* Map Reduce

→ Processing layer in hadoop.
→ Large datasets is processed in parallel by dividing the work into a set of independent tasks.
→ It is scalable & can be used across many computers.
→ Transforms list of input data elements into lists of output data elements.

• Terminologies

1. Map Reduce Job
→ 'A full program' is a an execution of Mapper and Reducer across dataset.
→ It consist of input data, MR program, and configuration info.

2. Task
→ It is an execution of a Mapper or a reducer on a slice of data.
→ Also called Task-in-progress (TIP)
→ Means processing of data is in progress either on mapper or Reducer.

3. Task attempt.
→ It is a particular instance of an attempt to execute a task on node.
→ By default task attempt is 4 i.e if any node during processing goes down

framework reschedules the task to some other nodes
→ If fails 4 times, then job is considered failed.

Config. cmds:
mapreduce.map.maxattempts &
mapreduce.reduce.maxattempts.

4. Map Abstraction
→ Takes key/value pair as input, whether data is structured or unstructured format.
- Key is reference to the input value.
- Value is the dataset on which to operate.

→ Processing
- Custom function defined by user
- Applies to every value in value input.

→ Output
- o/p of map is called intermediate o/p.
- Can be diff. from input pair.
- o/p is stored on local disk from where it is shuffled to reduce nodes.

5. Reduce Abstraction.
→ Takes intermediate key/value pairs as input. Usually we do aggregation or summation sort of computation.

→ o/p → It is the final o/p
It is written on HDFS.

Input (K|V) → mapper → shuffle & sort → reducer → O/P → HDFS.

• Map Reduce Working

1. Input files
→ Data req. for mapper is stored in i/p files which is in HDFS.

2. Input format
→ It defines how these input files are split & read. It selects files or other objects that are used for i/p.
- I/P format creates i/p splits.

3. Input Splits
→ It is the logical representation of data which will be processed by individual mapper.

No. of input splits = No. of map. tasks

4. Record Reader
→ It communicates with Input split & converts data into K|V pairs suitable for reading by mapper
- By default, it uses TextInputFormat for converting data into K|V pairs.
- It assigns byte offset (unique no.) to each line present in file.
Further this K|V pairs are sent to mapper.

5. Mapper
→ Processes each i/p record from record reader & generates new K|V pair (intermediate o/p) stored in local disk (temporary data).
→ Further sent to combiner.

6. Combiner
→ Also known as 'Mini-Reducer'.
- Performs local aggregation on mapper data, helping to minimize the data transfer b/w mapper & reducer.
- O/P is sent to partitioner

7. Partitioner
→ comes into picture if there are more than 1 reducer.
→ It takes o/p from combiner & performs partitioning
- Partitioning of o/p takes place on based of keys & then sorted.
- Records having same key value goes to same partition & then each partition is sent to reducer.

8. Shuffle & Sort.
→ O/P is shuffled to the reducer node.
- It is the physical movement of data, done over network.
- Once shuffled, then o/p is merged & sorted & sent to reducer.

9. Reducer
→ process the intermediate o/p of mapper & generates o/p which is stored on HDFS.

10. Record writer
→ It writes these o/p k/v pairs from reducer phase to o/p files.

11. Output format
→ The way these o/p k/v pairs are written in o/p files by recordwriter is determined by OutputFormat.

* i/p files → i/p format → i/p splits → RecordReader → Mapper → Combiner → partitioner → shuffling & sorting → Reducer → Record writer → o/p format.

→ Configuring no. of splits by setting mapred.max.split.size during job execution

→ No. of mappers = {(Total data size)/(i/p split size)} - determined by i/p format.

→ OpCollector.collect() method, writes o/p of reduce to File system.

→ Right no. of reducers :

0.95 or 1.75 × (<no. of nodes> * <no. of max. containers per node>)

0.95 → reducers launch immediately & starts transferring map o/p as the map finish.

1.75 → 1st round finished by faster nodes & 2nd wave of reducers is launched doing much better job of load balancing.

Increasing no. of reducers:
- inc. the framework overhead.
- Inc. load balancing.
- Lower cost of failures.

- Phases of reducers (shuffle, sort & reduce).

*→ Input Format
→ Functionality :
- The files or other objects that should be used for input is selected by i/p format.
- defines i/p splits, which defines size of individual map task & its potential execution server.
- defines RecordReader.

• Types
1. FileInputFormat
→ Base class for all file-based i/p formats.

2. TextInputFormat (Default format).
→ Useful for unformatted data or line-based records like log files.
→ K/V pair.

3. KeyValueTextInputFormat
→ It treats ~~entire~~ line by breaking to (key, value)
and splites by a tab character ('\t').
→ key → upto tab character
value → remaining after tab character.

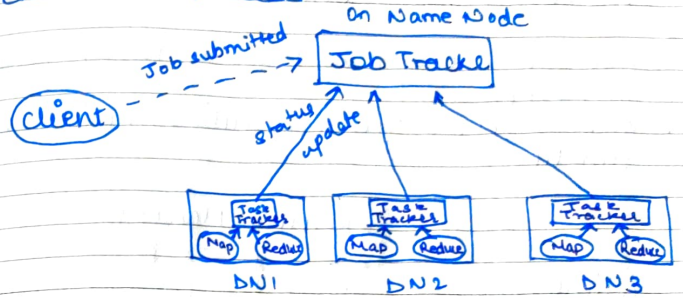
4. SequenceFileInputFormat
→ Reads sequence files (binary files that store seq. of binary K/V pair).
→ Seq. files block-compress & provide direct serialization & deserialization.
→ K/V are user defined.

5. SequenceFileAsTextInputFormat
→ Converts seq. file K/V values to Text objects by calling 'tostring'.

6. SequenceFileAsBinaryInputFormat
→ Using this we can extract the seq. file's keys & value as an opaque binary object.

7. DBInputFormat
→ It reads data from relational DB, using JDBC
→ Key → long Writable
Value → DBWritables.

• Execution Process



Job tracker : acts like master (responsible for complete execution of submitted Job)

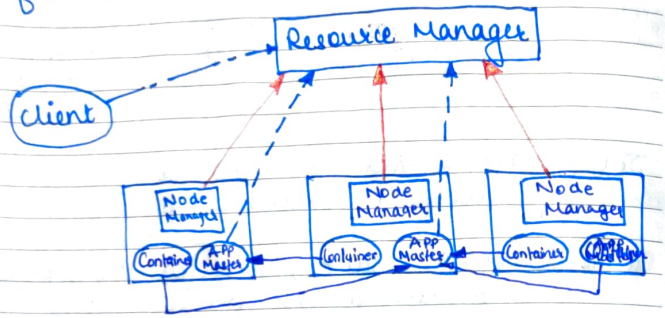
Task Trackers : Slaves, each of them performing the task given by job tracker.

★ → Job → Job Tracker (coordinate the activity by scheduling task to run on diff. data node) →
Information (task & code) given to task tracker → task tracker sends progress report to job tracker & also sends heartbeat signal for current state →
task is performed and written on HDFS.

→ If any task fails, job tracker reschedules it on diff task tracker.

★ YARN (Yet Another Resource Negotiator)

→ Resource management layer of Hadoop introduced in Hadoop 2.x.
→ Also performs job scheduling/monitoring function.



• Resource Manager : → Master Daemon, responsible for resource assignment among all applications.
- Has two components :
1. Scheduler
   - Responsible for allocating resources.
2. Application Manager
   - Functions :

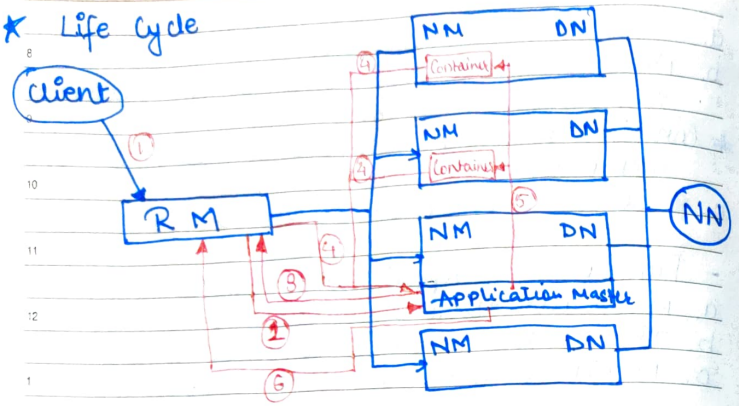
a. Accepts job submission
b. Negotiates the first container for executing Application master.
Container : Incorporates elements such as CPU, memory, disk & network.
c. Restarts application master on failure.

- Functions of Application Master.
a. Negotiates resource container from scheduler.
b. Tracks the resource container status.
c. Monitors progress of the application.

• Node Manager
→ Manages application & workflow & that particular node.
→ It registers with RM & sends heartbeat with the health status of node.
→ Monitors resource usage, performs log mgmt. & also kills a container based on dir? from RM.
→ Responsible for creating the container process & start it on the request of App. Master.

Flow :
→ Job → RN → Application Manager —req. container→ resource
Application master —resource req. thr. NM→ container (Task run)
After completion → Application manager unregisters itself from RM.

★ Life Cycle



(1) client submits job to RM
(2) In RM → App. Manager negotiates first container i.e App. Master on one of DN.
(3) The Application master requests resources req. for the job to be processed through NM.
(4) The container information is sent to App. Master & are launched in NN (resp) where data is present.
(5) The application master executes the task provided by client in the containers in resp. NN.
(6) After the completion the App. Master close it self (released its own containers).

If App. Master fails the RM reshedules the App. Master on diff. node.

★ Join in N-R.

→ Operation to join/combine two large datasets.
→ Joining starts by comparing the size of datasets
→ Smaller dataset is distributed to every DN.
→ During join, either mapper or reducer uses the smaller dataset to perform lookup for matching records from large dataset & then combines the records to form O/p.

Types of Join

(1) Map-Side Join (Performed by mapper).
- Join is performed before actual data is consumed by map function.
- The input to each map must be in form of partition & is in sorted order.

(2) Reduce-side Join (performed by reducer).
- No necessity of structured form (or partitioned).