Unit-8

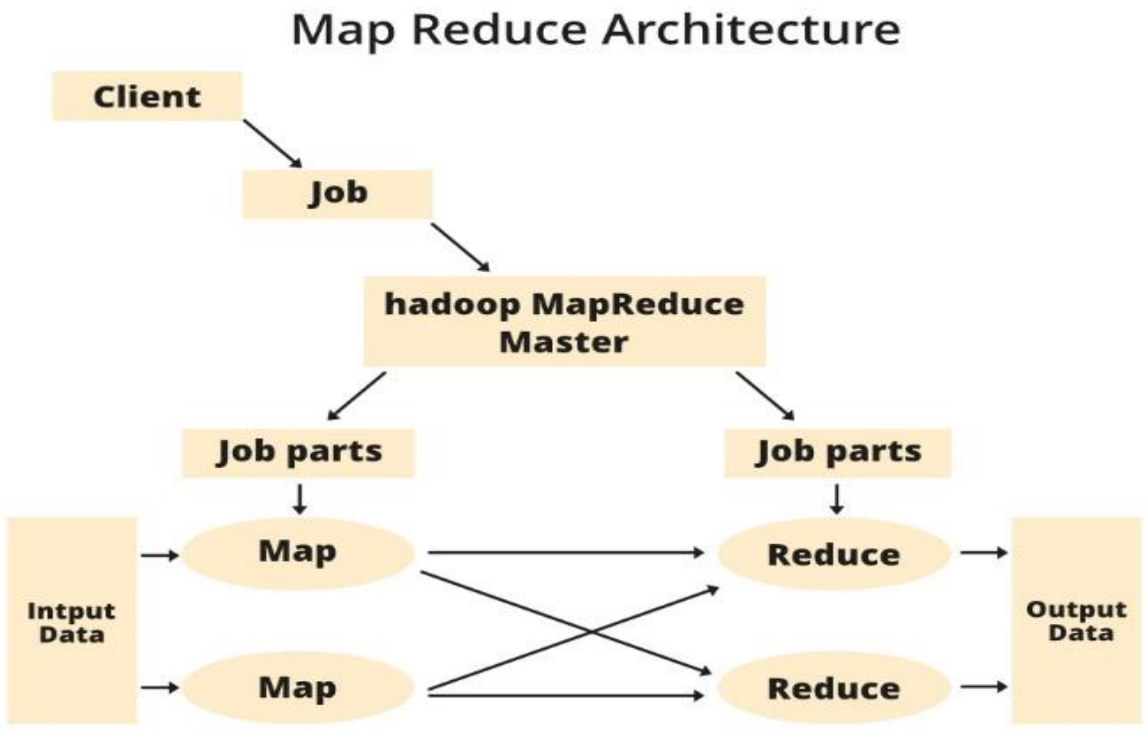
Processing Your Data with MapReduce

Topic covered

- Recollecting the Concept of MapReduce Framework
- Developing Simple MapReduce Application
- Points to Consider while Designing MapReduce.

Recollecting the Concept of MapReduce Framework

- MapReduce and HDFS are the two major components of Hadoop which makes it so powerful and efficient to use. MapReduce is a programming model used for efficient processing in parallel over large data-sets in a distributed manner. The data is first split and then combined to produce the final result. The libraries for MapReduce is written in so many programming languages with various different-different optimizations. The purpose of MapReduce in Hadoop is to Map each of the jobs and then it will reduce it to equivalent tasks for providing less overhead over the cluster network and to reduce the processing power. The MapReduce task is mainly divided into two phases Map Phase and Reduce Phase.


Map Reduce Architecture
Client
Job
hadoop MapReduce Master
Job parts
Job parts
Intput Data
Map
Map
Reduce
Reduce
Output Data

Components of MapReduce Architecture:

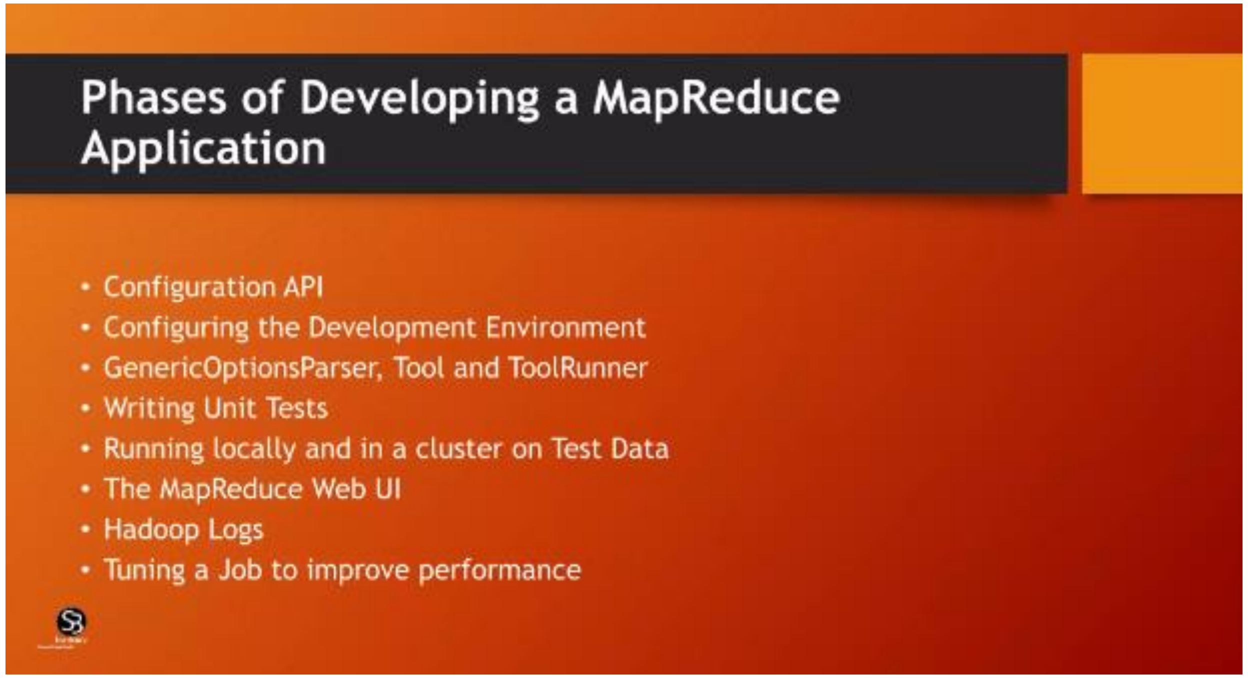
- **Client:** The MapReduce client is the one who brings the Job to the MapReduce for processing. There can be multiple clients available that continuously send jobs for processing to the Hadoop MapReduce Manager.
- **Job:** The MapReduce Job is the actual work that the client wanted to do which is comprised of so many smaller tasks that the client wants to process or execute.
- **Hadoop MapReduce Master:** It divides the particular job into subsequent job-parts.
- **Job-Parts:** The task or sub-jobs that are obtained after dividing the main job. The result of all the job-parts combined to produce the final output.
- **Input Data:** The data set that is fed to the MapReduce for processing.
- **Output Data:** The final result is obtained after the processing.

- In **MapReduce**, we have a client. The client will submit the job of a particular size to the Hadoop MapReduce Master. Now, the MapReduce master will divide this job into further equivalent job-parts. These job-parts are then made available for the Map and Reduce Task. This Map and Reduce task will contain the program as per the requirement of the use-case that the particular company is solving. The developer writes their logic to fulfill the requirement that the industry requires. The input data which we are using is then fed to the Map Task and the Map will generate intermediate key-value pair as its output. The output of Map i.e. these key-value pairs are then fed to the Reducer and the final output is stored on the HDFS. There can be n number of Map and Reduce tasks made available for processing the data as per the requirement. The algorithm for Map and Reduce is made with a very optimized way such that the time complexity or space complexity is minimum.

- The MapReduce task is mainly divided into **2 phases** i.e. Map phase and Reduce phase.
- **Map:** As the name suggests its main use is to map the input data in key-value pairs. The input to the map may be a key-value pair where the key can be the id of some kind of address and value is the actual value that it keeps. The *Map()* function will be executed in its memory repository on each of these input key-value pairs and generates the intermediate key-value pair which works as input for the Reducer or *Reduce()* function.
- **Reduce:** The intermediate key-value pairs that work as input for Reducer are shuffled and sort and send to the *Reduce()* function. Reducer aggregate or group the data based on its key-value pair as per the reducer algorithm written by the developer.

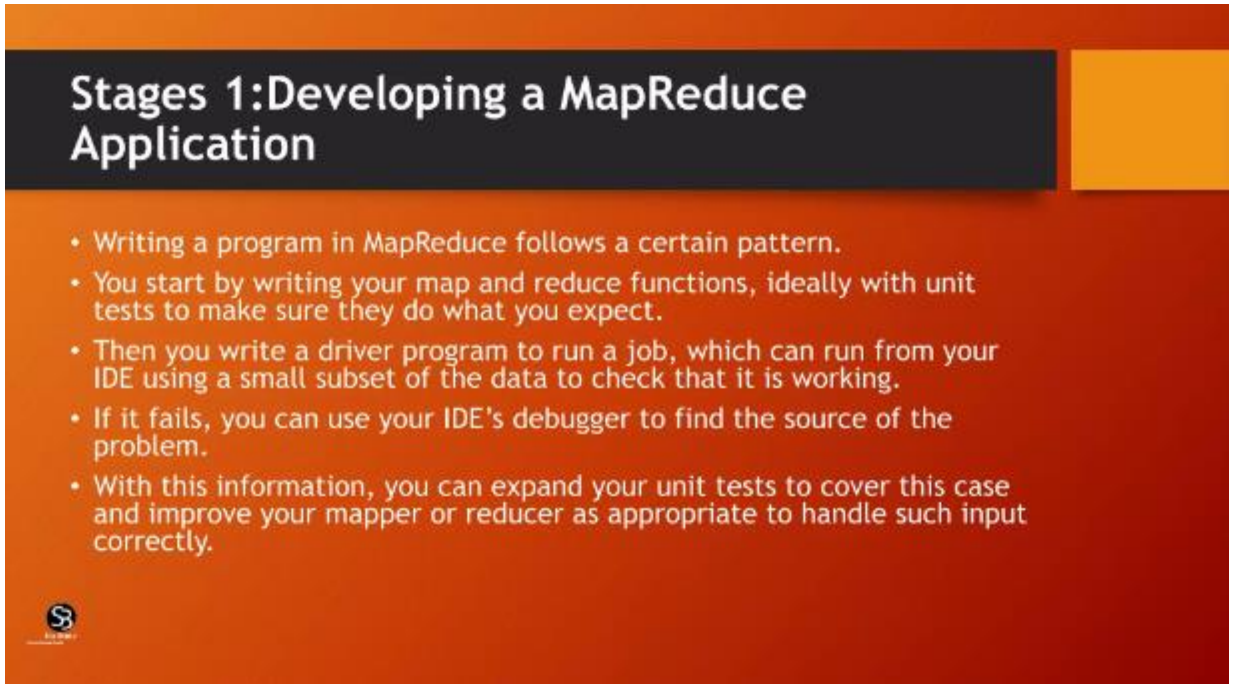
- How Job tracker and the task tracker deal with MapReduce:
- **Job Tracker:** The work of Job tracker is to manage all the resources and all the jobs across the cluster and also to schedule each map on the Task Tracker running on the same data node since there can be hundreds of data nodes available in the cluster.
- **Task Tracker:** The Task Tracker can be considered as the actual slaves that are working on the instruction given by the Job Tracker. This Task Tracker is deployed on each of the nodes available in the cluster that executes the Map and Reduce task as instructed by Job Tracker.
- There is also one important component of MapReduce Architecture known as **Job History Server**. The Job History Server is a daemon process that saves and stores historical information about the task or application, like the logs which are generated during or after the job execution are stored on Job History Server.

Developing Simple MapReduce Application



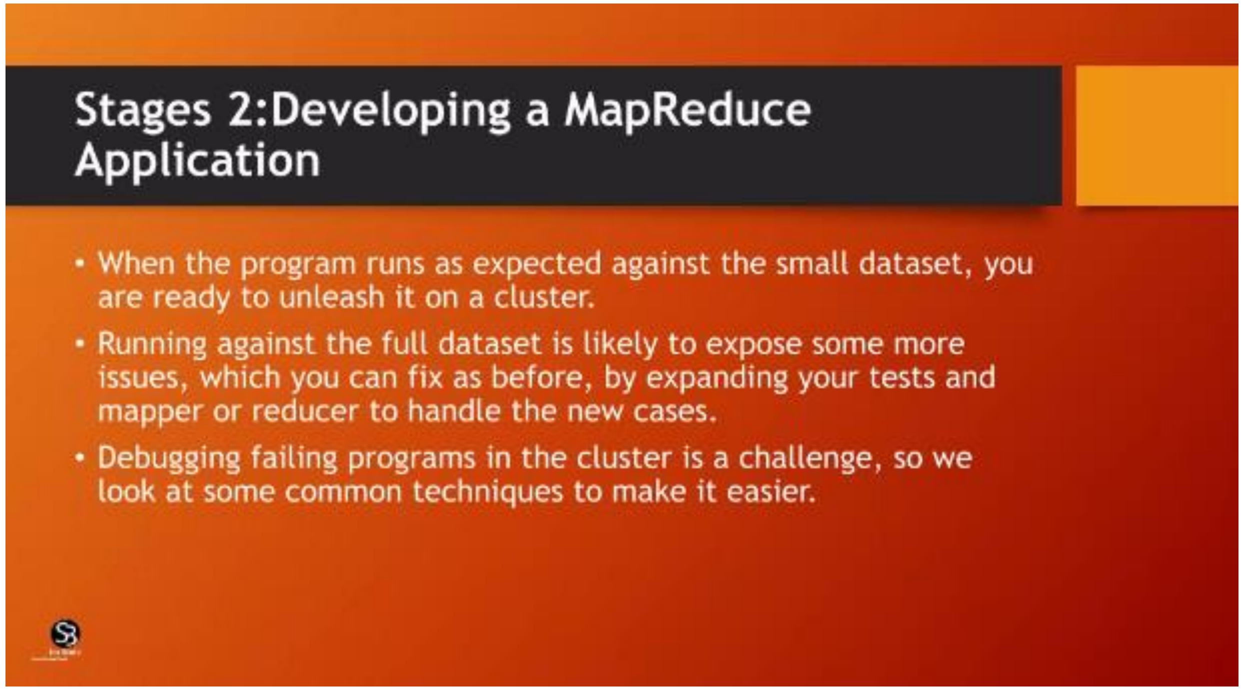
Phases of Developing a MapReduce Application

- Configuration API
- Configuring the Development Environment
- GenericOptionsParser, Tool and ToolRunner
- Writing Unit Tests
- Running locally and in a cluster on Test Data
- The MapReduce Web UI
- Hadoop Logs
- Tuning a Job to improve performance



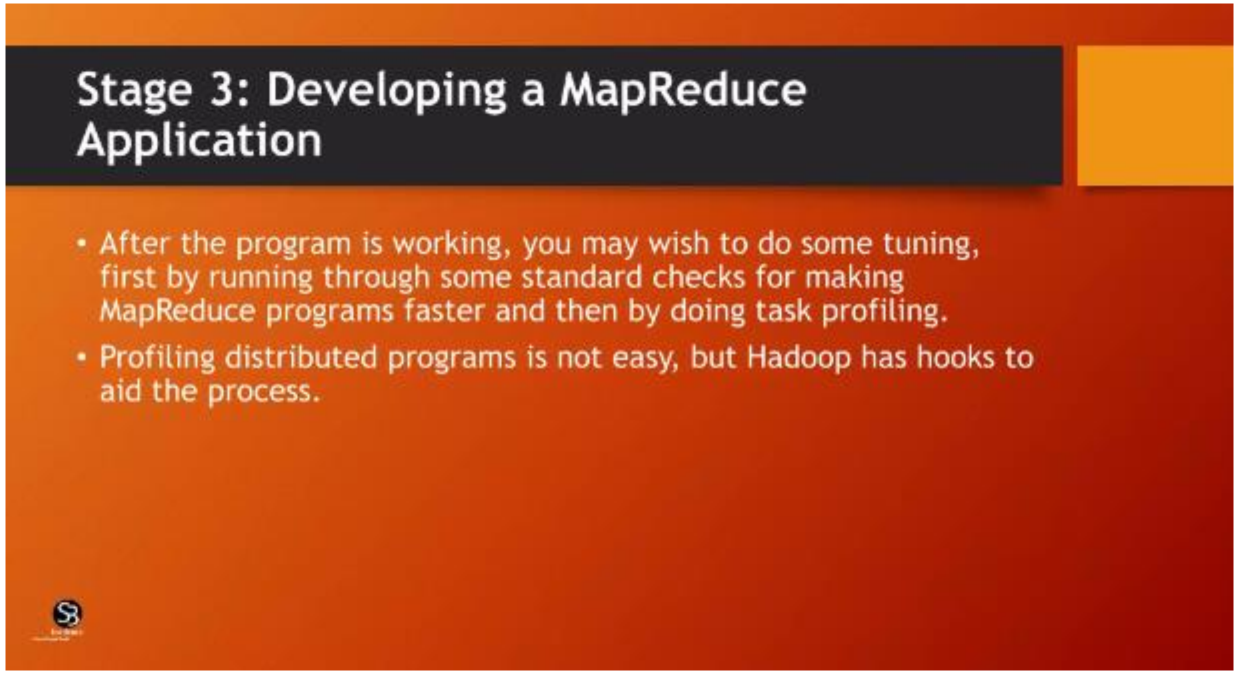
Stages 1:Developing a MapReduce Application

- Writing a program in MapReduce follows a certain pattern.
- You start by writing your map and reduce functions, ideally with unit tests to make sure they do what you expect.
- Then you write a driver program to run a job, which can run from your IDE using a small subset of the data to check that it is working.
- If it fails, you can use your IDE's debugger to find the source of the problem.
- With this information, you can expand your unit tests to cover this case and improve your mapper or reducer as appropriate to handle such input correctly.



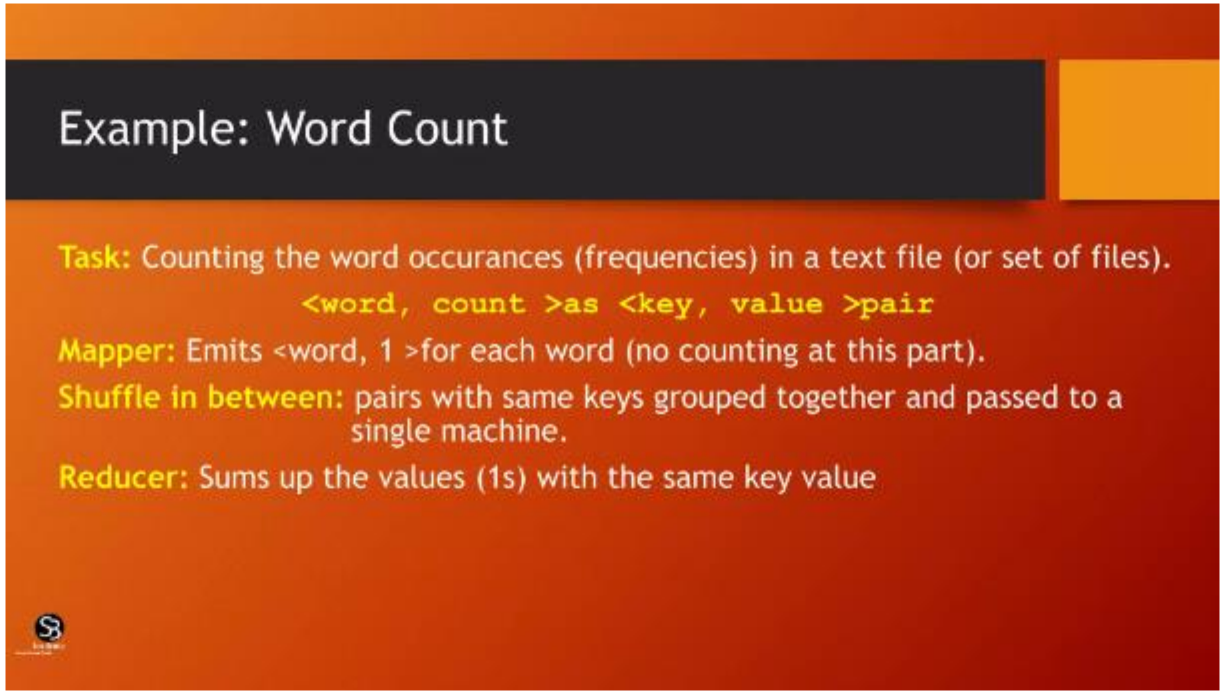
Stages 2:Developing a MapReduce Application

- When the program runs as expected against the small dataset, you are ready to unleash it on a cluster.
- Running against the full dataset is likely to expose some more issues, which you can fix as before, by expanding your tests and mapper or reducer to handle the new cases.
- Debugging failing programs in the cluster is a challenge, so we look at some common techniques to make it easier.



Stage 3: Developing a MapReduce Application

- After the program is working, you may wish to do some tuning, first by running through some standard checks for making MapReduce programs faster and then by doing task profiling.
- Profiling distributed programs is not easy, but Hadoop has hooks to aid the process.



Example: Word Count

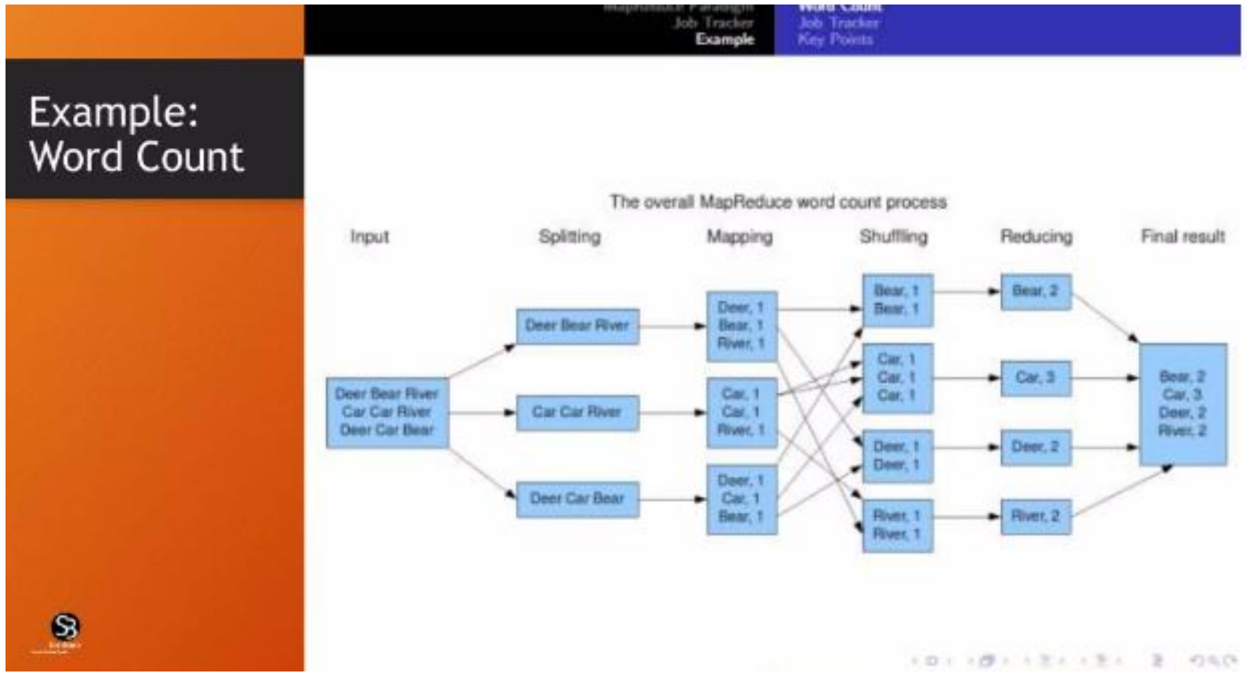
Task: Counting the word occurances (frequencies) in a text file (or set of files).

`<word, count >as <key, value >pair`

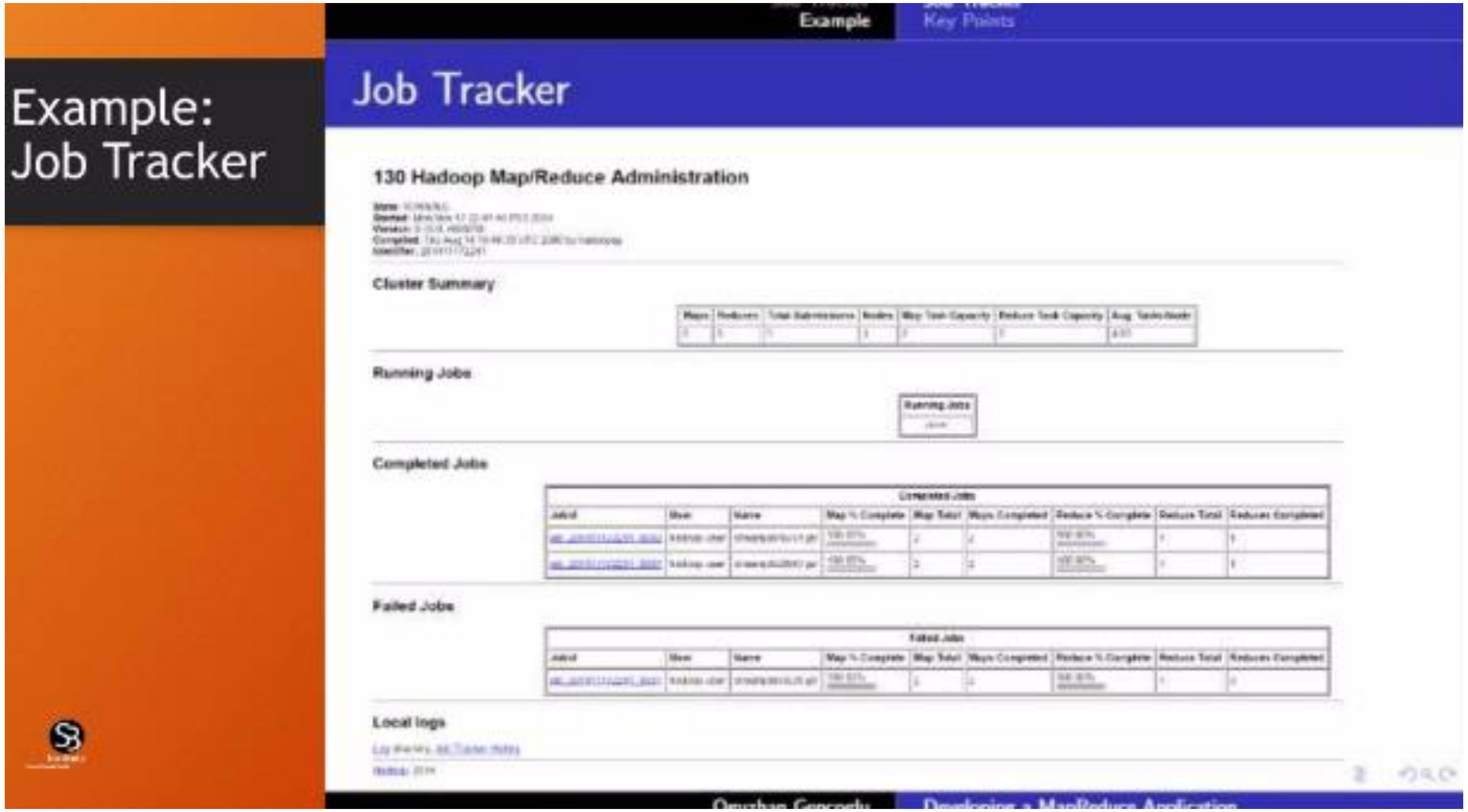
Mapper: Emits <word, 1 >for each word (no counting at this part).

Shuffle in between: pairs with same keys grouped together and passed to a single machine.

Reducer: Sums up the values (1s) with the same key value



Example: Word Count



Example

Key Points

Example: Job Tracker

Job Tracker

130 Hadoop Map/Reduce Administration

State: [illegible]
Started: [illegible]
Version: [illegible]
Compiled: [illegible]
Identifier: [illegible]

Cluster Summary

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Running Jobs

Running Jobs
[illegible]

Completed Jobs

Completed Jobs								
Jobid	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Failed Jobs

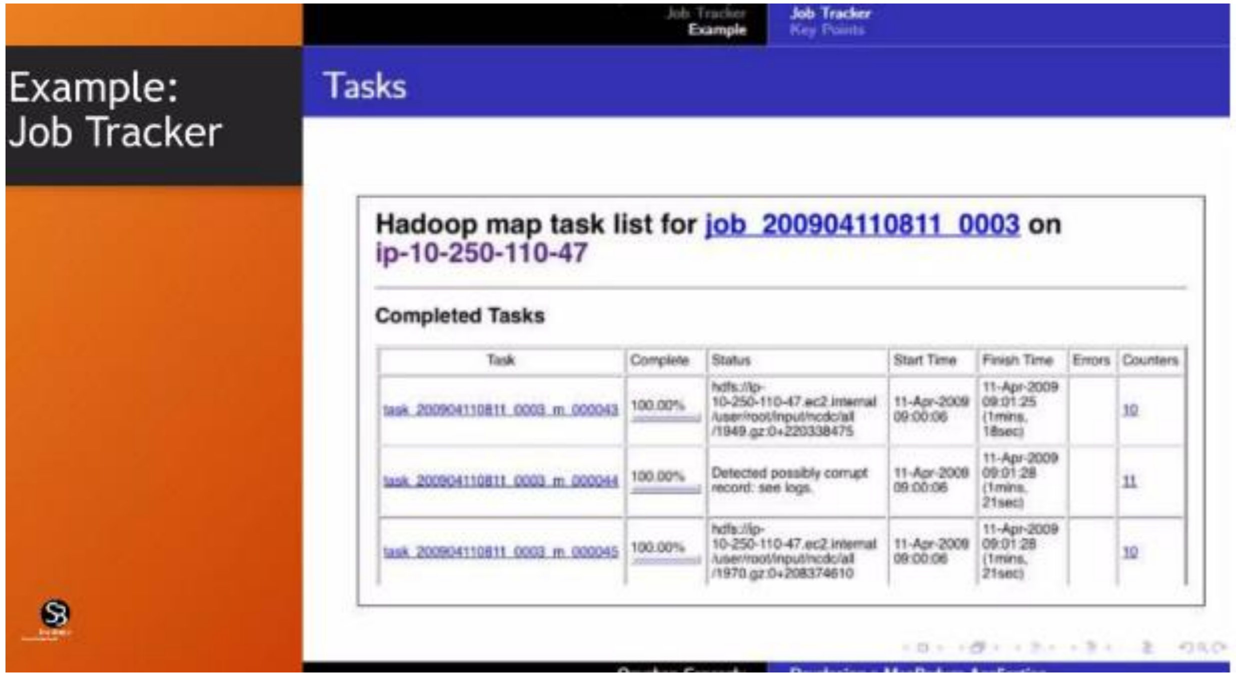
Failed Jobs								
Jobid	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Local logs

[illegible]

[illegible]

Developing a MapReduce Application



Job Tracker
Example

Job Tracker
Key Points

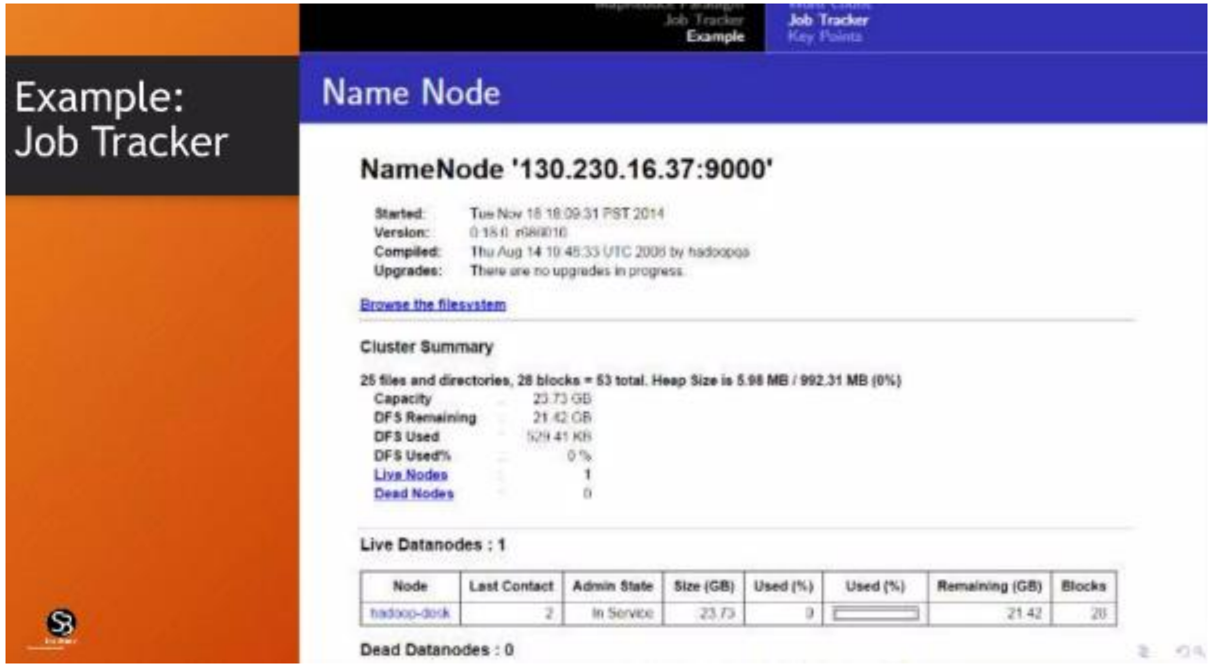
Example: Job Tracker

Tasks

Hadoop map task list for job_200904110811_0003 on ip-10-250-110-47

Completed Tasks

Task	Complete	Status	Start Time	Finish Time	Errors	Counters
task_200904110811_0003_m_000043	100.00%	hdfs://ip-10-250-110-47.ec2.internal/user/root/input/ncdc/all/1949.gz:0+220338475	11-Apr-2009 09:00:06	11-Apr-2009 09:01:25 (1mins, 18sec)		10
task_200904110811_0003_m_000044	100.00%	Detected possibly corrupt record: see logs.	11-Apr-2009 09:00:06	11-Apr-2009 09:01:28 (1mins, 21sec)		11
task_200904110811_0003_m_000045	100.00%	hdfs://ip-10-250-110-47.ec2.internal/user/root/input/ncdc/all/1970.gz:0+208374610	11-Apr-2009 09:00:06	11-Apr-2009 09:01:28 (1mins, 21sec)		10



Example:
Job Tracker

Name Node

NameNode '130.230.16.37:9000'

Started:	Tue Nov 18 18:09:31 PST 2014
Version:	0.18.0, r686010
Compiled:	Thu Aug 14 10:45:33 UTC 2008 by hadoopqa
Upgrades:	There are no upgrades in progress.

Browse the filesystem

Cluster Summary

25 files and directories, 28 blocks = 53 total. Heap Size is 5.98 MB / 992.31 MB (0%)

Capacity	:	23.73 GB
DFS Remaining	:	21.42 GB
DFS Used	:	529.41 KB
DFS Used%	:	0 %
Live Nodes	:	1
Dead Nodes	:	0

Live Datanodes : 1

Node	Last Contact	Admin State	Size (GB)	Used (%)	Used (%)	Remaining (GB)	Blocks
hadoop-desk	2	In Service	23.73	0	▭	21.42	28

Dead Datanodes : 0

Points to Consider while Designing MapReduce.

- MapReduce is a programming model for processing large datasets in parallel, distributed computing environments. When designing a MapReduce job, there are several important points to consider to ensure optimal performance and efficient use of resources. Here are some key considerations:
- Input data format: The input data format should be optimized for MapReduce. Typically, the data should be in a structured format such as CSV, JSON, or Avro. If the data is unstructured or in a proprietary format, preprocessing may be necessary to convert it into a structured format.

- Partitioning strategy: The input data should be partitioned into smaller, more manageable chunks. The partitioning strategy should be chosen based on the characteristics of the data and the resources available. For example, if the data is skewed, it may be necessary to use a custom partitioner to ensure that the data is evenly distributed among the reducers.
- Mapper function: The mapper function should be designed to process each record independently and in parallel. The function should be simple and efficient to maximize performance. It should also be designed to emit intermediate key-value pairs that can be processed by the reducer.

- Reducer function: The reducer function should be designed to process the intermediate key-value pairs generated by the mapper. Like the mapper function, the reducer should be simple and efficient. It should also be designed to minimize the amount of data transferred over the network.
- Combiner function: The combiner function is an optional optimization that can be used to reduce the amount of data transferred between the mapper and reducer. It should be designed to perform a partial reduction of the intermediate key-value pairs before they are sent to the reducer.

- Number of mappers and reducers: The number of mappers and reducers should be chosen based on the size of the input data and the available resources. The number of mappers should be chosen to maximize parallelism, while the number of reducers should be chosen to balance workload and minimize the amount of data transferred over the network.
- Fault tolerance: The MapReduce job should be designed to handle failures gracefully. This can be achieved by replicating data and using speculative execution to rerun failed tasks.
- Testing: Testing is an important part of the MapReduce design process. It is important to test the job on small datasets before running it on larger datasets to ensure that it performs as expected. It is also important to monitor the job during execution to identify and resolve any issues that arise.