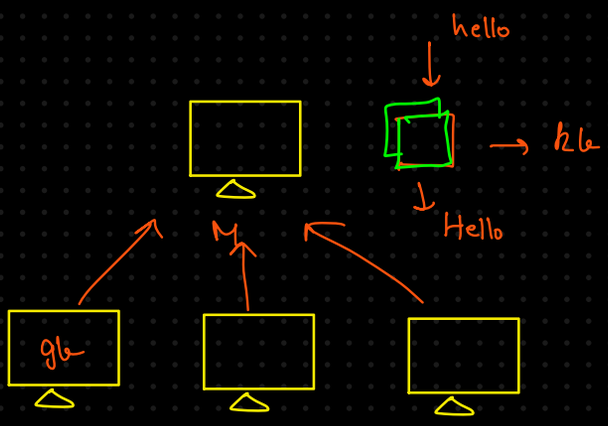
Map Reduce

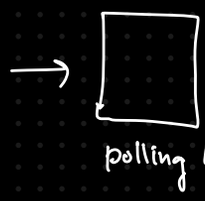
data → share ✓
process ✓

Code locality



data is moving towards the code

USA → election



Chrome → download

data locality
+
Concept of local processing
⟹ Send code to data

We are going to aim for distributed processing ⟹ Map Reduce Framework

Map Reduce GMR ⟶ MR

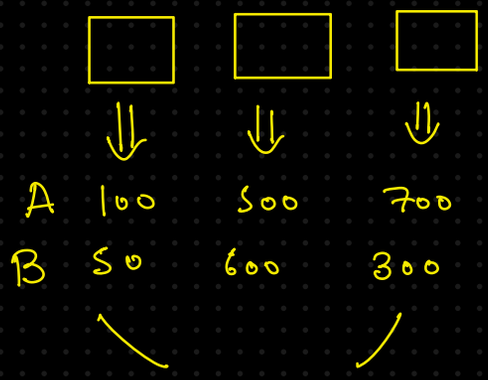
Map reduce is a programming model designed for processing large dataset in a distributed computing environment.

→ writ understanding distributed processing.

→ Spark ⇒ 70%

It simplifies handling massive data by dividing task into smaller, manageable subtasks and distributing them across multiple machine in a cluster.

eg → vote count
word count
product sales



1. We are going to have data in a distributed manner via hdfs.

→ Map

→ Reduce

input data → split into chunk → mapper → intermediate output → reducer → final output

each and every input and output is going to be in a key-value pair

(K , V)

Traditional lang code like we have been writing doesn't work in a distributed way.

→ Map and Reduce

→ Reducer work after mapper phase

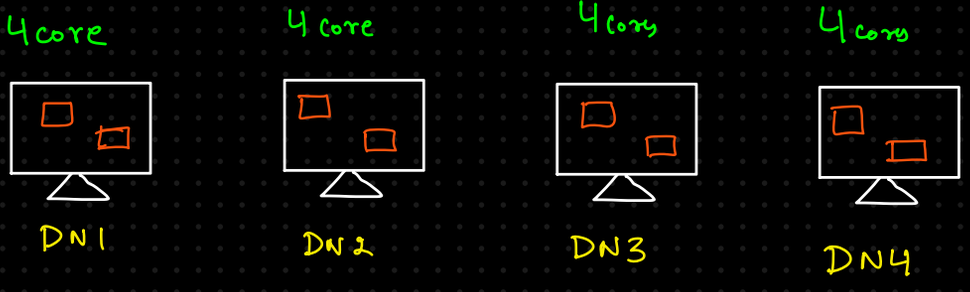
→ block ⟶ map ⟶ Intermediate O/P ⟶ Reducer ⟶ final output
(K, V)

Map Reduce and Cluster

400 mb

4 node cluster

DN cpu + ram + hdd



1 cpu core can process 1 mapper Job on 1 block / input split of data

32 blocks → we can process 16 core via above cluster ⇒ we will not be failing the processing rather it is going to happen in batches $ will take time

if we decrease block size ⇒ parallelism ↑↑

2 blocks ⇐ 512 mb ↵ 1gb → 128 mb block size = 8 blocks

↳ 64 mb ⇒ 16 blocks

↳ 4 mb ⇒ 64 blocks

1. Distributed Processing
2. Fault tolerant
3. Data Locality
4. Scalability

Map Reduce - Word Count example

We want to count the occurence of each word in a file via the map reduce framework.

1. Input data (input.txt)

(K,V) record reader converts the input to (K,V) pair for mapper input

2. Mapper

no. of mapper task = no. of blocks

Mapper input → (1, gohu vegeta friezа)

(1, gohu vegeta gohan)

↓

(gohu,1)
(vegeta,1)
(gohan,1)

(Gohu,2),(vegeta,1)

Mapper output

3. Combiner (Optional)

Without combiner: mapper sends all the key value pairs to the reducer, increasing shuffle data

With combiner: Partial aggregation at each mapper to reduce shuffle data

((Gohu,1),(Gohu,1)) ⟶ (Gohu,2)

4. Shuffle and sort phase

→ data from all mapper is shuffled and grouped by Key

Gohu {1,1,2,1}
Gohan. [1,1,1]

→ Keys are sorted for efficient reduction

5. Reducer

Each reducer processes the grouped Key value pair and aggregate value.

We can control the number of reducer

6. Final Output

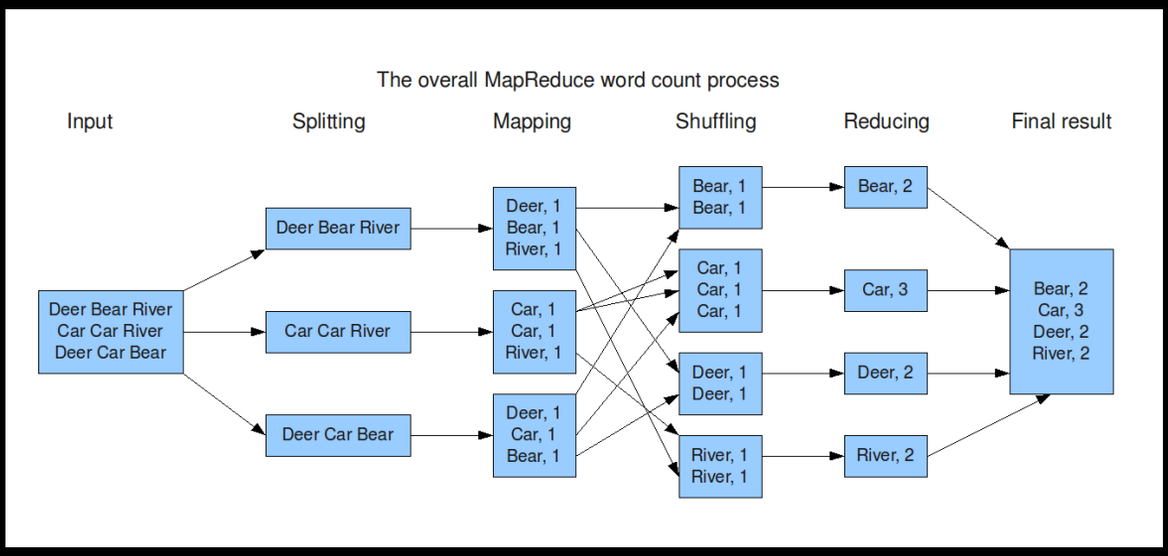
final output is written to hdfs or any other location.

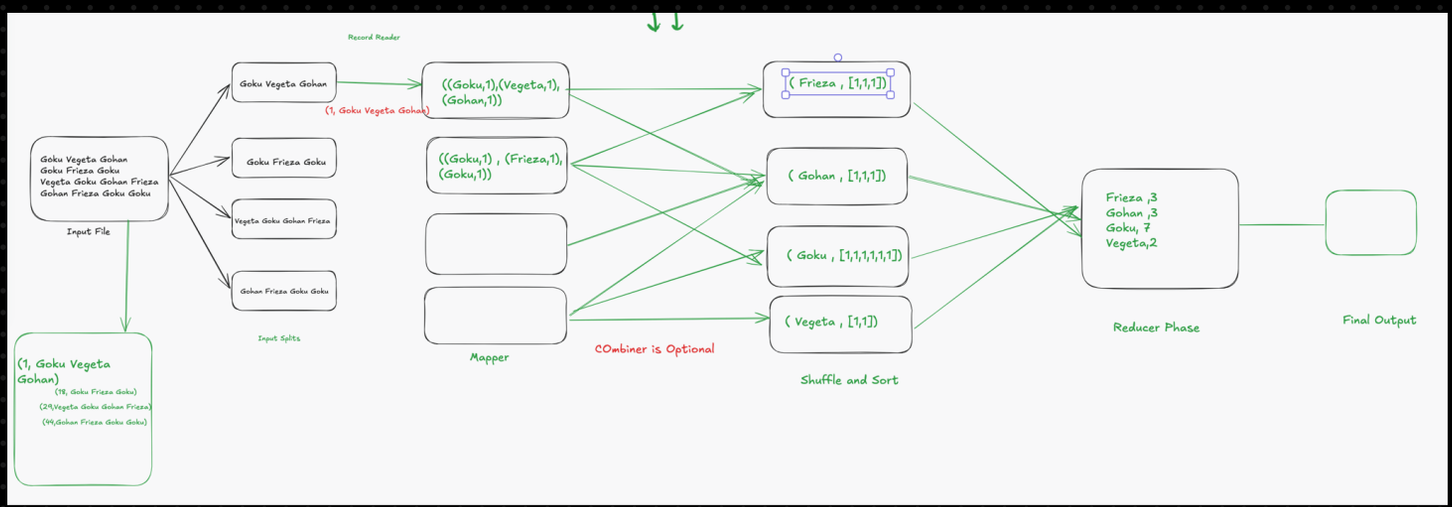
Reducer Input [Goku , [1,1,1])

↓

Reducer output [Goku , 3]

Step	Process	Output Example
Input Splitting	Splitting input file into HDFS chunks.	Split 1: `Goku Vegeta Gohan`, Split 2: `Goku Frieza Goku`, etc.
Mapper Phase	Generating key-value pairs (`word`, `1`) from each split.	`("Goku", 1), ("Vegeta", 1), ("Gohan", 1)`
Combiner Phase	Optional local aggregation at Mapper to reduce shuffle data.	`("Goku", 2), ("Frieza", 1)`
Shuffle and Sort	Grouping and sorting intermediate key-value pairs by key.	`("Goku", [1, 2, 1, 2]), ("Frieza", [1, 1, 1])`
Reducer Phase	Aggregating grouped data to produce final results.	`("Goku", 6), ("Frieza", 3)`
Final Output	Writing final aggregated results to HDFS.	`Frieza 3`, `Gohan 3`, `Goku 6`, `Vegeta 2`


The overall MapReduce word count process
Input
Splitting
Mapping
Shuffling
Reducing
Final result
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, 1
Bear, 1
Car, 1
Car, 1
Car, 1
Deer, 1
Deer, 1
River, 1
River, 1
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2


Record Reader
Goku Vegeta Gohan
(1, Goku Vegeta Gohan)
((Goku,1),(Vegeta,1),
(Gohan,1))
Goku Vegeta Gohan
Goku Frieza Goku
Vegeta Goku Gohan Frieza
Gohan Frieza Goku Goku
Input File
Goku Frieza Goku
Vegeta Goku Gohan Frieza
Gohan Frieza Goku Goku
Input Splits
((Goku,1) , (Frieza,1),
(Goku,1))
Mapper
COmbiner is Optional
(Frieza , [1,1,1])
(Gohan , [1,1,1])
(Goku , [1,1,1,1,1,1])
(Vegeta , [1,1])
Shuffle and Sort
Frieza ,3
Gohan ,3
Goku, 7
Vegeta,2
Reducer Phase
Final Output
(1, Goku Vegeta
Gohan)
(18, Goku Frieza Goku)
(29,Vegeta Goku Gohan Frieza)
(49,Gohan Frieza Goku Goku)

Map Reduce - Practical on Hadoop Cluster

⇒ Normal Map Reduce with 1 reducer

By default, Hadoop uses a single reducer if not specified in the job configuration.

hadoop jar name — input file — input file

output dir → should not be created.

_SUCESS is the file which indicates that the job was successful

Steps:

1. We have a input file
2. we wrote the MR code in Java
3. we created the Java Jar from the Wordcount.java
4. Sent the input file → hdfs
5. we ran MR via hadoop jar — jar — name — input — output
6. we are able to see the results.

2 important things

1. Ofcourse MR is optimized for large inputs.
2. A single reducer can get bottlenecked or overloaded by lots of data.

Map Reduce Practical : More than one Reducer

2 tb ⟶ 1 tb ⟶ single reducer ↗ bottleneck
↘ time consuming

We can have multiple reducers to split the workload, parallel processing of grouped keys is achieved via this

1. We have to make sure that we do/employ mappers to achieve parallelism & do maximum heavy lifting

10 mapper → 10 min
1 reducer → 40 min
inc. ⇓
8 reducer ⇒ 5 min

50 min ⟶ 15 min

2. We cannot have chain of reducers layers
We can have a single reducer layer

Partitioning

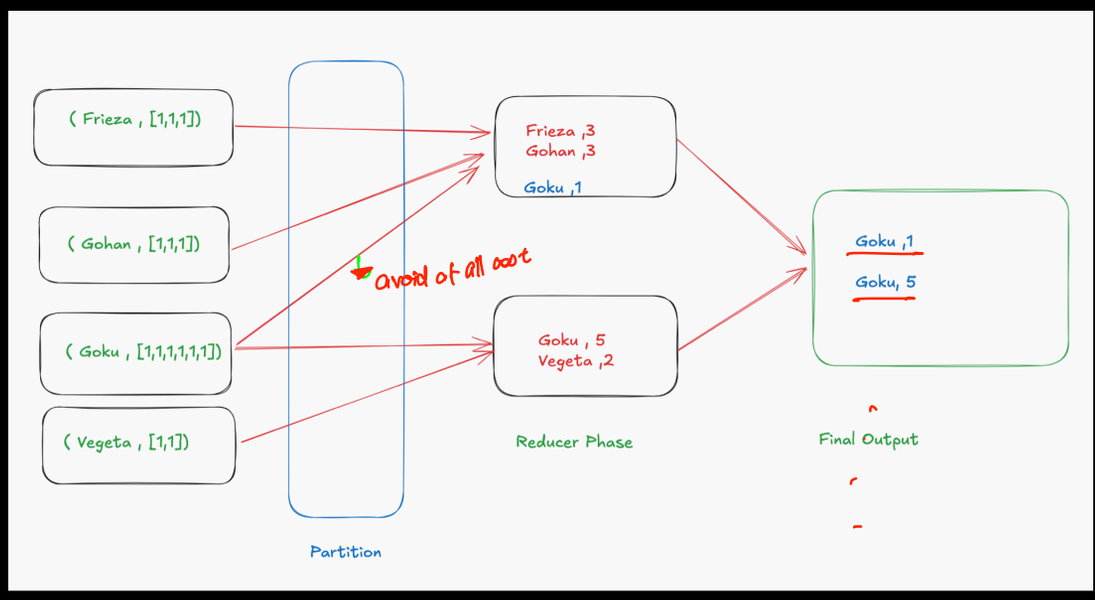
determine how intermediate keys are distributed to reducers.

> ensure that same key goes to same reducer

hashing

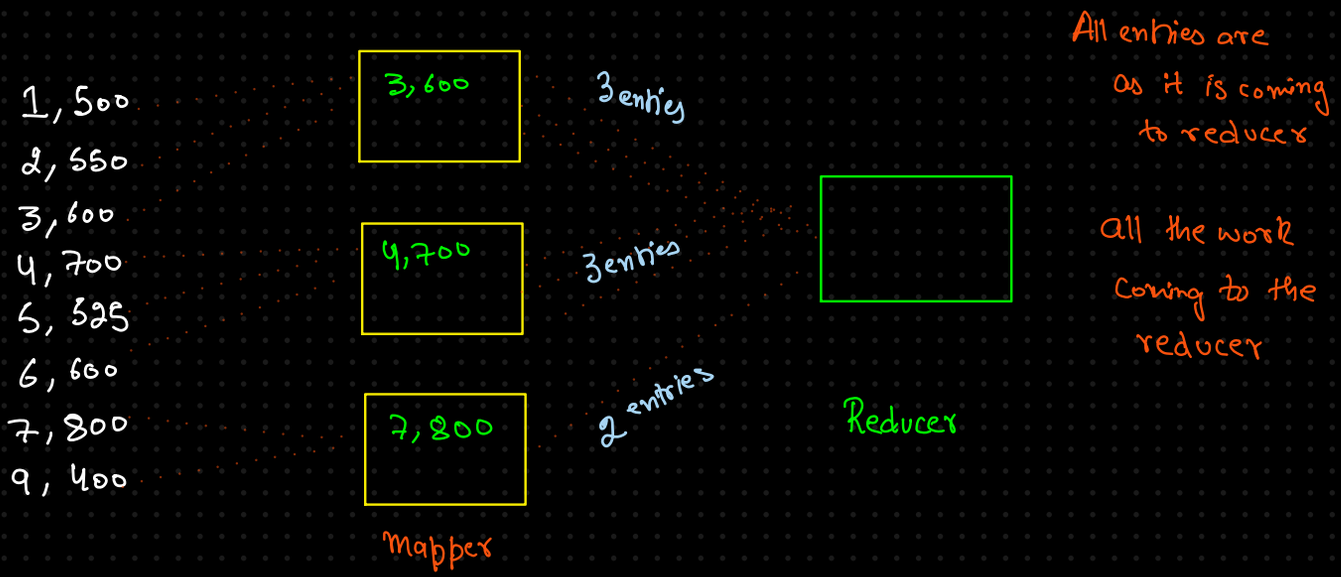
modulus operator
%

a-m → 1
n-z → 2



Combiner

⇒ A mini-reducer that perform partial aggregation at mapper level.



Using Combiner →

We should have calculated local answer in each of the mapper.

& then our reducer would just have to check for 3 entries.

It reduces the amount of data getting moved over the network

Q: Can we always use Combiner?

Find the median, average

So for non-associative or non commutative operation we cannot use Combiner.

main priority is getting right

Map Reduce with Zero Reducer

Partition

Combiner

→ when the output of mapper layer is the final output

→ filtering, data transformation or format conversion.

Goku Vegeta Frieza $\xrightarrow{>4}$ Vegeta Frieza

When we dont require any aggregation.

Map Reduce on a log file + Industry Use case

Log file

filtering, group by

hive, Piglatin ⇒ abstraction of MR

Amazon

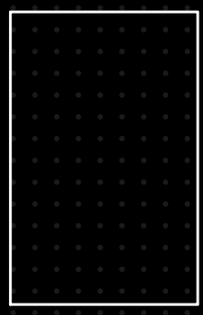
Iphone — Phone, Mobile, Apple Handheld phone

Samsung —

Huawei —

Nokia —

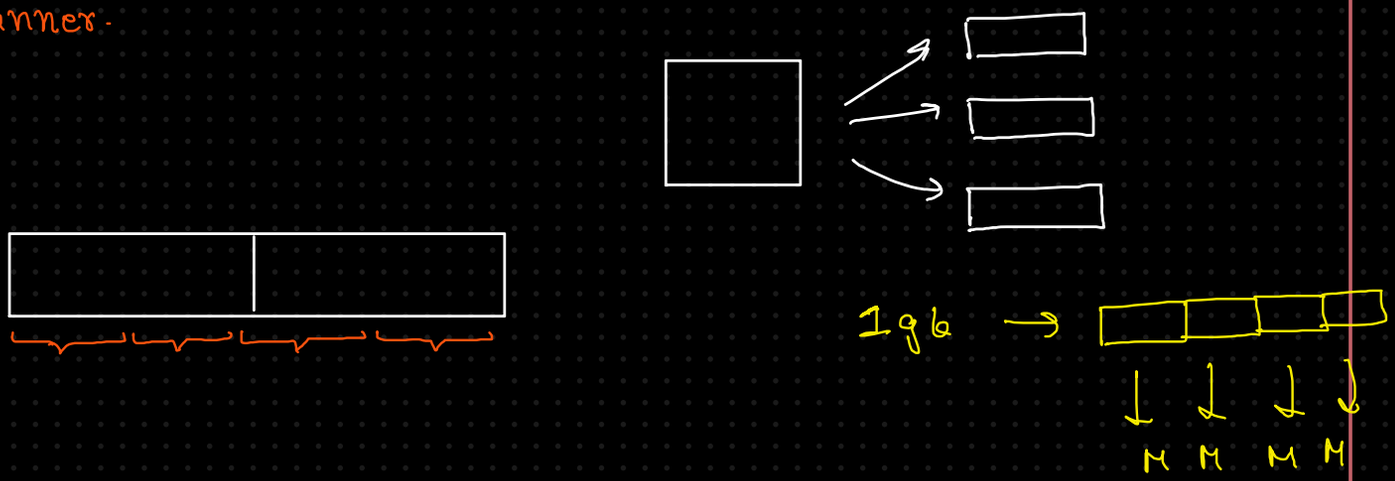
TV



Input Split

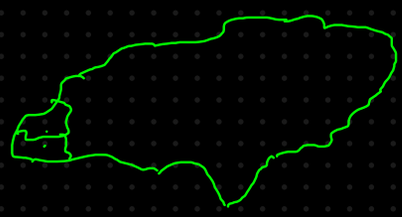
block / input split

An Input split is a logical division of the input data used by Hadoop MR to process large dataset efficiently in a distributed manner.



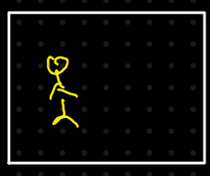
One core can run one mapper job on one split of data.

eg ⇒ Countries on earth / land

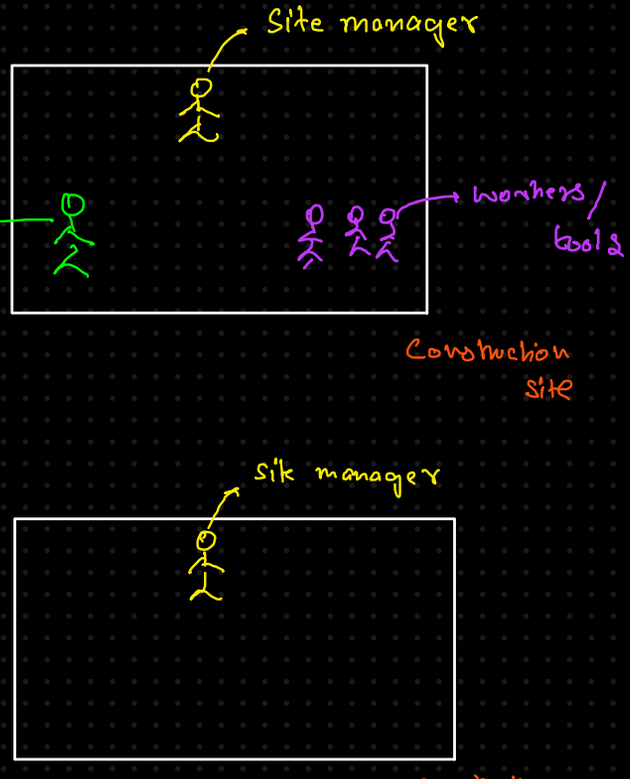


YARN analogy

Construction

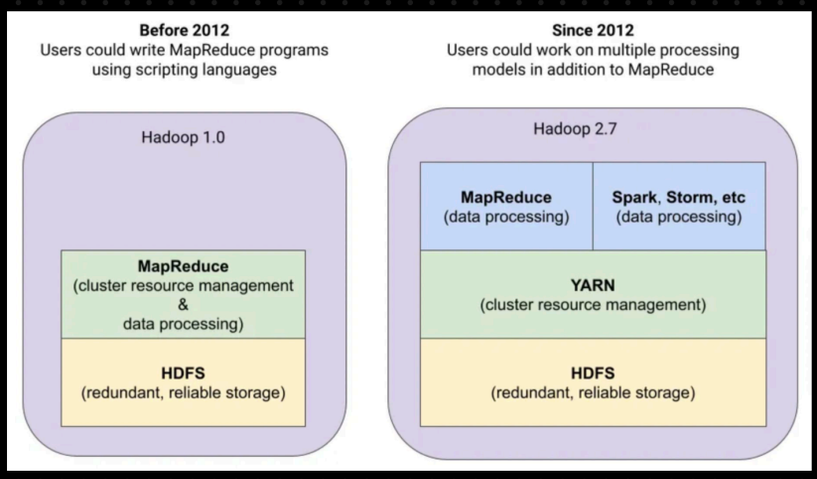


Central office



★ interview

YARN (Yet Another Resource Negotiator)



Interview
Spark

still getting used
↓
dataproc cluster

resource management layer in hadoop.

components of Yarn

1. Resource Manager: master daemon of YARN — Central office
   Allocate resources
   Efficiently resource usage across jobs

   (a) Scheduler
   (b) Application Manager: manages application in our cluster
   Application master

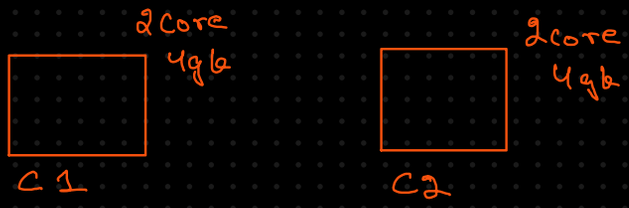
2. Node Manager: updates everything to Resource manager — Site manager
   keeps the data to RM updated
   Container

3. Application Master: ≈ no. of jobs
   negotiates resources with RM
   work with node mager for executing and monitoring the task — project manager

4. Containers : physical resource (RAM, core, disk)

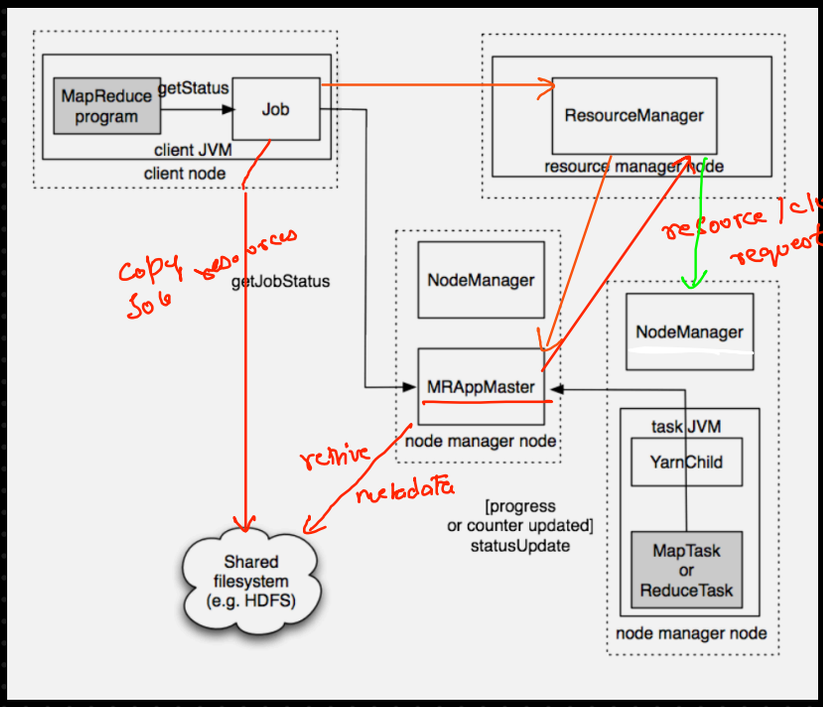
team of workers $ tools

executes the task assigned by the Node manager



YARN Process - Step by Step process

1. Job Submission
2. Launching of Application Masters
3. Requesting Containers
4. Allocating Resources (RM → NM)
5. Executing tasks
6. Monitoring Progress
7. Job completion



RM needs to free up the resources for future jobs.