Assignment - Benefit/Limitation Case of S.E

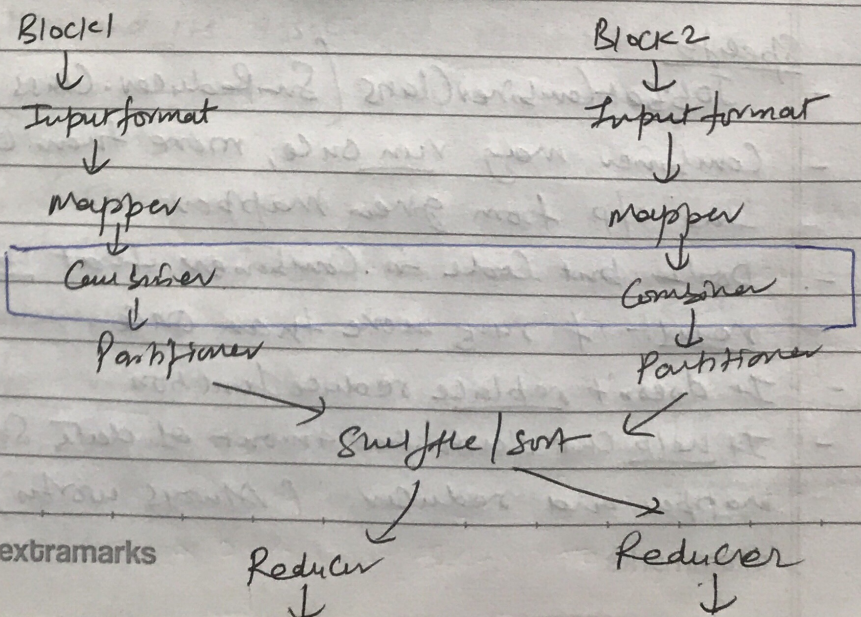
Date: Page: 15

* Task Complete - inform Job tracker.
* Whichever copy finishes first - definitive copy
* Kill other task (slow)
* Enabled by default
* - mapred.map.task.speculative.execution
  - mapred.reduce.task.speculative.execution

] Config

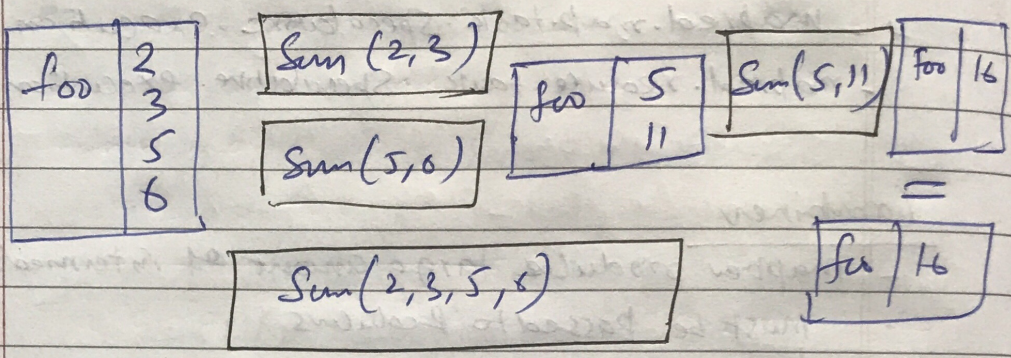
Combiner

- mapper produces large amount of intermediate data
- Must be passed to Reducers
  ↳ Can result in lot of n/w traffic
- like a mini reducer
- Run locally on single mapper's o/p
- o/p from Combiner is sent to reducers.
- Combiner - Reducer are often identical
  ↳ Possible only if operation performed is Commutative and associative
  ↳ I/p and o/p data type for C/R must be identical
- Combiner func is an optimization function

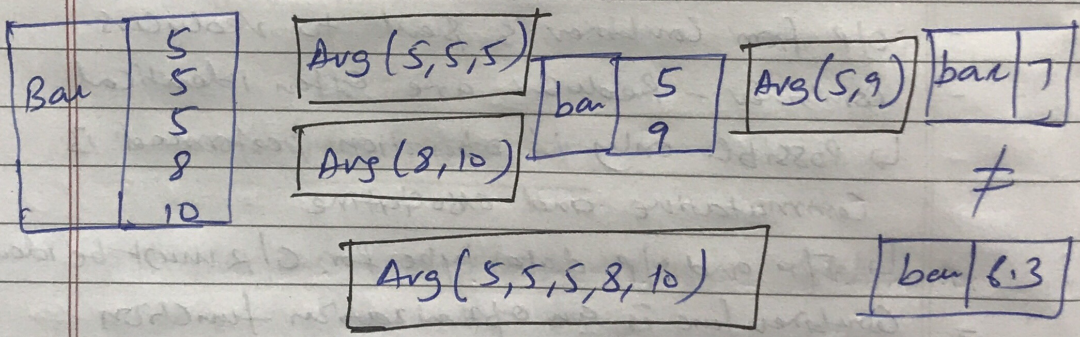


eg- in wordcount with Combiner (FIS)

→ Some Reducers may be used as Combiner if operation is associative + commutative eg SumReducer



→ Some Reducers cannot be used as a Combiner eg AverageReducer



Specify
- Job.setCombinerClass(SumReducer.class)
- Combiner may run once, more than once on the o/p from given mapper
- Don't put code in Combiner that influence result if runs more than once.
- It doesn't replace reduce function
- It help cut down the amount of data shuffle b/w mapper and reducer (always worth)

Configuration file (mapred-site.xml)

* Stored in /etc folder
* Contain Configuration info

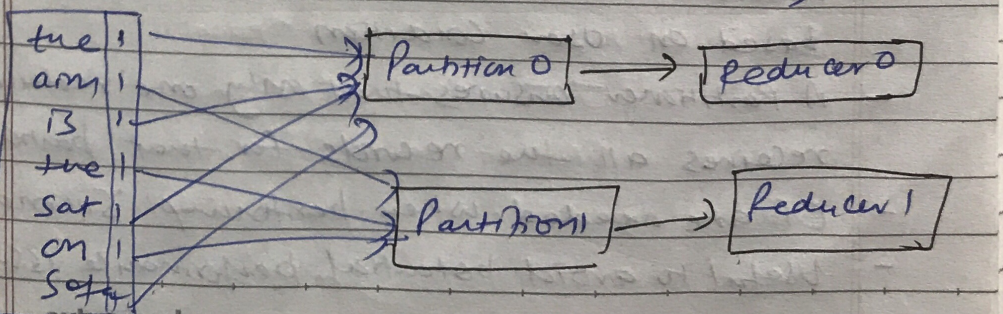
1- mapreduce.framework.name - yarn
2- mapreduce.map.memory.mb - (1024) large resource limit for mappers
3- mapreduce.reduce.memory.mb (3072) large resource limit for reducers
4- mapreduce.task.io.sort.mb (512) Higher memory limit while sorting data for efficiency
5- mapreduce.map.java.opts - (Xmx1024m) larger heap size for child jvms of map
6. mapreduce.reduce.java.opts - (Xmx2560m) larger heap size for child jvm of reduce
7. mapreduce.reduce.shuffle.parallelcopies (50) Parallel copies run by reducer.

Partitioner

- Determine which reducer each intermediate key and its associated value goes to

getPartition:

(inter_key, inter_value, num_reducers) → Partition



- Partitioning of keys of intermediate o/p is controlled by Partitioner
- Each mapper o/p is partitioned and records having same key value go into same partition
- Then each partition is sent to reducer.
- Partition phase takes place after map and before reduce
- Default Partitioner

- The default partitioner is HashPartitioner
  ↳ use Java hashcode method
  - Guarantees all pair with same key go to same Reducer

```
public class HashPartitioner<K,V> extends Partitioner<K,V>{

    public getPartition (K key, V value, int numReduceTask)
    { return (key.hashCode() & Integer.MAX_VALUE)
                % numReduceTask;
    }
```

Custom partitioner (CP)

- Process allow to store the result in different reducers based on user condition
- A Partitioner ensures that only one reducer receives all the records for that particular key
- CP are needed when performing Secondary Sort
- Useful to avoid potential performance issues -

avoid one reducer having to deal with many very large lists of values.

How to create

1) Create a class extends Partitioner
2) Override getPartition method
   → return an int b/w 0 and 1 less than no of Reducer
   eg - 10 Reducer (0-9)

3)
```
Public class MyPartitioner<K,V> extends Partitioner(K,V)
{
    Public int getPartition(K key, V value, int
                                    numReduceTask)
    {
        //determine reducer number b/w 0 - numReduce
                                              Task-1
        return reducer;
    }
}
```

4) Specify C.P in your driver code
   job.setPartitionClass(MyPartitioner.class)

Hadoop V1 components → HDFS (V1)
→ MapReduce (MRV1)

Hadoop Ecosystem
MapReduce V1
HDFS V1
Hadoop common module

Hadoop V2 components → HDFS V2
→ YARN (~~MRV2~~) MRV2
→ MapReduce (MRV1)

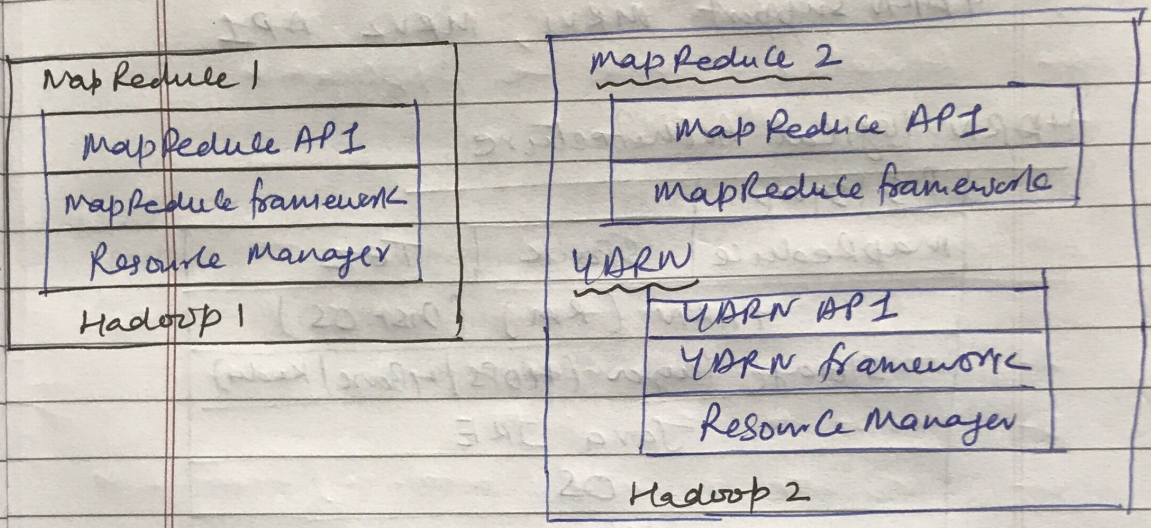
Hadoop Ecosystem	
MR (V1)	
YARN (MRV2)	
HDFS V2	
Hadoop Comm module	

Hadoop V1 limitations

1) Batch Processing
2) Not suitable for Real time Data Processing
3) Support 4000 nodes per cluster
4) More burden on JT (scheduling, monitoring, client request, Job history)
5) SPOF → NN - Cluster down
   JT - Can't exe the prog
   SNN - Can't get the latest FS image
6) Scalability - 4000 (node) (4K)
   40,000 Tasks (40K)

YARN
- 10 K Node
- 10K Jobs
- 100K task

7) fixed Slot Size - Utilization
8) Availability - rapidly changing complex state in JT memory make it diff to retrofit HA in JT Service
9) Doesn't support multi-tenancy



YARN

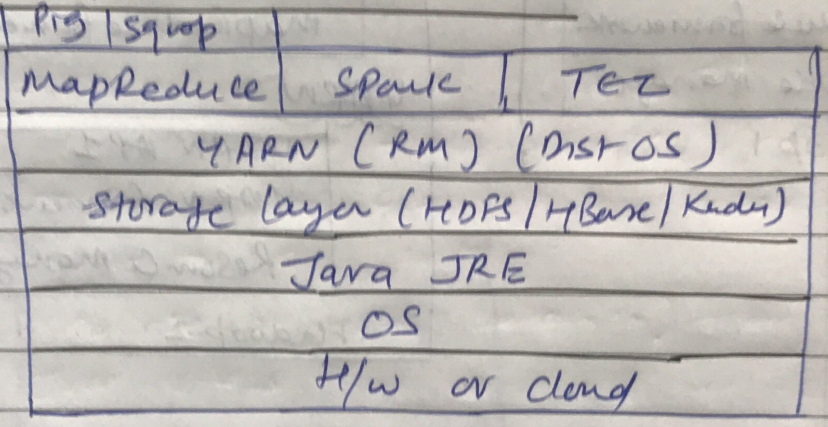
- Yet Another Resource Negotiator
- MRv_2 - Next generation MR
- Distributed Operating Sys.
- General framework to support other distributed computing Paradigm
- Yahoo 2010 began to increase performance by smaller memory utilization, enhance scalability, flexibility so to run many dist framework in parallel.
- Hide resource mgt details from user.

Main Idea

- Split Job tracker responsibility
  1) Resource Mgt (Job Scheduling)
  2) App Master (Task monitoring / States / Job Progress)

- YARN Support MRV1, MRV2 API

YARN logical Architecture

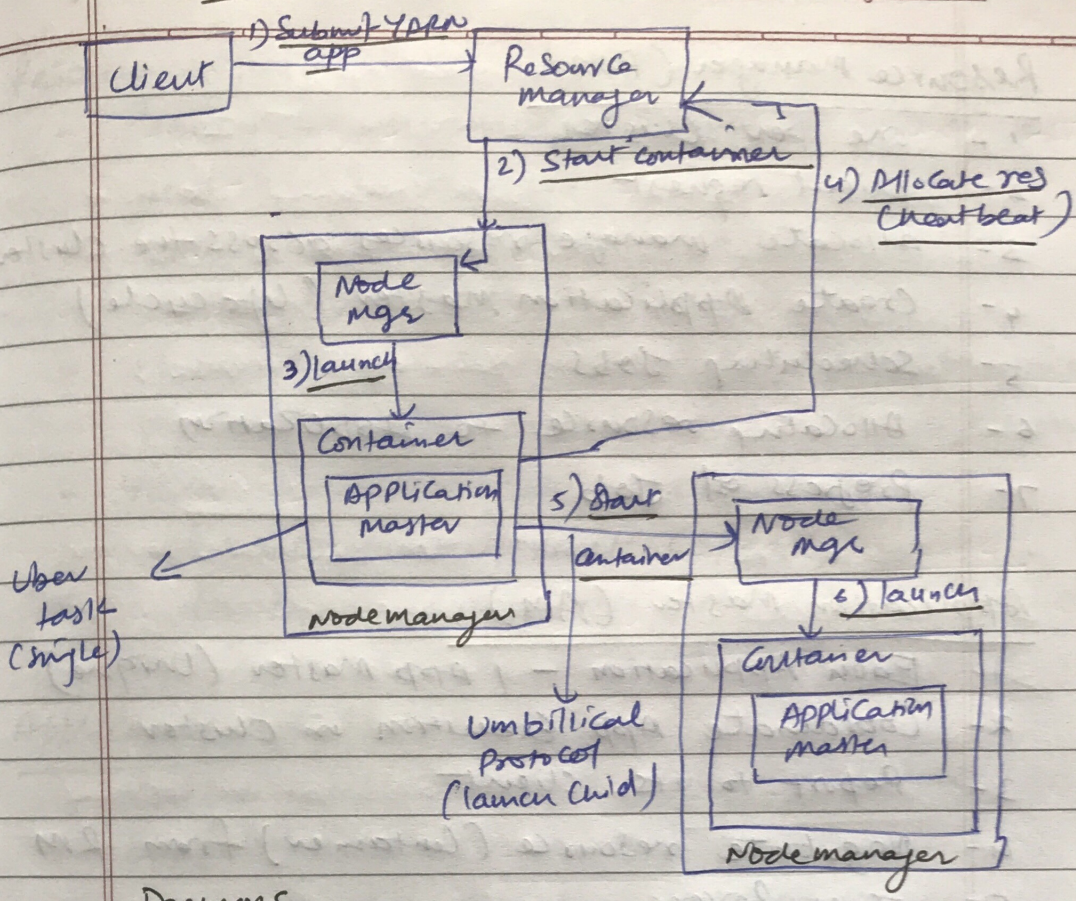


Benefits

1) High Scalability (10K nodes, 100K tasks)
2) High Availability (HA for both RM, AM)
3) Support m/t Programing model
4) Multi Tenancy (Beyond MR, even diff version of MR)
5) Multiple Name space
6) Cluster utilization (Diff slots for map and reduce)

YARN Architecture

Date: Page: 23



Daemons

1) Node Manager 2) Resource Manager

Node Manager (NM)

1- Running on all worker nodes
2- launch and monitor Container
3- Keep RM update (Heart Beat)
4- Container life cycle mgt
5- Monitor resource (CPU, memory)
6- tracking node health
7- Kill container direct by RM

Resource Manager (RM)

1- One per cluster
2- Client request
3- Allocate manage resources across the cluster
4- Create Application Master (Life cycle)
5- Scheduling Jobs
6- Allocating resource to Applications
7- Progress of Job

Application Master (AM)

1- Each Application - 1 App Master (Unique)
2- Coordinate App Execution in Cluster
3- Report to its client
4- Negotiate resource (Container) from RM
5- JVM Process
6- Run a computation in the container (Return to client)
7- Request more container from RM to run a distributed computation

Container

1- Physical resources (RAM, CPU Cores)
2- Container may be Unix process or Linux cgroup
3- Can have multiple Containers in Single node