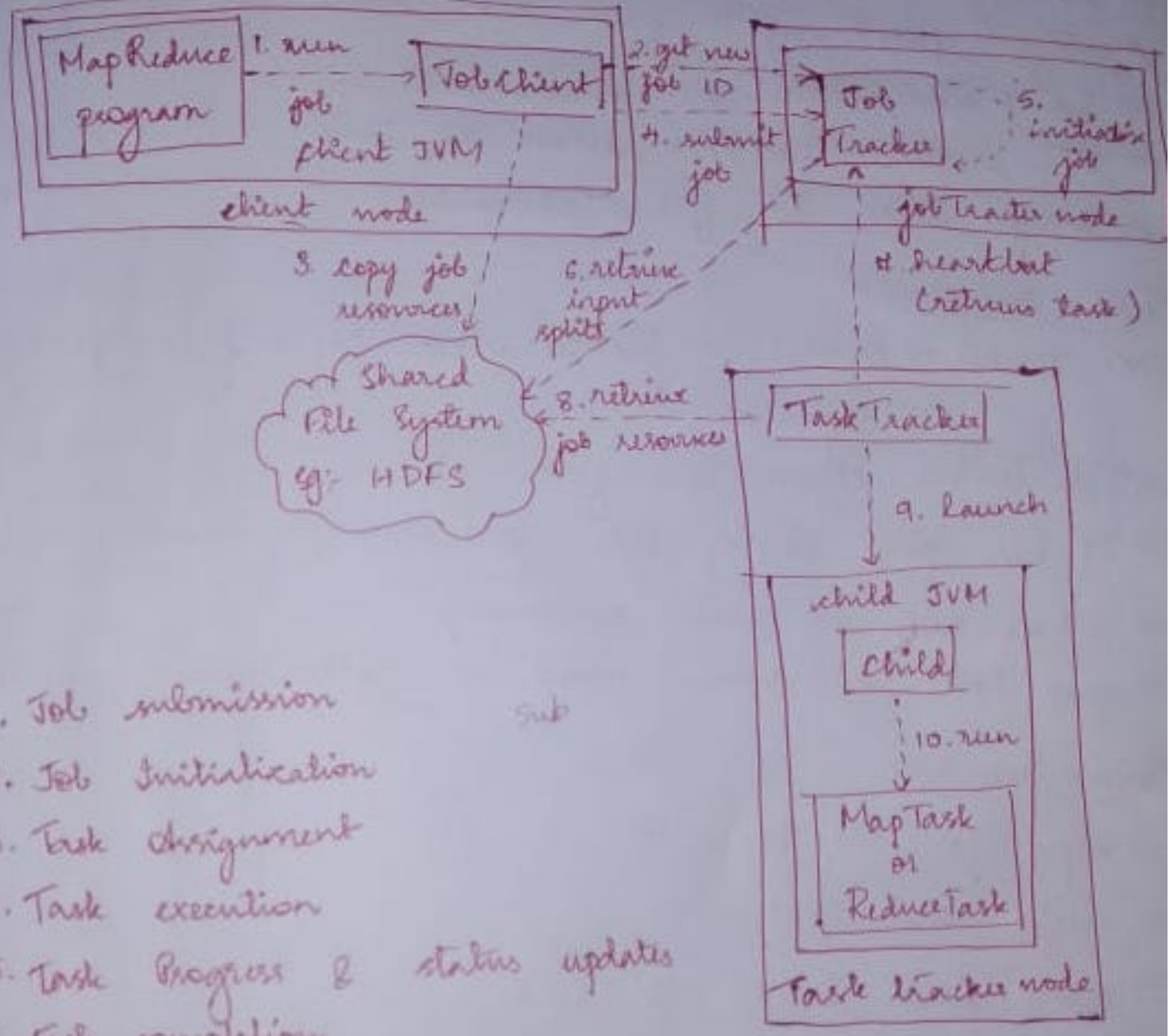
UNIT - 3

Anatomy of classic map reduce job run



1. Job submission sub
2. Job Initialization
3. Task assignment
4. Task execution
5. Task Progress & status updates
6. Task completion

Job submission

→ ask JobTracker for new job ID

→ check o/p specification of job

→ computes i/p splits for job

→ copy resources ~~to~~ needed to run job

→ tell job tracker that job is ready for execution

Job initialization

→ job object is created

→ retrieves input splits from HDFS, create list of tasks.

→ Job^{JT} tracker creates one map - each split

→ JT creates setup task, clean up task

→ assign tasks to free TaskTrackers

Task Assignment

→ Job tracker → chooses a job to select task from - scheduling algorithm

→ Task^{TT} tracker chooses task from job

→ TT has fixed no. of slots for map & reduce

→ reduce task is chosen randomly

→ map tasks is chosen based on data locality, n/w location

Task Execution

→ copy jar file from HDFS

→ creates local wd

→ create task tracker for task

→ tasks on same JVM runs setup task and cleanup task

→ heartbeat

Progress & status updates

→ Mapper & reducer to TaskTracker - setting a flag

→ heartbeat

→ JT combines all updates

→ Client uses getStatus()

Job completion

- → JT changes status to successful
- → sends HTTP notification to client which cas waitforCompletion().
- → client console — job statistics & counter info
- → JT & TT - clean up action for job

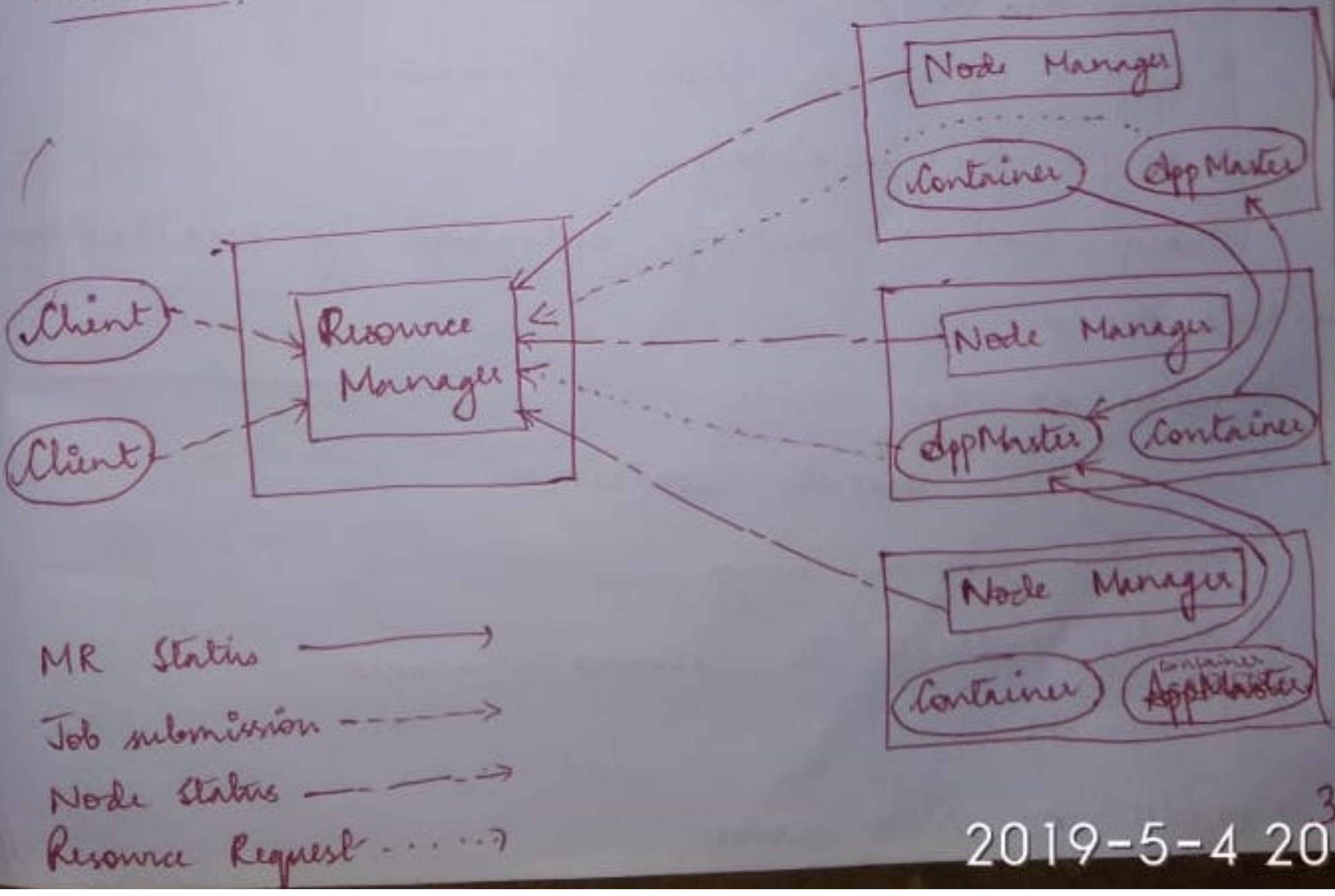
{YARN (Hadoop MR2)}

Limitations

- → Scability < max. cluster size - 4000 / max. concurrent tasks - 40000
- → Supports single of failure
- → restrart is very tricky

YARN

Architecture



Components

→ Resource Manager

↳ Resides on master node

↳ Manages resource scheduling for dif computing applicati
in optimum way

↳ coordination scheduler and ApplicationMaster

• Scheduler

↳ resides on mater node

↳ schedules job execution as per submission request

↳ allocates resources to applications

↳ coordinates with application master, keeps track of
resources of running applications

• Application Master

↳ resides on master node

↳ helps & coordinates with scheduler

↳ accepts job submissions

↳ keeps track of running application by coordiantion

→ Node Manager

↳ resides on slave nodes

↳ manages & executes containers

↳ monitors resource usage

↳ sends heartbeats to resource manager

→ Application Master

↳ resides on slave nodes

2019-5-4 20:4

↳ per application, that is, if multiple jobs are submitted can have? more than one instance of App Master on slave node.

↳ negotiates, suitable resource containers on slave node from RM

↳ works with one or more NMs to monitor task execution

YARN	MR
→ supports variety of processing engines & applications	→ supported own batch processing applications only
→ separates duties across multiple components	→ consolidated most of work in single component
→ can dynamically allocate polos of resources to applications	→ provides static allocations of resource for designated tasks

Needs

→ offers scalability, resource ultilization, high availability and performance improvements

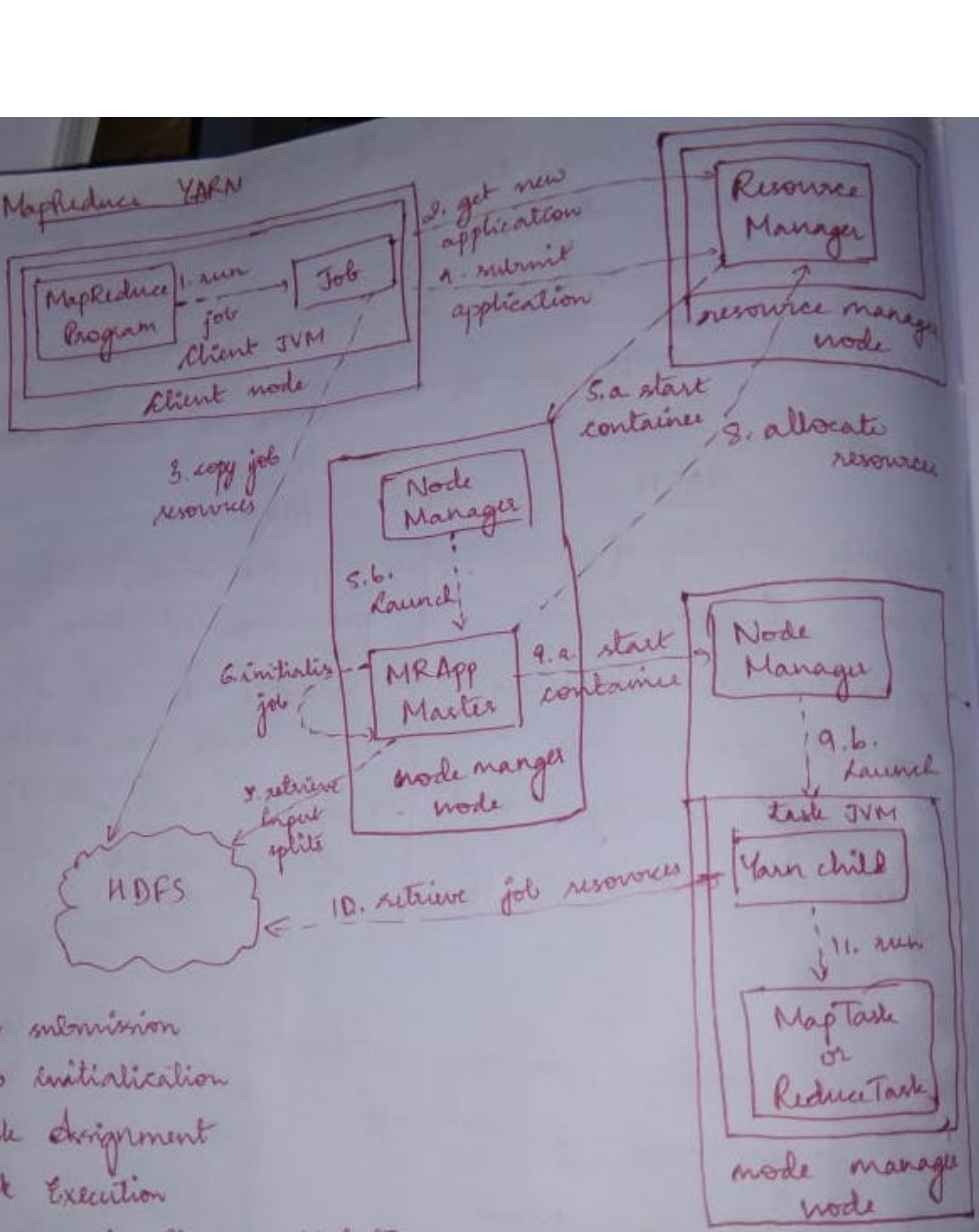
→ opend uses for HBase

→ separates HDFS from MR

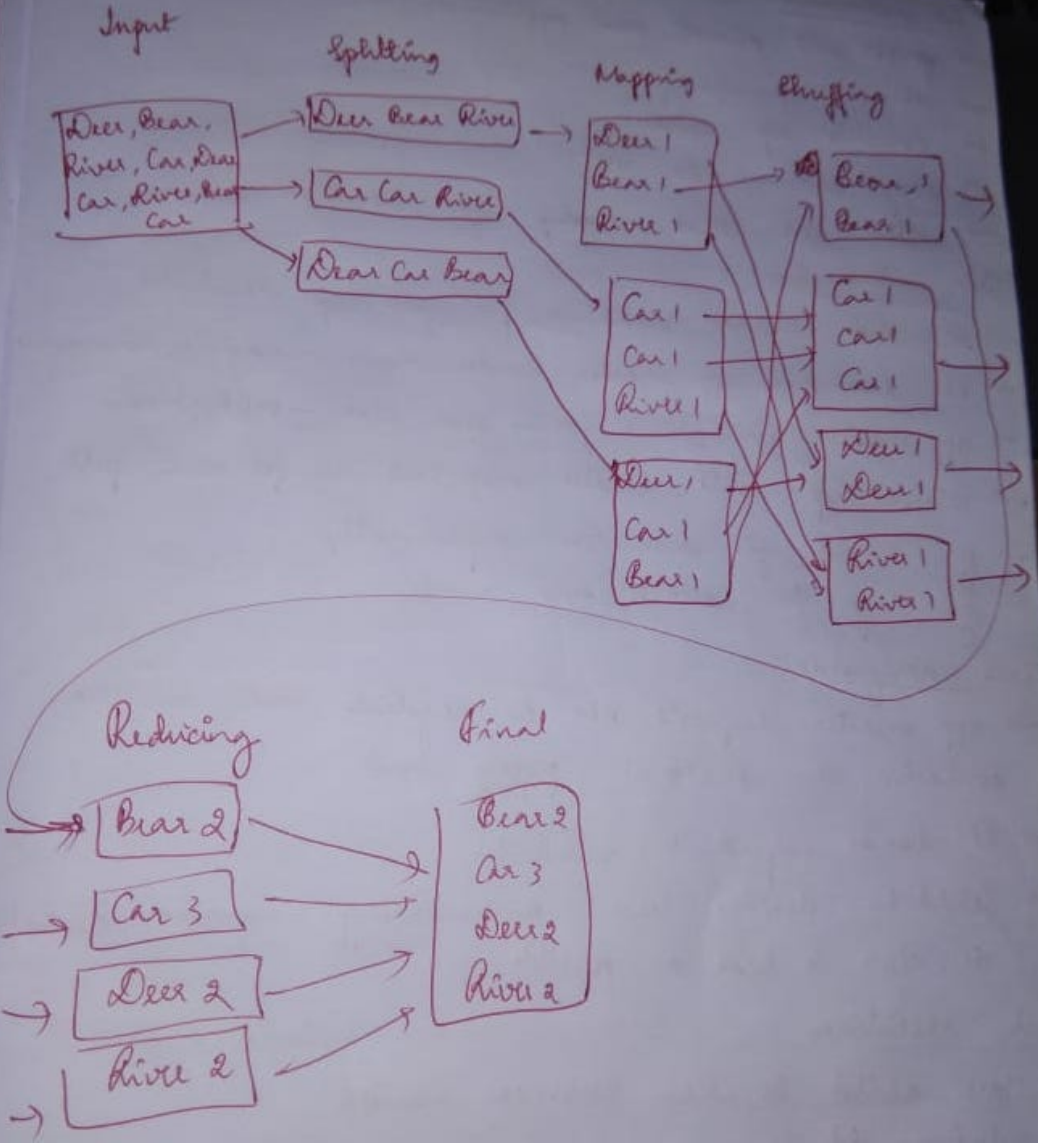
→ manages resources in clustered envt.

→ interacts with compute resources (on behalf of application) and assigns resources to each application based on filtering criteria.

MapReduce YARN



1. Job submission
2. Job Initialication
3. Task Assignment
4. Task Execution
5. Progress & Status Updates
6. Job completion


Input
Splitting
Mapping
Shuffling
Deer, Bear, River, Car, Dear Car, River, Bear Car
Deer Bear River
Car Car River
Dear 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
Reducing
Final
Bear 2
Car 3
Deer 2
River 2
Bear 2
Car 3
Deer 2
River 2

Job submission

→ RM will allocate new application ID
→ check o/p specification
→ compute i/p splits
→ copy resources to HDFS
→ notify RM that ready for execution

Job initialization

→ allocated resource container for job by scheduler
→ RM launches app master under node manager's managem
→ app master initializes job — java class — MRAppMaster
→ retrieves i/p splits, creates map task obj for each split
→ for small job, own JVM sequentially
→ for big job launch new node

Task assignment

→ app master requests RM to negotiate more resource container on heatbeat- piggy back
→ RM hands request to scheduler
→ Scheduler decides based on memory requirement, data as close to task as possible

Task execution

→ Yarn child localizes resources needed
→ retrieves job resources
→ runs map/reduce task
→ JVM reuse supported

2019-5-4 20:4

Progress & Status Update

→ mapper or reducer reports status to App Master over umbilical interface

→ NM → heartbeats → RM

→ RM sends client

Job completion

→ client ← HTTPClientProtocol → calls waitForCompletion()

→ AppMaster & containers clean up

→ archive as history → job info.

Yarn

→ ha fault Tolerance

→ Network Compatibility

→ supports programming paradigms other than MR (multi tenancy)

↳ runs all on same Hadoop cluster

Data serialization (ds)

process of translating DS or object state into format that can be stored and reconstructed later in same or another computer envt.

1) JSON

→ Lightweight data-interchange format

→ easy for humans to write & read

→ easy for machines to parse & generate

2019-5-4 20:45

Features → mostly human readable code
→ simple and straightforward specification
→ widespread support
→ support JS data types

2) BSON

→ Binary JSON
→ binary encoded serialization of JSON.
→ also has extensions ~~not~~ that allow representation of ~~[illegible]~~ data types not part of JSON spec

Features → convinient storage of binary info

3) Message Packs

→ like JSON but fast & small
→ binary format of SPEC

Features → designed for efficient transmission over wire
→ better JSON compatibility
→ smaller than BSON
→ type checking
→ ~~[illegible]~~ streaming API

4) YAML

→ human friendly ds std for all pgm langages

Features → truly human readable code
→ compact
→ syntax for relational data
→ suitable especially for viewing/editing of DS