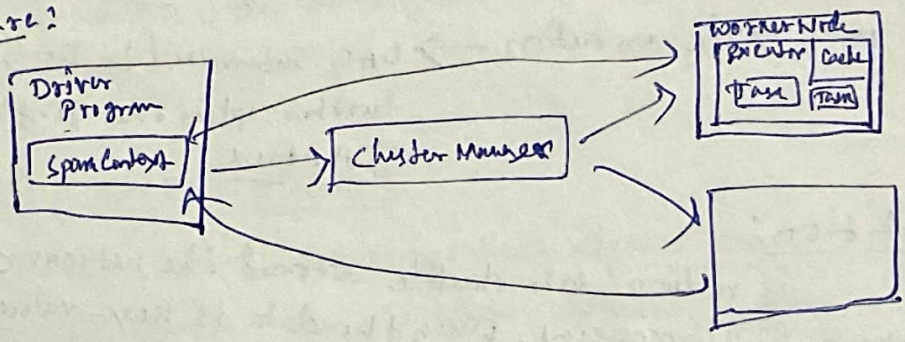
(1) Available resources in your cluster — ex no of available cores.

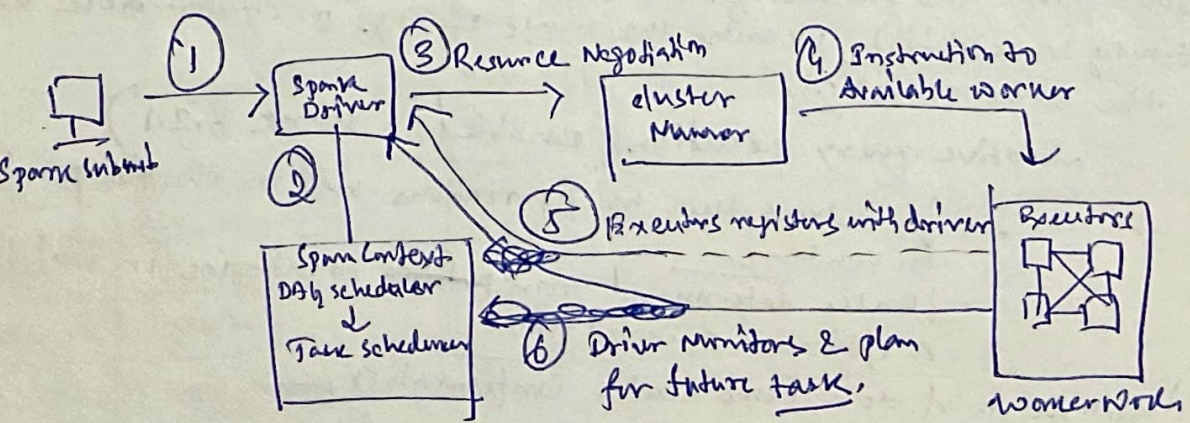
(2) How your data looks (sizing, cardinality)

Ans What is Spark Core?

distributed compute engine with all functionalities attached on its top. (Ex- mllib, sparkSQL, Graphx, spark streaming).

Ans Spark architecture:





cluster manager can be any of the following —

- → Spark Standalone mode
- → YARN
- → Mesos
- → Kubernetes.

Ans What is Spark Lazy evaluation:

Spark will not execute until an ACTION is called.

transformation → Lazy

Action → Eager.

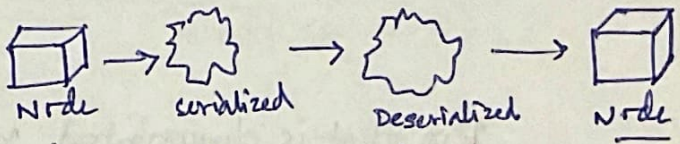
** Diff btw cache & persist:

both provide an optimization mechanism to store the intermediate computation of an RDD, DF or DS. so that it can be reused.

Cache()
by default saves it in memory (MEMORY_ONLY) for RDD.
(MEMORY_AND_DISK) for DF & DS.

persist()
stores in user defined storage level.

** What is serialization in spark,



In distributed system, data transfer over network is the common task. If this is not handled efficiently, you may endup facing problems like, high memory usage, network bottlenecks, & performance issues.

Serialization converts objects into stream of byte in an optimal way to transfer it over nodes of network.

By default, Java Serialization, but it leads to large serialization formats. we can fine tune the performance by extending java.io.Externalizable use Kryo serialization, takes less memory.

** map vs Flatmap:

map
- one-to-one transformation function
- Can return only one item

flatmap
- one-to-many transformation function.
- output is flattened.

** what are the various levels of persistence in spark.

MEMORY_ONLY
MEMORY_ONLY_SER
MEMORY_AND_DISK
MEMORY_AND_DISK_SER
DISK_ONLY
OFF_HEAP
MEMORY_ONLY_2

** accumulator in spark!

like a global variable in spark application.
By default spark provides ~~three~~ accumulator methods for long, double & collection types.

** What are broadcast variable?

→ Broadcast variable available to all executors executing the spark application.

→ size of the data you are broadcasting should be in MBs not GBs.

Ex- replacing country short form with full countryname.

what is streaming checkpoint/checkpoint in spark?

- to achieve fault tolerance in Apache Spark.

→ It truncate RDD lineage graph & save the application state timely to reliable storage (HDFS).

two types of checkpointing in spark —

① Reliable checkpointing:
Saved to HDFS.

② Local checkpointing:
Saved to local disk in executor's memory.

RDD that is checkpointed will be computed twice, first it will compute the job, & launches another job to flush checkpoint.
So, it is recommended to do a rdd.cache() before rdd.checkpoint(), In this case 2nd job will not recompute the rdd, instead it will just read the cache.

Spark Context vs Spark session

before spark 2.0.0,
In order to use APIs of SQL, Hive, streaming, separate Context need to be created.

after spark 2.0.0, spark session provides a single point of entry to interact with underlying spark functionality.
SQL, Hive, streaming all includes in spark session.

Memory allocation in spark

Example

Nodes → 6

each Nodes → 16 Cores, 64 GB RAM.

1 core & 1 GB needed for OS & Hadoop Daemons.

So each node has 15 cores & 63 GB RAM.

No of Core ⇒ Concurrent tasks an executor can run.

⇒ optimal value is 5.

No of executor:

No of executors (In each node) = No of cores in each Node(15) / No of cores (5)

= 3. ✓

total 6 Nodes, so 6 × 3 = 18 executors.

- 1 executors for application master.

So, 17 executors.

--num-executors = 17.

Memory for each executor:

3 executor per node & available RAM for each node = 63 GB.

So, memory for each executor = 63/3 = 21 GB.

you need small overhead memory when you request for executor memory.

that is,

max (384 MB, 10% of 21 GB)

⇒ 2.1 GB.

So, executor memory to request = 21 - 2.1 GB

= 18.9 GB

So, if you request 18.9 GB, then total (18.9 + 2.1) GB will be submitted to YARN. ✓

Stages in Spark

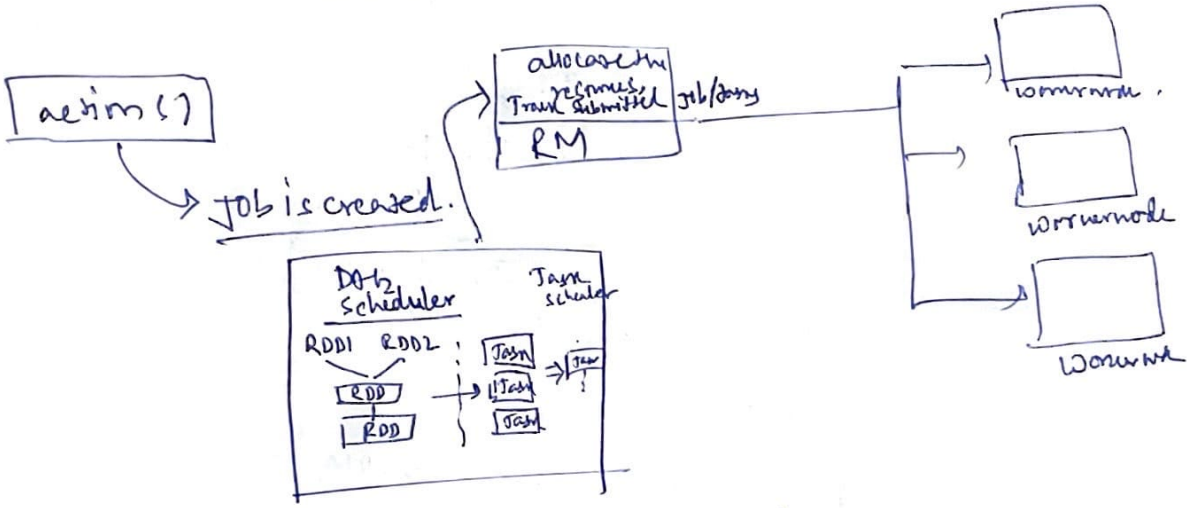
Stage is the -

1. physical unit of execution
2. step in the physical execution plan.
3. stage is the collection of tasks - one task per partition.
4. Job is a collection of stages.

Two types of stages in spark -

→ ShuffleMapStage

→ Result stage.



⇒ We can associate stage with many other dependent parent stages.

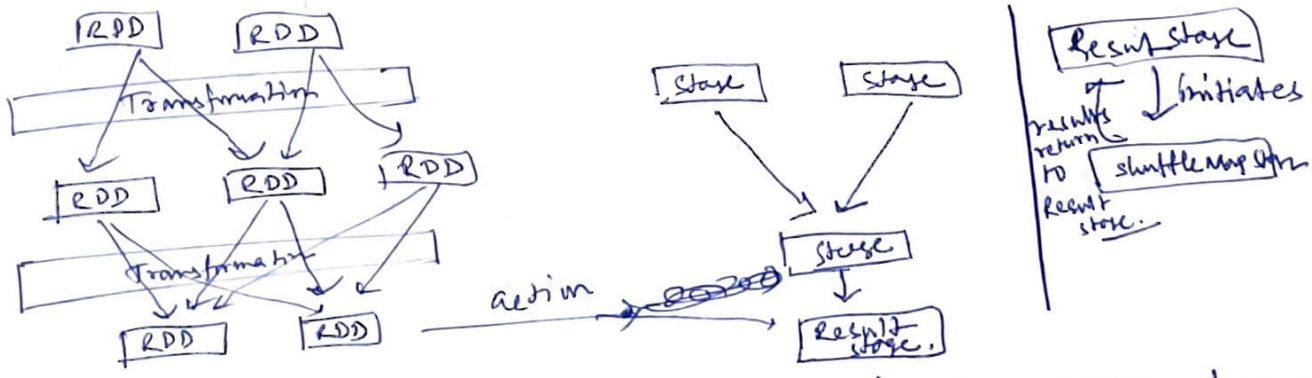
⇒ Data is passed between the stages through Exchange/shuffle.

⇒ stage submission triggers execution of a series of dependent parent stages.

ShuffleMapStage:

Intermediate stage, produces data for another stages.

also, writes map output files for a shuffle.



ResultStage:

→ A stage that executes a spark action ~~stage~~ in a user program by running a function.

→ also helps for computation of the result of an action.

** Create a table structure from another table:

```
Create table new-table as
(select * from old-table where 1=2)
```

** What is a Constraint & what are the seven constraint?

↳ is a set of rule which controls the data goes into the table.

Ex- primary key
Foreign key
check,

```
Create table students
(student-id Number not null,
student-name varchar(20),
MARKS NUMBER check(marks>35))
```

Default

used to set a default value for a Column.

```
create table persons(
    ID int not null,
    LastName varchar(255) Not null,
    Age int,
    City varchar(255) DEFAULT 'Kolkata'
    );
```

Nullability:

NULL or NOTNULL.

Unique key:

Same as primary key but Unique can accept NULL as its value.

Surrogate key:

also called synthetic primary key. It is generated when a new record is inserted into a table automatically by a database.

Ex- System date & timestamp
Random alphanumeric string.

OLTP	vs	OLAP
online transactional processing		online analytical processing.
Highly normalized.		highly denormalize.
Current data.		historical data.
fast for delta operation		fast for read operation.
Delta operations.		only select operation.
	Terms Used:	dimension table, fact table.

** Explain about cursors ?

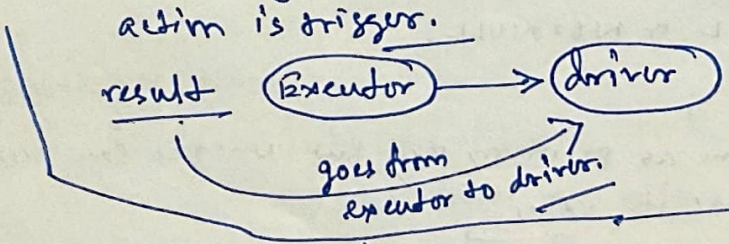
temporary DB object which are used to loop through a table on row-by-row basis. five types of cursors :

** what is action & transformation in spark ?

transformation → produces new RDD from existing RDD.

- Narrow [Map, flatMap, MapPartition, Filter, Sample, Union]
- Wide [Intersection, Distinct, ReduceByKey, GroupByKey, Join, Cartesian, Repartition, Coalesce].

action → computing transformation does occure only when action is trigger.



count()
collect()
take(n)
top()
countByValue()
reduce()
aggregate()
foreach()

** what is serialization & deserialization ?

In distributed system, data transfer over the network is the most common task.
if not handled properly → you may end up facing numerous problems, like high memory usage, network bottlenecks,

Serialization helps to transfer data effectively over network.
→ converts object into a stream of bytes & vice versa.
to optimally transfer it over nodes of network.

Java serialization (default) / Kryo serialization (recommended by Spark).

** checkpointing: It is the process of saving intermediate RDDs to a reliable storage system. By peridically checkpointing RDDs, spark can recover lost data partitions & continue processing from the checkpointed state instead of recomputing from scratch.

HDFS, S3 etc.

~~spark~~

** repartition & coalesce

```
val x = range(1, 13)
df2 x.toDF("number")
df.rdd.partitions.size => 4
```

How data is divided into partition —

Partition 00000 : 1, 2, 3
Partition 00001 : 4, 5, 6
Partition 00002 : 7, 8, 9
Partition 00003 : 10, 11, 12.

Let's do a shuffle with repartition method & get this data on two partitions. df.repartition(2)

Partition A : 1, 3, 4, 6, 7, 9, 10, 12
Partition B : 2, 5, 8, 11.

** join two data frame:

df.join(df1, on=['session-id'], how='inner')

how = 'left'
how = 'leftouter'
how = 'right'
how = 'rightouter'
how = 'fullouter'
how = 'leftsemi'
how = 'leftanti'

multiple conditions.

df.join(df2, [df1.val < df2.val, df1.val2 < df2.val2], how='inner')

** remove duplicates from dataframe:

df2 = df.distinct() → drop/remove duplicate rows (all columns) from df.

df2 = df.dropDuplicates()

or

df.dropDuplicates("department", "salary").

** How to handle Bad or Corrupt records in Spark?

option1:

spark.read.option("badRecordsPath", "/tmp/")

option2: Permissive mode

spark.read.option("mode", "PERMISSIVE")

It will process corrupted data also.
but you will see some inaccurate results
like NULL etc. for such records.

option3: Dropmalformed mode

It excludes non-parsable records.

.option("mode", "DROPMALFORMED").

option4: Failfast mode:

Spark throws an exception & halts the
data loading process when it finds
any bad or corrupted records.

.option("mode", "FAILFAST").

option5: using columnNameOfCorruptRecord

spark.read.csv(columnNameOfCorruptRecord = "corrupted").

** multiple delimeter

after spark 3.0

spark.read.option("delimeter", "\|").csv(_)

** modify files in HDFS

HDFS is for write once
& read multiple times.
but for append, you can use
echo "your text" | hdfs dfs -appendToFile - /user/hduser/myfile.txt.

** fill null values in DF:

df.na.fill(value=0)
only a particular column.
df.na.fill(value=0, subset=["prices"])