BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DSECL ZG 522: Big Data Systems

Session 8: Hadoop MapReduce and YARN

Janardhanan PS

janardhanan.ps@wilp.bits-pilani.ac.in

Topics for today

- **Hadoop MapReduce**
  - ✓ **MapReduce runtime**
  - ✓ More examples
  - ✓ Hadoop Streaming (p23)
- Yet Another Resource Negotiator (YARN) (p25)
  - ✓ Architectural components
  - ✓ Workflow
  - ✓ Resource model and implementation
  - ✓ Resource scheduling
  - ✓ YARN sample commands

MapReduce Run Time Architecture - Overview

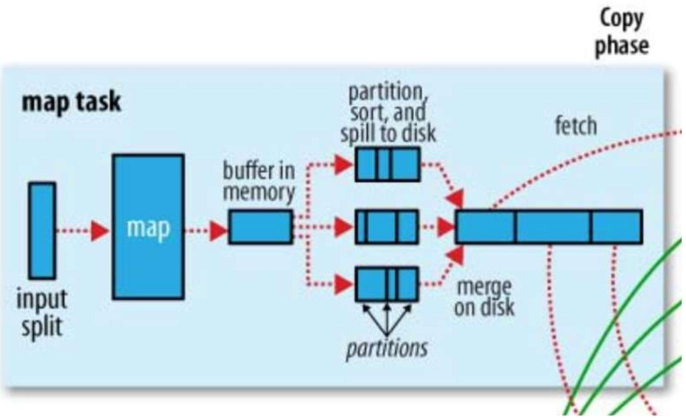
- Input dataset is split into multiple pieces of data (several small sets)
- Framework creates a master (application master - AM) and several worker processes and executes the worker processes remotely
- Several Map tasks work simultaneously and read pieces of data that were assigned to each map.
- Map worker uses the map function to process only those data that are present on their server and generates key/value pair for the extracted data.
- Map worker uses partitioner function to divide the data into regions. **Partitioner** decides which reducer should get the output of specified mapper.
- When the map workers complete their work, the master (AM) instructs the reduce workers to begin their work.
- The reduce workers in turn contact the map workers to get the key/value data for their partition (shuffle). The data thus received from various mappers is merge sorted as per keys.
- Then it calls reduce function on every unique key. This function writes output to the file.
- When all the reduce workers complete their work, the master(AM) transfers the control to the user program.

MapReduce Design Patterns

- **Summarization patterns:** get a top-level view by summarizing and grouping data
- **Filtering patterns:** view data subsets such as records generated from one user
- **Distinct patterns**: Eliminates duplicates in the data
- **Data organization patterns:** reorganize data to work with other systems, or to make MapReduce analysis easier
- **Join patterns:** analyze different datasets together to discover interesting relationships
- **Metapatterns:** piece together several patterns to solve multi-stage problems, or to perform several analytics in the same job
- **Input and output patterns:** customize the way you use Hadoop to load or store data

- Refer: *MapReduce Design Patterns* - Donald Miner & Adam Shook , O'Reilly

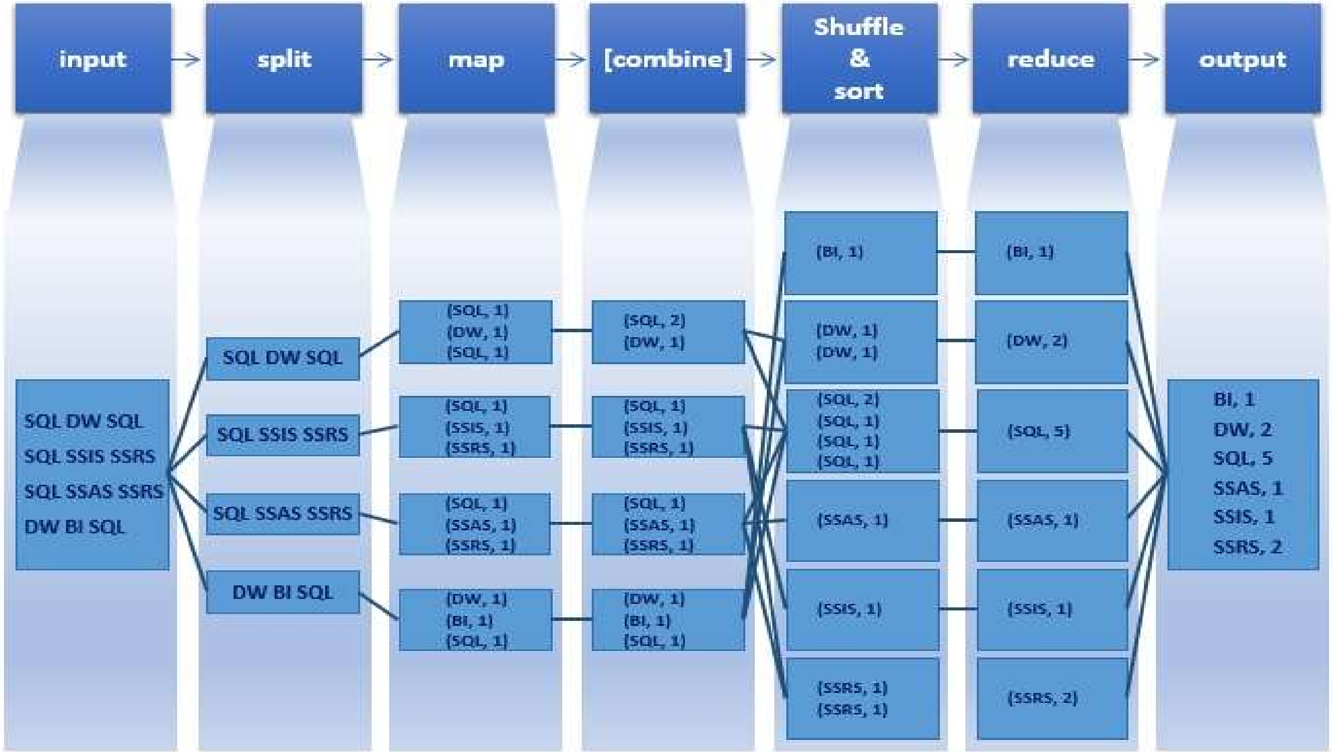
Map



1. Record Reader
   - ✓ Reads each record created by input splitter to pass key-value pair into Map
   - ✓ Can read text, binary etc.
   - ✓ Examples: LineRecordReader, SequenceFileRecordReader
2. Map
   - ✓ User defined function to create key, value pair from input key, value pair
3. Combiner
   - ✓ Localized optional reducer for better performance - can be same as reducer code (discussed later)
   - ✓ e.g. <hello,1> x 3 pairs on same node can be <hello, 3>
4. Partitioner
   - ✓ Takes intermediate output from map and shards it to send to different reducers, i.e. determines which reducer should get a shard (some sorting happens based on partition logic)
   - ✓ Default partitioner: **key.hashcode()%num_reducers** spreads keyspace evenly among reducers
   - ✓ Ensures same key is sent to same reducer
   - ✓ Shards are written to disk waiting for reducer to pull
   - ✓ Custom partitioner class can be implemented (see T1, Pg 226 example, also discussed later)

MapReduce Data Flow

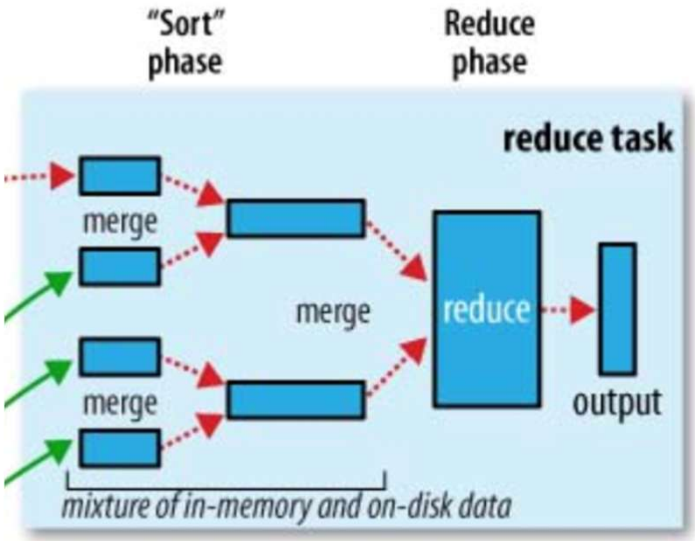
MapReduce – Word Count Example Flow



https://stackoverflow.com/questions/22141631/what-is-the-purpose-of-shuffling-and-sorting-phase-in-the-reducer-in-map-reduce

Reduce

1. Shuffle and Sort
   - ✓ Shuffle gets data from map node to reduce node or reduce task to happen where the relevant post-map data partition is
   - ✓ Shuffling can start before Map is entirely complete
   - ✓ Data is merge-sorted by key coming in from multiple maps
2. Reduce
   - ✓ Call user defined function per key grouping, e.g. <India,{1,1,1,1}>
   - ✓ A reduce call looks at a unique key but global state can be maintained in Reduce task in class variable
   - ✓ Can control number of reducers, e.g. one reducer if a sequential computation has to calculate one final value.
3. Output Format
   - ✓ Writes final output of reducer with separator of key, value and separator between pairs



MR job execution

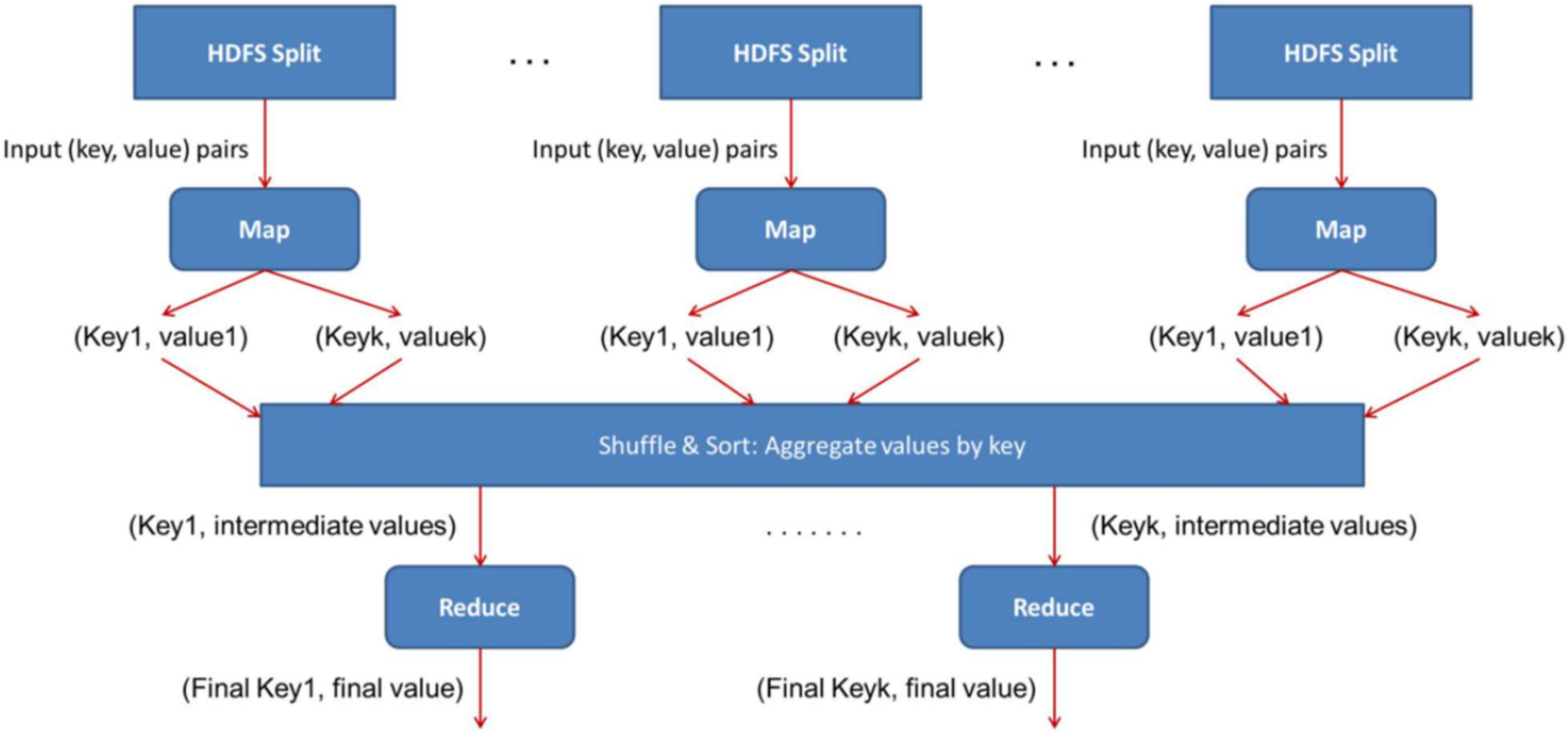


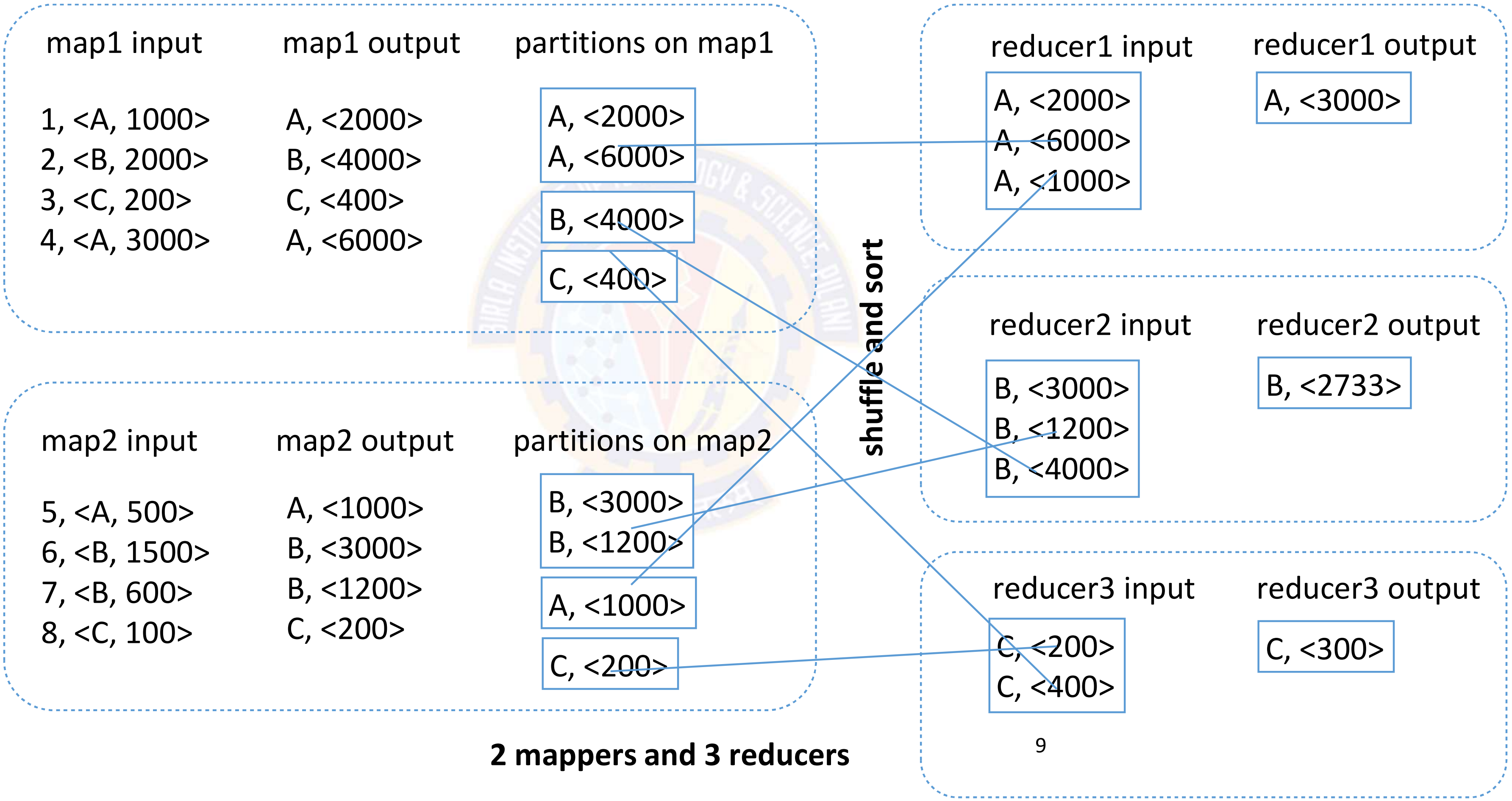
Image ref: https://data-flair.training/blogs/hadoop-architecture/

Map Reduce Example (input <k, v> find avg(2v) for each k)

input
file

A, 1000
B, 2000
C, 200
A, 3000

A, 500
B, 1500
B, 600
C, 100



map1 input

1, <A, 1000>
2, <B, 2000>
3, <C, 200>
4, <A, 3000>

map1 output

A, <2000>
B, <4000>
C, <400>
A, <6000>

partitions on map1

A, <2000>
A, <6000>

B, <4000>

C, <400>

map2 input

5, <A, 500>
6, <B, 1500>
7, <B, 600>
8, <C, 100>

map2 output

A, <1000>
B, <3000>
B, <1200>
C, <200>

partitions on map2

B, <3000>
B, <1200>

A, <1000>

C, <200>

shuffle and sort

reducer1 input

A, <2000>
A, <6000>
A, <1000>

reducer1 output

A, <3000>

reducer2 input

B, <3000>
B, <1200>
B, <4000>

reducer2 output

B, <2733>

reducer3 input

C, <200>
C, <400>

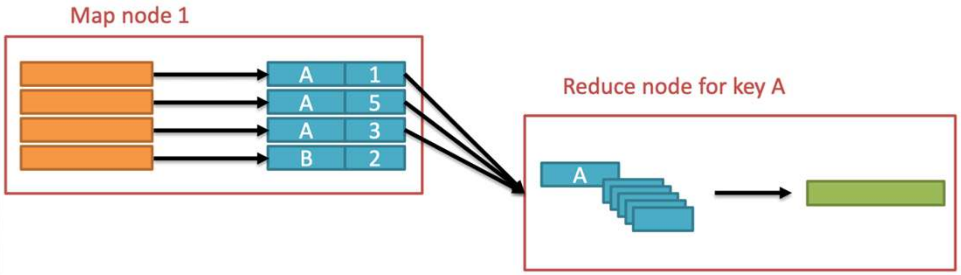
reducer3 output

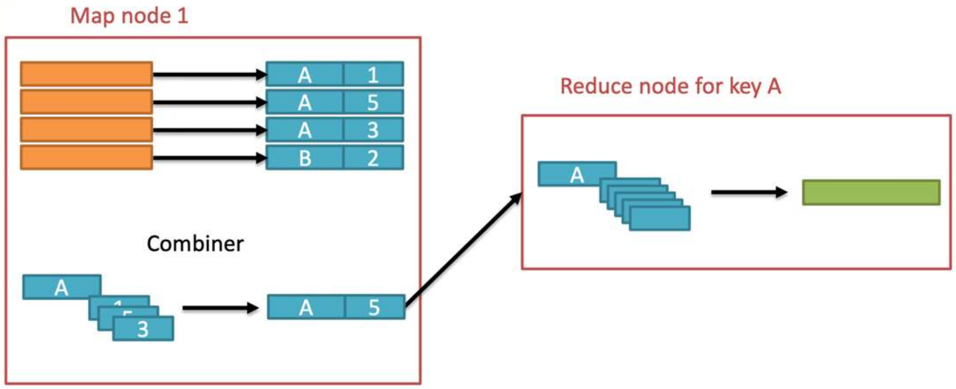
C, <300>

2 mappers and 3 reducers

A word about Combiner

- Combiners can optimize the reduce phase by pre-processing on each node to compress data but output has to be same type as a map
- The reducer can also be set as the combiner class only if reduce is Commutative and Associative (C&A) operation
  - ✓ max(max(1,2), max(3,4,5)) = max(max(2,4), max(1,5,3)) or any other order
  - X avg(avg(1,2),avg(3,4,5)) = 2.75… but avg(avg(2,4),avg(1,5,3)) = 3
- If not C&A then optionally write a new combiner logic





Performance optimizations

- We studied types of parallelism earlier
- The goal for parallel computation is balanced and maximized utilization of the cluster from CPU and memory standpoint
- So carefully look at
  - ✓ Partitioner - will a custom distribution of keys help and not use the key hash code default
  - ✓ Combiner - will the reducer or a custom combiner help for compression of data to reducer

Topics for today

- Hadoop MapReduce
  - ✓ MapReduce runtime
  - ✓ **More examples**
  - ✓ Hadoop Streaming
- Yet Another Resource Negotiator (YARN)
  - ✓ Architectural components
  - ✓ Workflow
  - ✓ Resource model and implementation
  - ✓ Resource scheduling
  - ✓ YARN sample commands

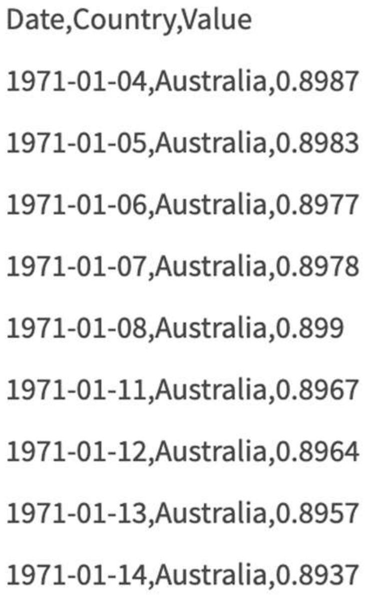
Example 1: Sorting by value

- Sort all employees by their salary given input file with records <name, salary>
- MapReduce automatically does sorting on keys given <key, value> pairs produced by Map.
- But we need to sort on values (salary) and not keys (name).
- So swap the key and values in map()
- Can set comparator class if value sorting is not done properly with default comparator

- Map logic
  - ✓ <key=k, value=v> -> <key=v, value=k>
- Reduce logic
  - ✓ write <k,v> - no extra logic
- In Driver
  - ✓ Set job.NumReduceTasks(1)
  - ✓ Set job.setComparatorClass(intComparator.class)

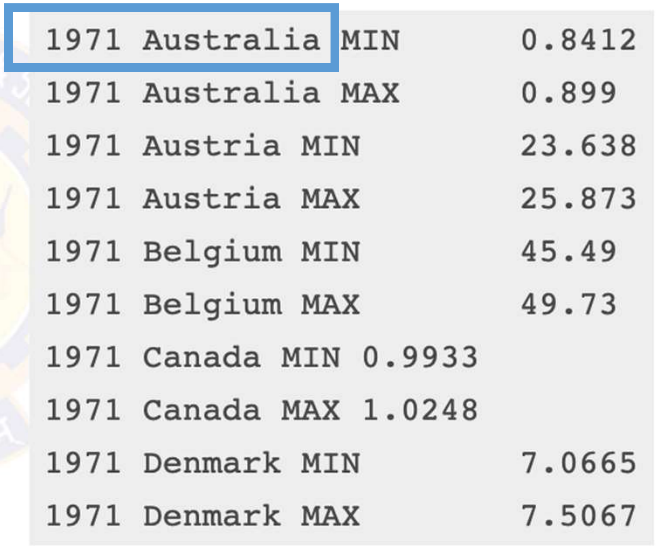
Example 2: Find max / min in each group

Find max / min FX rate every year for each country



```
Date,Country,Value
1971-01-04,Australia,0.8987
1971-01-05,Australia,0.8983
1971-01-06,Australia,0.8977
1971-01-07,Australia,0.8978
1971-01-08,Australia,0.899
1971-01-11,Australia,0.8967
1971-01-12,Australia,0.8964
1971-01-13,Australia,0.8957
1971-01-14,Australia,0.8937
```

Key is year and country combination



```
1971 Australia MIN       0.8412
1971 Australia MAX       0.899
1971 Austria MIN         23.638
1971 Austria MAX         25.873
1971 Belgium MIN         45.49
1971 Belgium MAX         49.73
1971 Canada MIN 0.9933
1971 Canada MAX 1.0248
1971 Denmark MIN         7.0665
1971 Denmark MAX         7.5067
```

Example 2 ...

Creates composite key of year + country in map for default hash-based partitioning

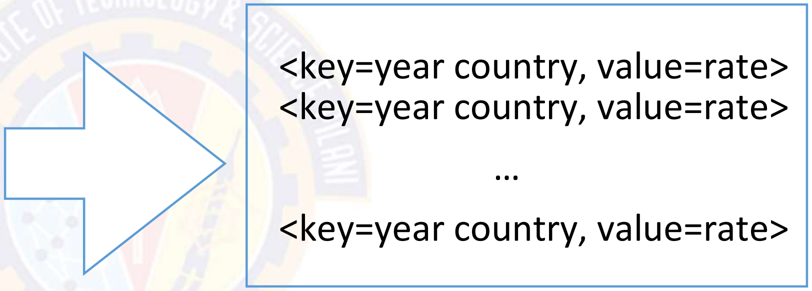
```
public void map(Object key, Text value, Context context
) throws IOException, InterruptedException {

try {
 //Split columns
 String[] columns = value.toString().split(comma);

 if ( columns.length<3 || columns[2] == null
         || columns[2].equals("Value")) {
      return;
   }
   //Set FX rate
   rate.set(Double.parseDouble(columns[2]));

   //Construct key: e.g. 1971 Australia
   YearCountry.set(columns[0].substring(0, 4) +" "+ columns[1]);

   //Submit value into the Context
   context.write(YearCountry, rate);
}
catch (NumberFormatException ex) {
   context.write(new Text("ERROR"), new DoubleWritable(0.0d));
}
}
```



<key=year country, value=rate>
<key=year country, value=rate>
...
<key=year country, value=rate>

Each map produces a mix of keys from input data

can you use a combiner here ? same logic as reducer ?

Example 2 …

Creates composite key of year + country in map for default hash-based partitioning

after shuffle-sort using key

<key=K1, value=rate>
<key=K1, value=rate>

…

<key=K1, value=rate>

<key=K2, value=rate>
<key=K2, value=rate>

…

<key=K2, value=rate>

…

#partitions = #keys = #reduce calls

example input to reduce() for a key:
<key=K1, value=rate1, rate2, …>

each reduce() call is for a key and value list in partition

```
public void reduce(Text key, Iterable<DoubleWritable>
 values, Context context) throws IOException,
 InterruptedException {

double min = 9999999999999999d;
double max = 0;

for (DoubleWritable val : values) {
  min = val.get()<min? val.get():min;
  max = val.get()>max? val.get():max;
}
minW.set(min);
maxW.set(max);

//Set key as Year Country Min/Max

Text minKey = new Text(key.toString()+" "+"MIN");
Text maxKey = new Text(key.toString()+" "+"MAX");

context.write(minKey, minW);
```

set #reducers = 1
if all results needed i
single node else collect
later and let partitions
run in parallel

can you use a combiner here ? same logic as reducer ?

Example 3: Custom partitioner instead of composite-key

- Map:
  - ✓ create list of <gender, {age, salary}>

<**male**, {45, 97000}>, <**female**, {29, 80000}>, …

- Partitioner:
  - ✓ Return partition number as a function of age - create 3 age groups
  - ✓ So, partition is not a hash of gender but age group - i.e. not 2 partitions only but 3 partitions created
- Reduce:
  - ✓ 3 age groups means 3 reducers
  - ✓ A specific age partition across all genders goes to a specific reducer
    - So, partition i goes to reducer i
    - In Driver set #reducers = 3
  - ✓ For each reducer, a call to reduce() is made for each gender because records are still <gender, {age, salary}>
    - So, 6 reduce() calls across 3 reducers and 2 values for gender key
  - ✓ Each reducer reduce() call finds max() of values in <gender, {salary list}>
  - ✓ So, each reducer finds max for each gender within the age group allocated to the reducer

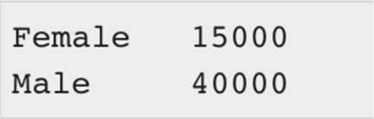
grp1: <female, {29, 80000}>, <male, …>, …
grp2: <male, {45, 97000}>, <female, …>, …
grp3: …

Example 3: Custom partitioning

Find max salary by gender and by age group

1201	gopal	45	Male	50,000	3 – Age 40-49
1202	manisha	40	Female	51,000	
1203	Priya	34	Female	35,000	2 – Age 30-39
1204	prasanth	30	Male	31,000	
1205	kiran	20	Male	40,000	1 – Age 20-29
1206	laxmi	25	Female	15,000	

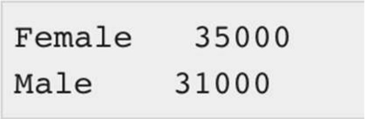
Output in Part-00000



```
Female    15000
Male      40000
```

age group 1

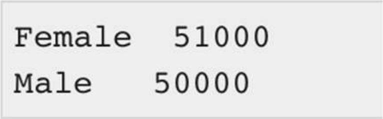
Output in Part-00001



```
Female    35000
Male     31000
```

age group 2

Output in Part-00002



```
Female   51000
Male    50000
```

age group 3

https://www.tutorialspoint.com/map_reduce/map_reduce_partitioner.htm

Example 4

Given a document with several sentences. Each word has a weight calculated based on letters in the word. A letter's weight is based on ASCII code. Find document weight.

Approach 1

- Map
  - ✓ input: a part of a document (write a splitter dividing document in chunks every N lines)
  - ✓ logic: convert into words, calculate word weight and segment weight
  - ✓ output: <segment id, weight>
- Reduce (set reducer count to 1)
  - ✓ input: <segment id>, <weight>
  - ✓ logic: keep a class variable to add up total weight across segments
  - ✓ output: <_, total weight>

Approach 2

- Map
  - ✓ input: a part of a document (write a splitter dividing document in chunks every N lines)
  - ✓ logic: convert into words, calculate word weight and emit word weights
  - ✓ output: <word, weight>
- Combine
  - ✓ input: local pairs <word, weight>
  - ✓ output: <segment id, weight>
- Reduce 2: same as reduce 1

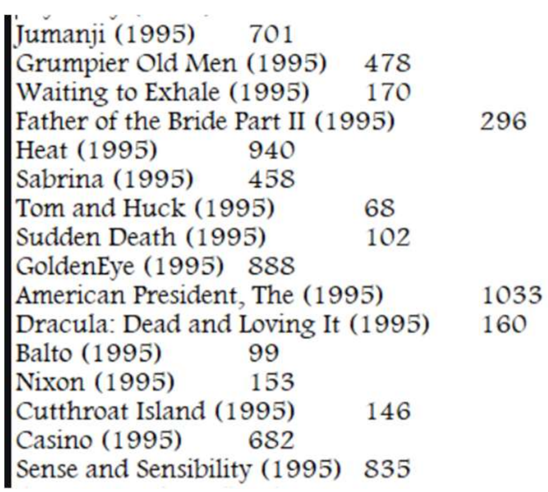
Approach 3

- Map
  - ✓ input: a part of a document (write a splitter dividing document in chunks every N lines)
  - ✓ logic: convert into words, calculate word weight and emit word weights
  - ✓ output: <word, weight>
- Combine (Same as reducer)
  - ✓ input: local pairs <word, weight>
  - ✓ output: <word, total weight across instances>
- Reduce: (set reducer could to 1)
  - ✓ calculate total weight across all words using a variable in reducer class

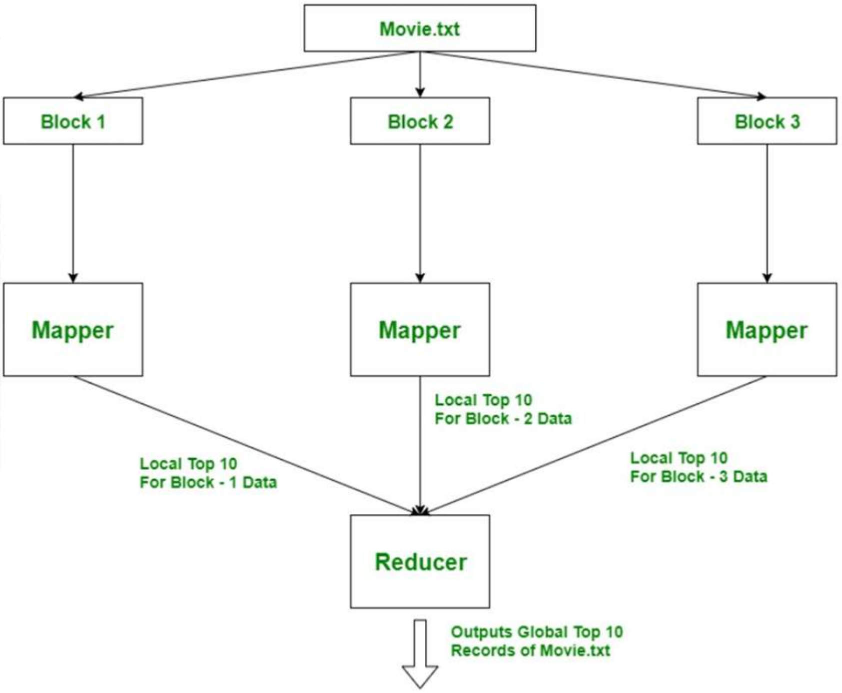
Example 5: Top N keys given key-value pairs

Find the Top 10 movies given <movie_name,#views>

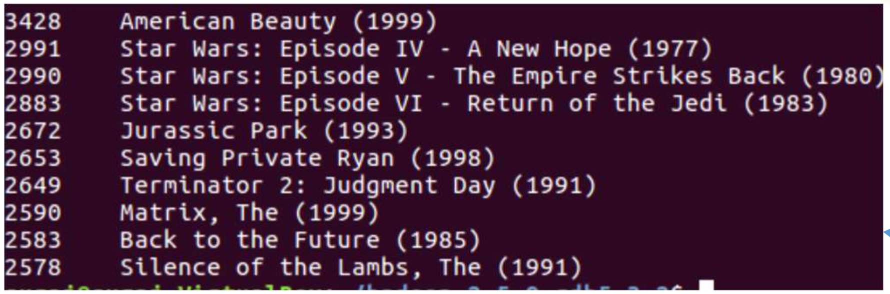
movie_name and no_of_views



```
Jumanji (1995)	701
Grumpier Old Men (1995)	478
Waiting to Exhale (1995)	170
Father of the Bride Part II (1995)	296
Heat (1995)	940
Sabrina (1995)	458
Tom and Huck (1995)	68
Sudden Death (1995)	102
GoldenEye (1995)	888
American President, The (1995)	1033
Dracula: Dead and Loving It (1995)	160
Balto (1995)	99
Nixon (1995)	153
Cutthroat Island (1995)	146
Casino (1995)	682
Sense and Sensibility (1995)	835
```



Movie.txt
Block 1
Block 2
Block 3
Mapper
Mapper
Mapper
Local Top 10 For Block - 1 Data
Local Top 10 For Block - 2 Data
Local Top 10 For Block - 3 Data
Reducer
Outputs Global Top 10 Records of Movie.txt



```
3428	American Beauty (1999)
2991	Star Wars: Episode IV - A New Hope (1977)
2990	Star Wars: Episode V - The Empire Strikes Back (1980)
2883	Star Wars: Episode VI - Return of the Jedi (1983)
2672	Jurassic Park (1993)
2653	Saving Private Ryan (1998)
2649	Terminator 2: Judgment Day (1991)
2590	Matrix, The (1999)
2583	Back to the Future (1985)
2578	Silence of the Lambs, The (1991)
```

Reference link

Example 5: Top N keys given key-value pairs

Find the Top N movies given <movie_name,#views>

```
Mapper class {
 TreeMap tmap

 map( key, value ) {
  get movie and views from value
  tmap.put(views, movie);
  if (tmap.size() > N)
    tmap.remove(tmap.firstKey())

 for each Map :
   get all tmap entries <key, value>
   context.write(key, value)
}
```

Keep a local sorting data structure like TreeMap or similar to locally sort top N in each map

M11, C11
M23, C23
…

M101, C101
M203, C203
…

…

locally sorted top N lists

```
Reducer class {
 TreeMap tmap

 reduce( key, value ) {
   tmap.put(value, key);
   if (tmap.size() > N)
     tmap.remove(tmap.firstKey());

 for each Reduce :
   get all tmap entries <key, value>
   context.write(key, value)
}
```

Set reducer count = 1 in Driver
Maintain a TreeMap or any other sort data structure to keep a global Top N

More efficient than passing all keys to reducer as in Example 1 on sorting.
We only want top N, so filter top N at map level.

Advanced examples

- Secondary sorting
  - ✓ http://www.javamakeuse.com/2016/04/secondary-sort-example-in-hadoop-mapreduce.html
- Iterative MR - k-means
  - ✓ https://github.com/thomasjungblut/mapreduce-kmeans/tree/master/src/de/jungblut/clustering/mapreduce

Hadoop streaming

- Enables to run any executable as map reduce tasks
- Creates map / reduce tasks from the submitted code and monitor progress till completion
- One can override defaults of input / output formats, partitioners, combiners etc. with own implementations
- One can also use an aggregator package as a special reducer that supports max, min, count etc.
- Can be used to run python code or any other executable

Hadoop Streaming (apache.org)

Hadoop streaming - Example

- Hadoop Streaming allows developers to use various languages for writing MapReduce programs
- It supports all the languages that can read from standard input and write to standard output
- Python
- C++
- Ruby, etc

1. Manual testing of streams

```
cat sample.txt | python mapper.py | sort -k1,1 | python reducer.py
```

2. Running the program with hadoop streaming

```
 hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.5.jar -input /sample.txt -output /myoutput -mapper "python
./mapper.py" -reducer "python ./reducer.py" -file ./mapper.py ./reducer.py
```

Mapper.py

Loop over the words array and print the word with the count of 1 to the STDOUT

```
words = line.split()
for word in words:
        print('%s\t%s' % (word, 1))
```

Reducer.py

Reads from STDIN and outputs unique words with count to STDOUT

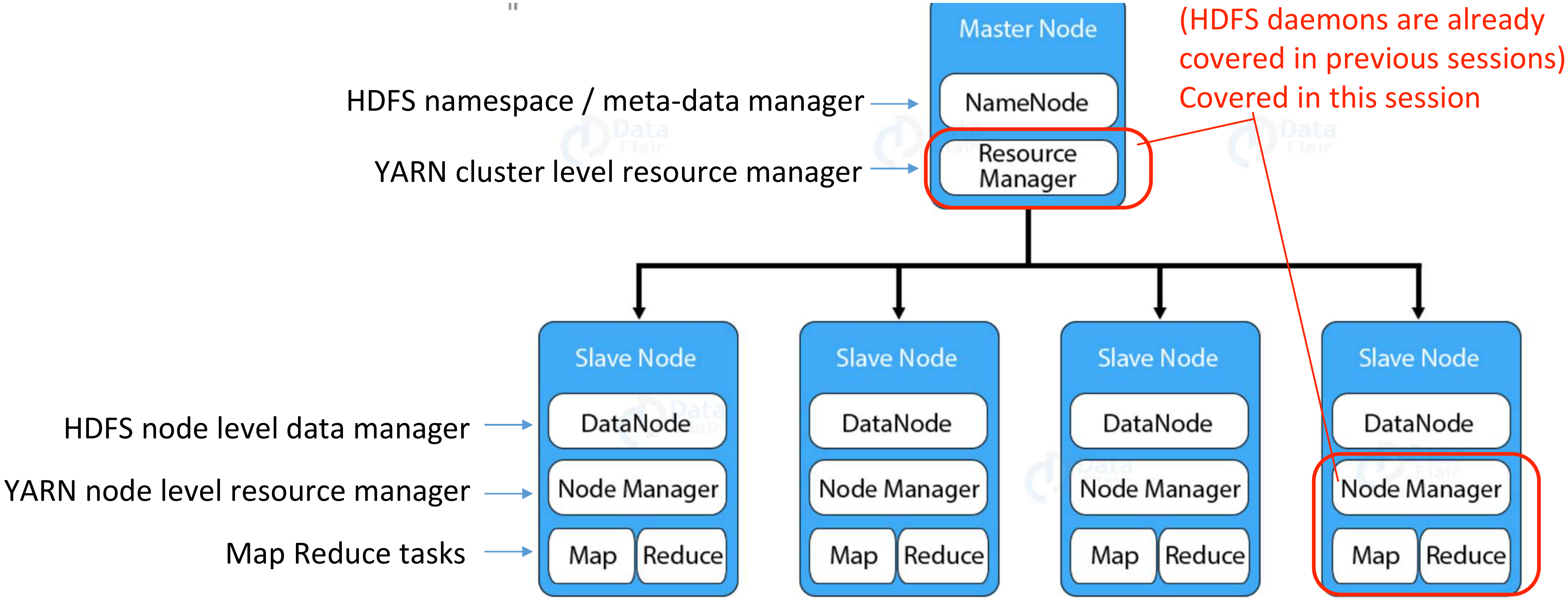
https://www.geeksforgeeks.org/hadoop-streaming-using-python-word-count-problem/

Topics for today

- Hadoop MapReduce
  - ✓ MapReduce runtime
  - ✓ More examples
  - ✓ Hadoop streaming
- **Yet Another Resource Negotiator (YARN)**
  - ✓ Architectural components
  - ✓ Workflow
  - ✓ Resource model and implementation
  - ✓ Resource scheduling
  - ✓ YARN sample commands

Hadoop 2 - Architecture

- Master-slave architecture for overall compute and data management
- Slaves (workers) implement peer-to-peer communication



HDFS namespace / meta-data manager

YARN cluster level resource manager

(HDFS daemons are already covered in previous sessions)

Covered in this session

HDFS node level data manager

YARN node level resource manager

Map Reduce tasks

***Note**: YARN Resource Manager also uses Application level App Master (AM) processes on slave nodes for application specific resource management*

YARN

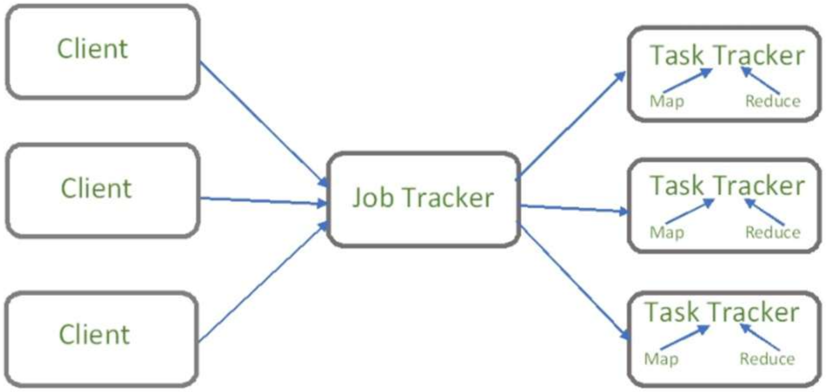
Yet Another resource Negotiator (YARN) was introduced in Hadoop 2.0 to overcome scalability bottleneck of Hadoop 1.0

https://www.cse.ust.hk/~weiwa/teaching/Fall15-COMP6611B/reading_list/YARN.pdf

- YARN makes Hadoop a general-purpose distributed computing framework
- It partitions system resources into **containers** and launches tasks in them
- Number of concurrently running tasks depends on the fraction of system resources allocated to the containers
- Large volume data is partitioned into chunks of equal sizes and tasks are assigned to each chunk
- YARN allows multi-tenant applications to run on the same cluster

Changes from Hadoop 1 - Why YARN ?

- Hadoop 1: MapReduce also did resource management with JobTracker on Master and TaskTrackers on slaves
- Hadoop 2: The resource management was decoupled from MapReduce data processing and the daemons were refactored to have a “redesigned Resource Manager”.
- Result
  - ✓ YARN - Yet Another Resource Negotiator
- YARN enables multiple applications to share same cluster and not require separate clusters for performance isolation
- Hadoop2 is backward compatible with Hadoop1 - can run Hadoop 1 jobs on Hadoop2

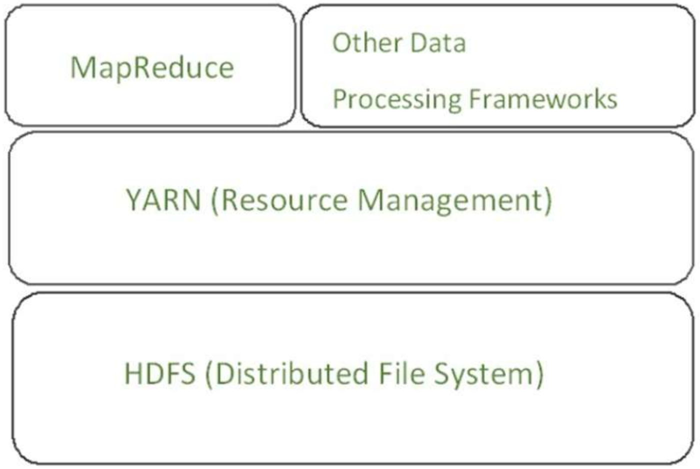


Hadoop 1.0 architecture

https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

Where does YARN fit in Hadoop ?

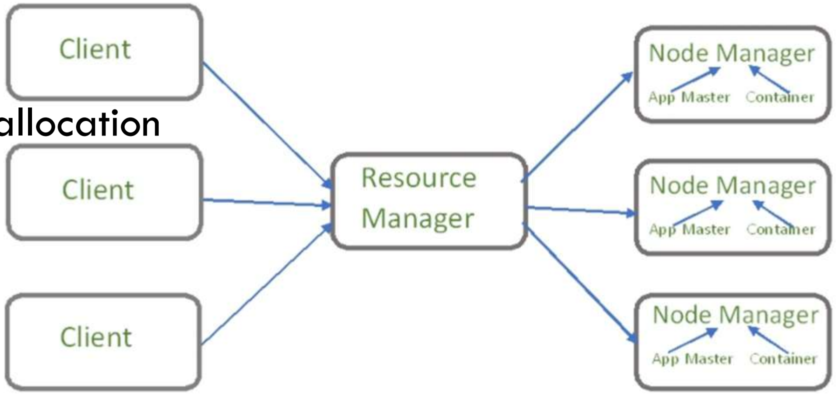
- YARN enables MapReduce and other data processing logic to run on the same Hadoop cluster using HDFS or other file systems.
- So now Hadoop can be used by other engines that are not batch processing
  - ✓ Graph, stream, interactive …
- YARN provides
  - ✓ Scalability across 1000s of nodes - environments tested with 10K+ nodes
  - ✓ Compatibility with MR and other engines
  - ✓ Dynamic cluster utilization
  - ✓ Multi-tenancy across various processing engines


MapReduce
Other Data Processing Frameworks
YARN (Resource Management)
HDFS (Distributed File System)

Hadoop 2.0

YARN components (1)

1. Client
   - ✓ Submits MR jobs
2. Resource manager
   - ✓ Key role is to schedule resources in the cluster
   - ✓ Takes request from client and talks to Node Managers for allocation
   - ✓ 2 components:
     - <u>App Manager: Master component of App management</u>
       - Accepts job request and sets up an AppMaster inside a container for each job on a worker.
       - Restarts the AppMaster on failure.
       - Main App specific work is done by AppMaster.
   - Scheduler : key function

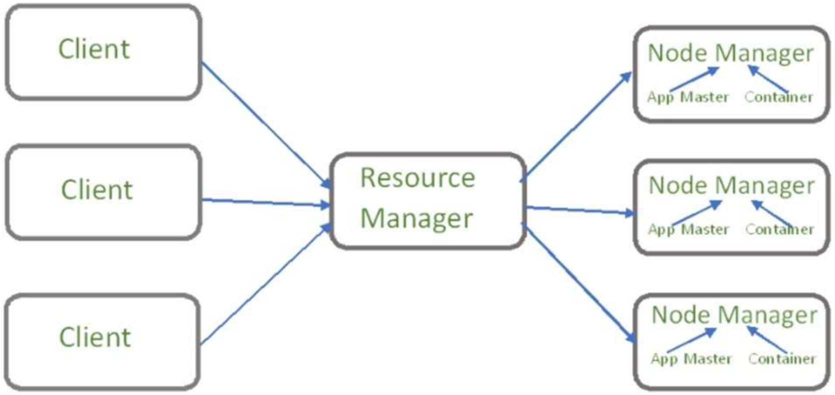


Hadoop Yarn architecture

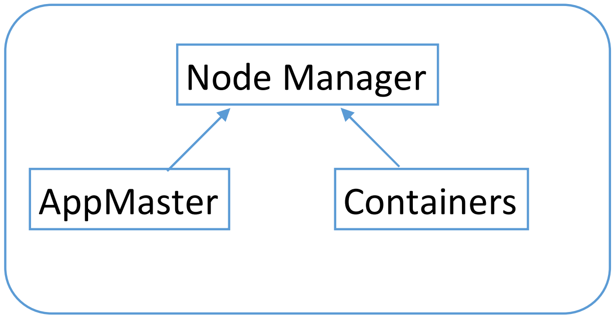
3. Node Manager
   - ✓ Takes care of management and workflows on a node.
   - ✓ Creating, killing containers, monitoring usage, log management

YARN components (2) - AppMaster

4. AppMaster
   - ✓ Negotiates resources from Resource Manager per application for starting containers on nodes
   - ✓ Sends periodic health status of application containers and tracks progress
   - ✓ Talks via Node Manager for updates and usage reports to Resource Manager
   - ✓ Clients can directly talk to AppMaster
5. Container
   - ✓ CPU, Memory, Storage resources on a node
   - ✓ Container Launch Context (CLC) - data structure that contains resource, security tokens, dependencies, environment vars



Hadoop Yarn architecture



What are Containers ?

- YARN arbitrates system resources in units of containers.
- Containers are customizable collection of resources like CPU and memory.
- Containers are basic schedulable units of execution for running tasks
- Usually, containers manifest in the form of JVMs
- The size of resources allocated to each container determines the granularity of the division of available system resources.
- Parallelism in execution of tasks in a Hadoop cluster is controlled by granularity in the division of resource allocation for containers.
- More number of containers increases resource management overheads

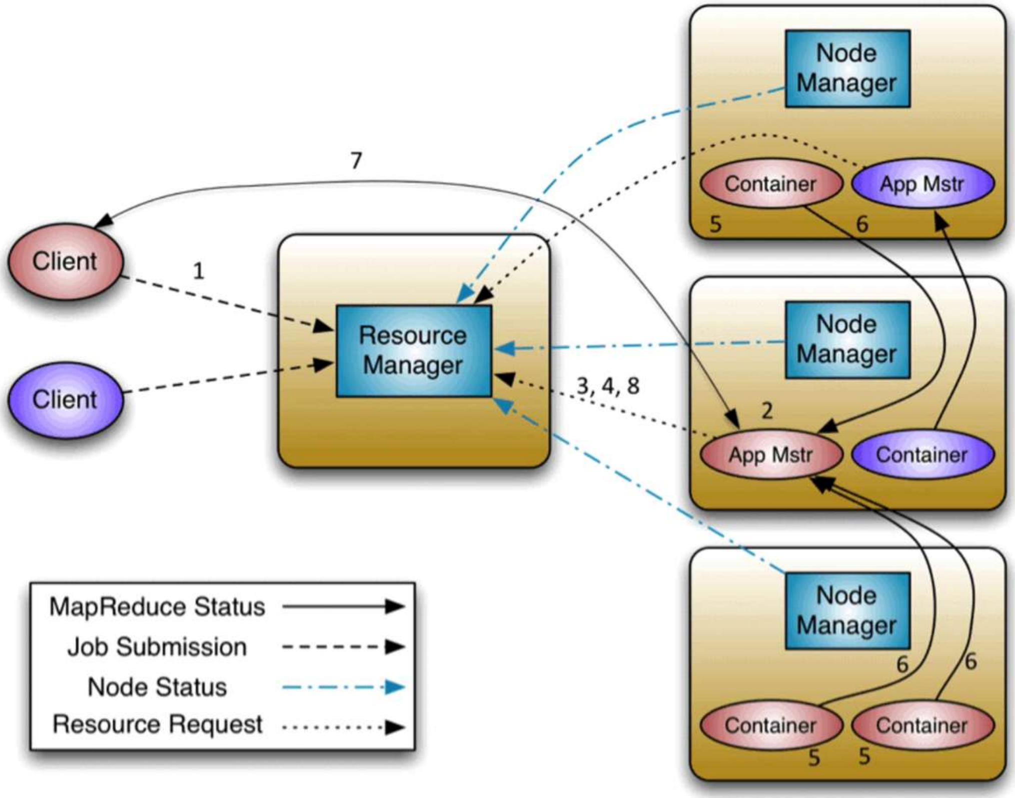
Analysis and Modeling of Resource Management Overhead in Hadoop YARN Clusters
https://ieeexplore.ieee.org/document/8328503

Topics for today

- Hadoop MapReduce
  - ✓MapReduce runtime
  - ✓ More examples
  - ✓ Hadoop streaming
- Yet Another Resource Negotiator (YARN)
  - ✓ Architectural components
  - ✓ **Workflow**
  - ✓ Resource model and implementation
  - ✓ Resource scheduling
  - ✓ YARN sample commands

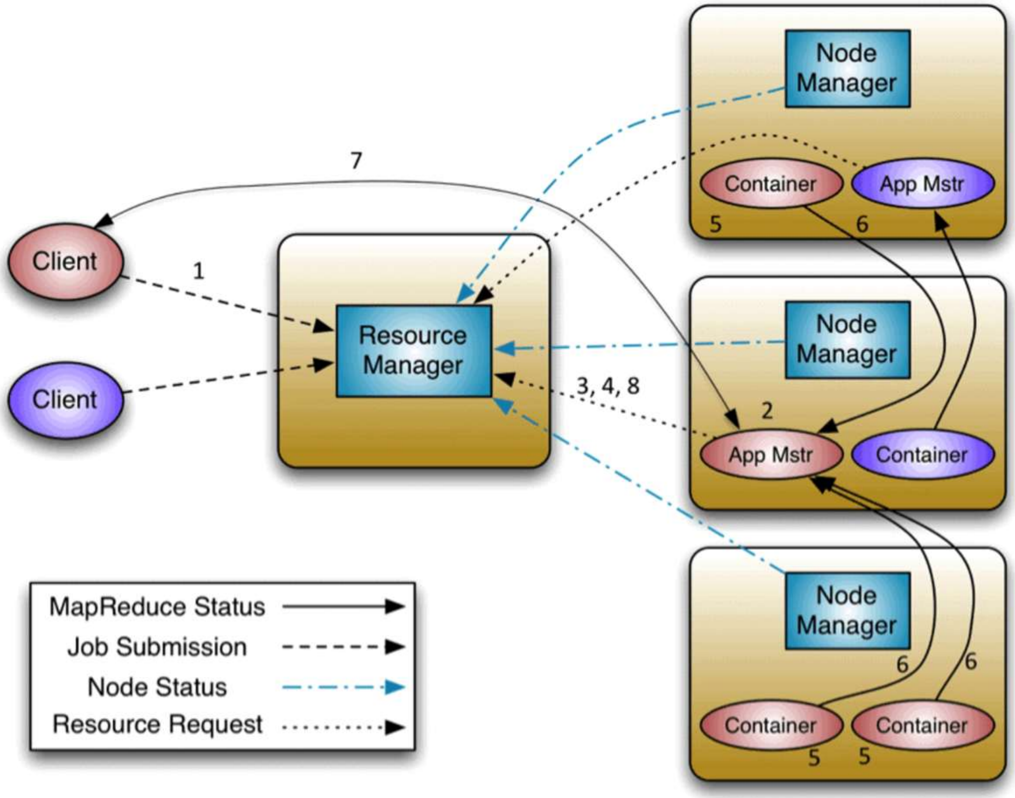
YARN workflow (1)

1. A client program *submits* the application / job with specs to start AppMaster
2. The ResourceManager asks a NodeManager to start a container which can host the ApplicationMaster and then launches ApplicationMaster.
3. The ApplicationMaster on start-up registers with ResourceManager. So now the client can contact the ApplicationMaster directly also for application specific details.



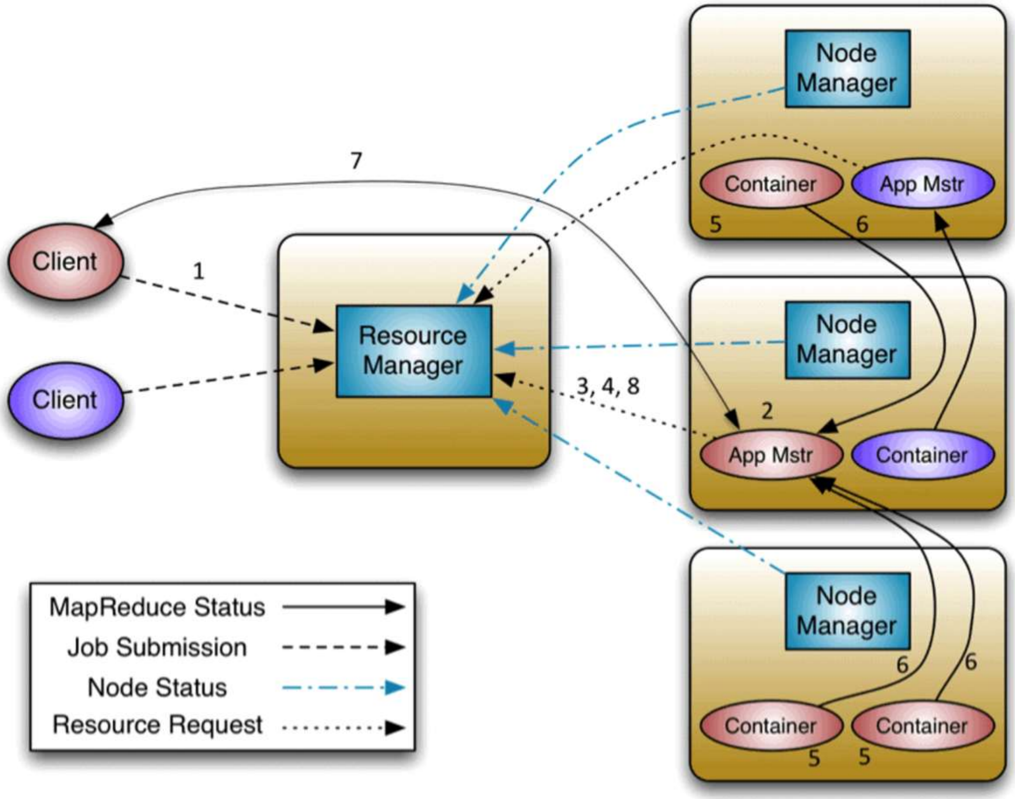
YARN workflow (2)

4. As the application executes, the AppMaster negotiates resources in the form of containers via the resource request protocol involving the ResourceManager.
5. As a container is allocated successfully for an application, the AppMaster works with the NodeManager on same or diff node to launch the container as per the container spec. The spec involves how the AppMaster can communicate with the container.
6. The app specific code inside container provides runtime information to the AppMaster for progress, status etc. via application-specific protocol.



YARN workflow (3)

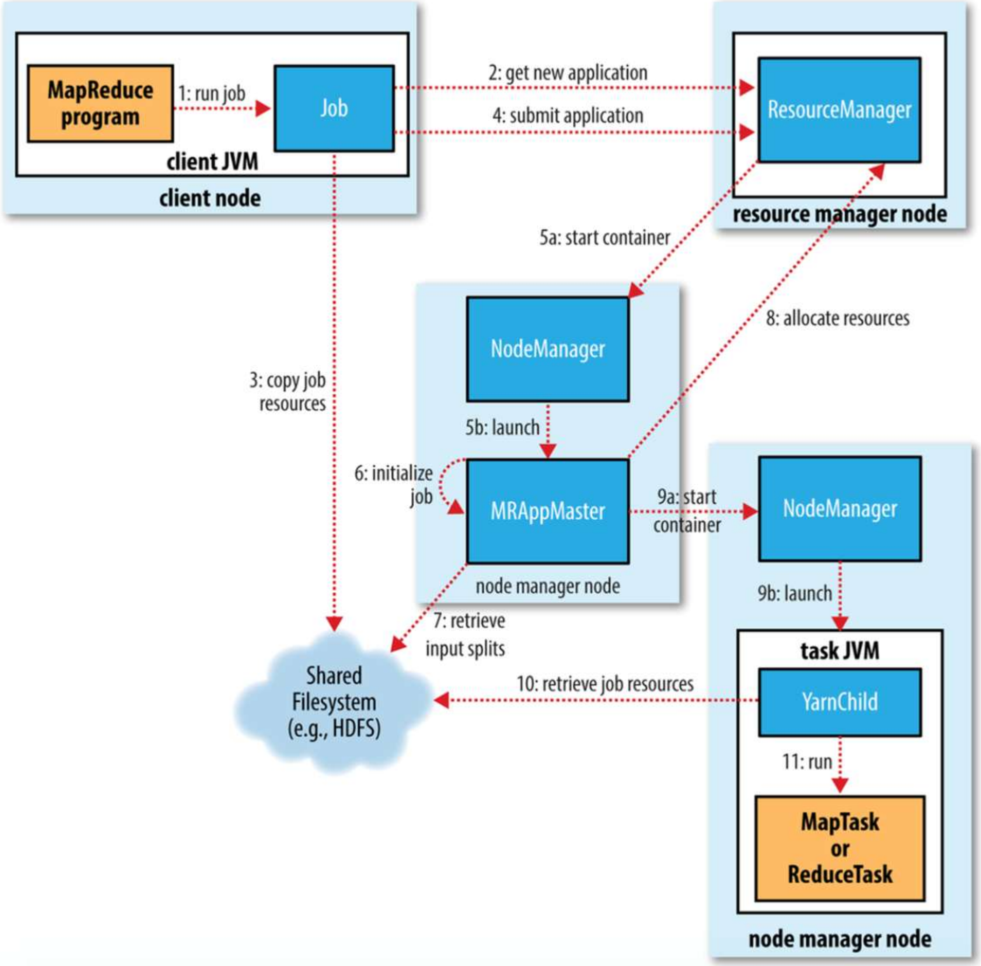
7. The client that submitted the app / job can directly communicate with the AppMaster for progress, status updates. via the application specific protocol.
8. On completion of the app / job, the AppMaster de-registers from ResourceManager and shuts down. So the containers allocated can be re-purposed.



More about AppMaster

- Consider it as a framework specific library that is installed on a special / first container of the application by the Resource Manager
- Responsible for all resource requests from an application perspective
- Is user specific and Resource Manager needs to protect itself / cluster from malicious resource use
  - ✓ This enables RM to be just a scheduler while application needs, tracking / monitoring etc. is left to AppMaster (as application rep) and NodeManager (as Hadoop rep)
  - ✓ With all app specific code moved out of Resource Manager, cluster can be used for multiple data processing engines
- So, resource management can scale to 10K+ nodes while AppMaster doesn't become a cluster-wide bottleneck because it only manages a job / application
- It is possible to have an AppMaster instance manage multiple applications - it is user specific code

Job submission flow



https://stackoverflow.com/questions/34709213/hadoop-how-job-is-send-to-master-and-to-nodes-on-mapreduce

Topics for today

- Hadoop MapReduce
  - ✓ MapReduce runtime
  - ✓ More examples
  - ✓ Hadoop streaming
- Yet Another Resource Negotiator (YARN)
  - ✓ Architectural components
  - ✓ Workflow
  - ✓ **Resource model and implementation**
  - ✓ Resource scheduling
  - ✓ YARN sample commands

YARN Resource model

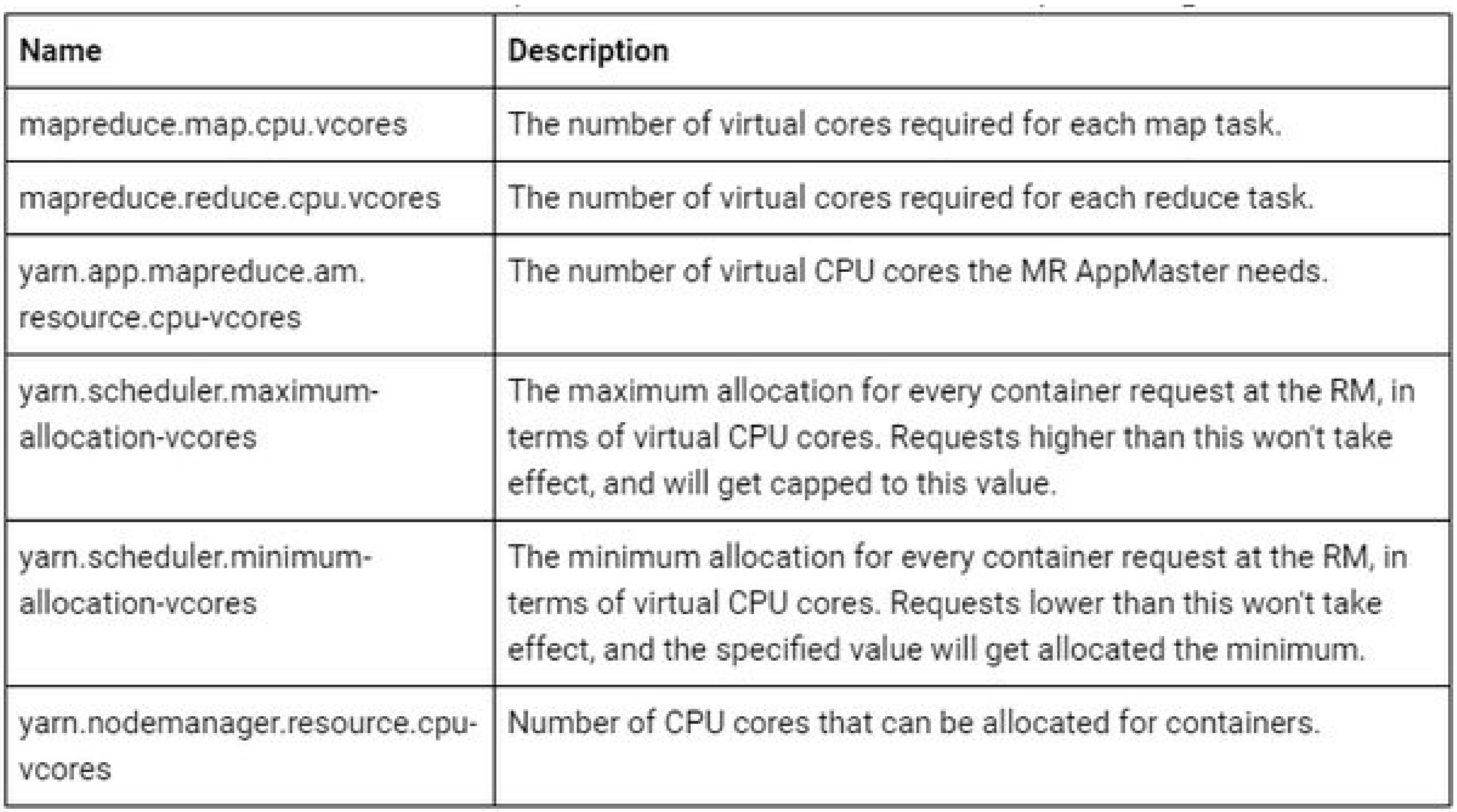
- Resource for an application is a set of containers.
- Each container is defined by :
  - Resource-name (hostname, rack name etc.)
  - Memory (in MB)
    - mapreduce.map.memory.mb
    - mapreduce.reduce.memory.mb
  - CPU (cores)
    - mapreduce.map.cpu.vcores
    - mapreduce.reduce.cpu.vcores
  - Possibly adding - Disk, network IO etc. gradually
- Resource Manager has complete view of resources across nodes to schedule a new request

http://192.168.29.135:8088

What are Vcores

- Vcore is the abbreviation for Virtual Cores and is a type of resource
- Vcores help sharing usage of host CPU in a Hadoop cluster
- Vcore indicates ‘usage share of a CPU core’
- VCores are configured by YARN in such a way that all resources available in the cluster are utilized in the best possible way.
- Vcore to Physical core ratio is decided by the nature of the workload
  - ✓ For CPU intensive applications, this ratio is 1, ie 1 Vcore per physical core
  - ✓ For IO intensive application, this ratio >= 4

Configuring Vcore allocation



Name	Description
mapreduce.map.cpu.vcores	The number of virtual cores required for each map task.
mapreduce.reduce.cpu.vcores	The number of virtual cores required for each reduce task.
yarn.app.mapreduce.am. resource.cpu-vcores	The number of virtual CPU cores the MR AppMaster needs.
yarn.scheduler.maximum-allocation-vcores	The maximum allocation for every container request at the RM, in terms of virtual CPU cores. Requests higher than this won't take effect, and will get capped to this value.
yarn.scheduler.minimum-allocation-vcores	The minimum allocation for every container request at the RM, in terms of virtual CPU cores. Requests lower than this won't take effect, and the specified value will get allocated the minimum.
yarn.nodemanager.resource.cpu-vcores	Number of CPU cores that can be allocated for containers.

YARN Tuning and Configuration

Tuning

- A YARN cluster is composed of host machines.
- Hosts provide memory and CPU resources.
- A Vcore (virtual core) is a usage share of a host CPU.
- Yarn containers are execution engines constituted by Vcores and memory
- Tuning YARN consists of optimally defining containers on worker hosts
- Tune YARN hosts to optimize the use of Vcores and memory

Configuration

- YARN and MapReduce have many configurable properties.
- The YARN tuning spreadsheet (**yarn-tuning-guide.xlsx**) lists the essential subset of these properties that are most likely to improve performance for common MapReduce applications
- (For Cloudera distribution - Watch this video before using the spreadsheet - https://youtu.be/lykWFhrGvJ4
- YARN tuning has three phases. (The phases correspond to the tabs in the YARN tuning spreadsheet)
  1. Cluster configuration, where you configure your hosts.
  2. YARN configuration, where you quantify memory and vcores.
  3. MapReduce configuration, allocates minimum and maximum resources for map and reduce tasks.

Resource requests

- A resource request :

  `<resource-name, priority, resource-requirement, number-of-containers>`

  resource-name is a host, rack or * for a container

  priority is within application for this request

  resource-requirement is CPU, memory etc. needed by a container

  number-of-containers is count of above spec containers needed by application

- One or more containers is the result of a successful allocation request. It is a “right” given to an application to use certain amount of resources on a specific host / node.

- AppMaster presents that “right” to the Node Manager on a host to use resources.
- Security and verification is done by NodeManager.

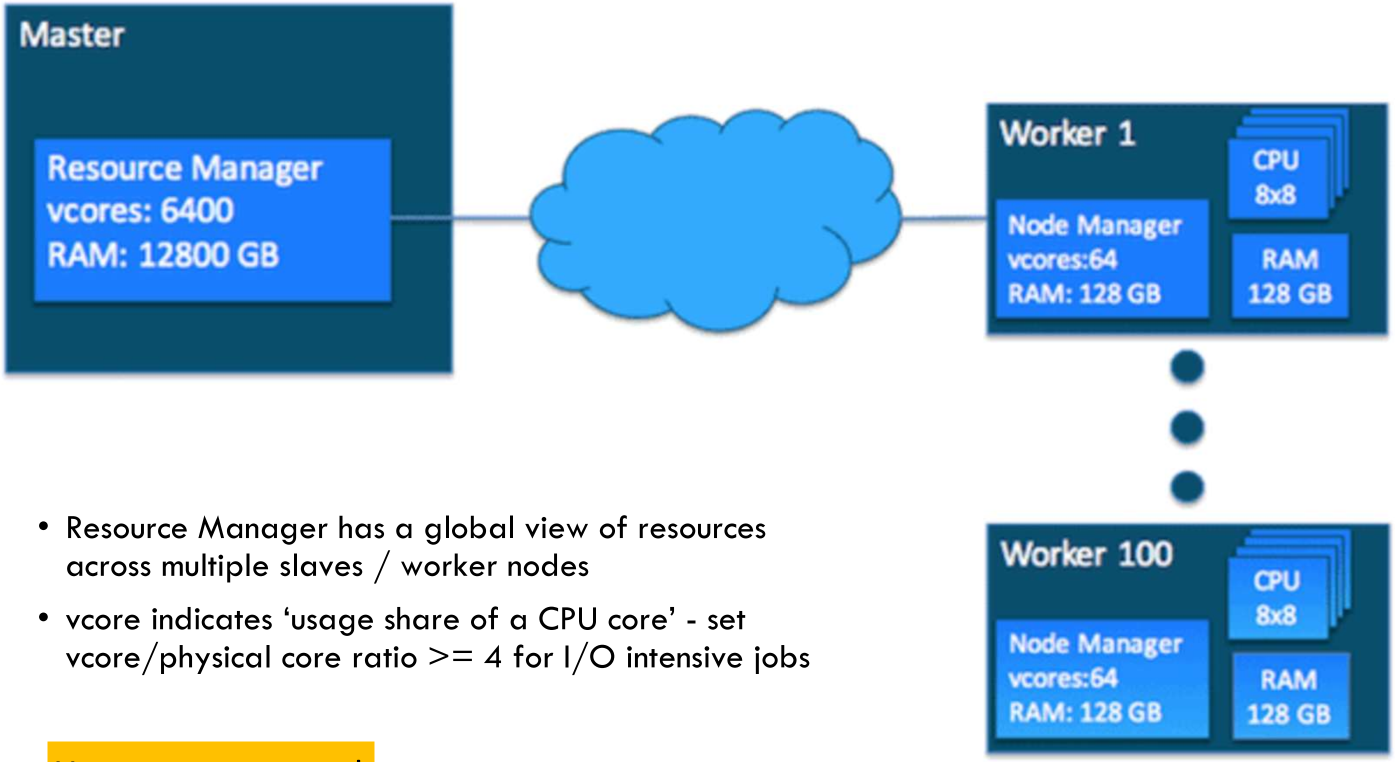
What can be launched in a Container ?

The YARN Container launch specification API is platform agnostic and contains:

- Command line to launch the process within the container
- Environment variables
- Local resources necessary on the machine prior to launch, such as jars, shared-objects, auxiliary data files etc.
- Security-related tokens.

This allows the Application Master to work with the NodeManager to launch containers ranging from simple shell scripts to C/Java/Python processes on Unix/Windows to full-fledged virtual machines (e.g. KVMs)

Example: Global resource view

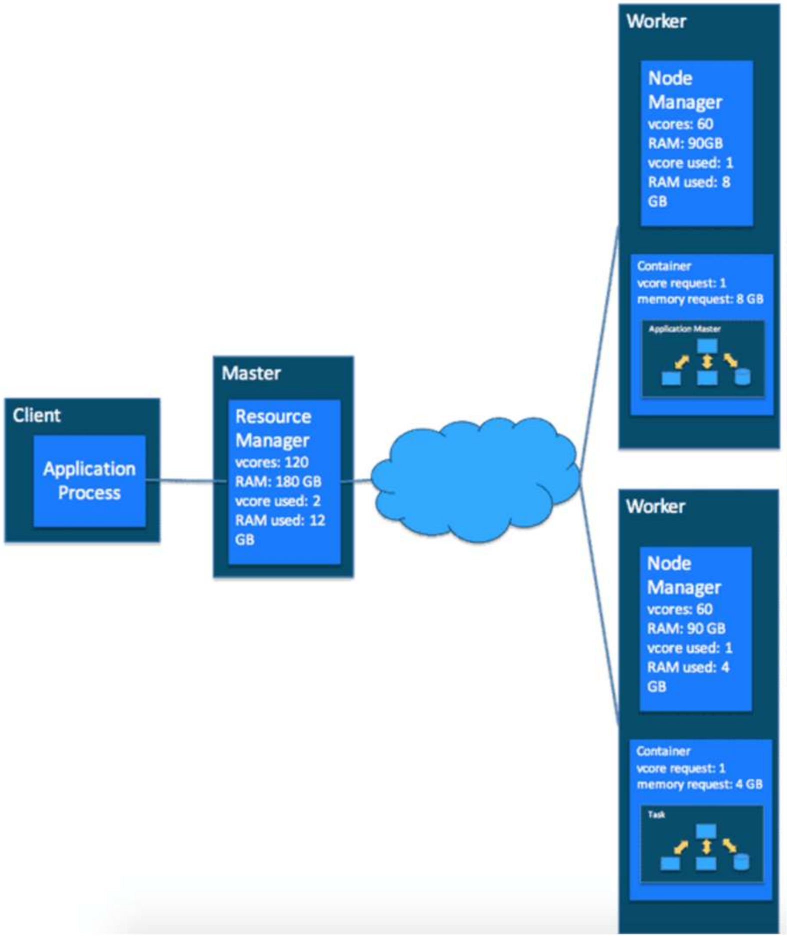


8 physical cores with 8 vcores on each

- Resource Manager has a global view of resources across multiple slaves / worker nodes
- vcore indicates 'usage share of a CPU core' - set vcore/physical core ratio >= 4 for I/O intensive jobs

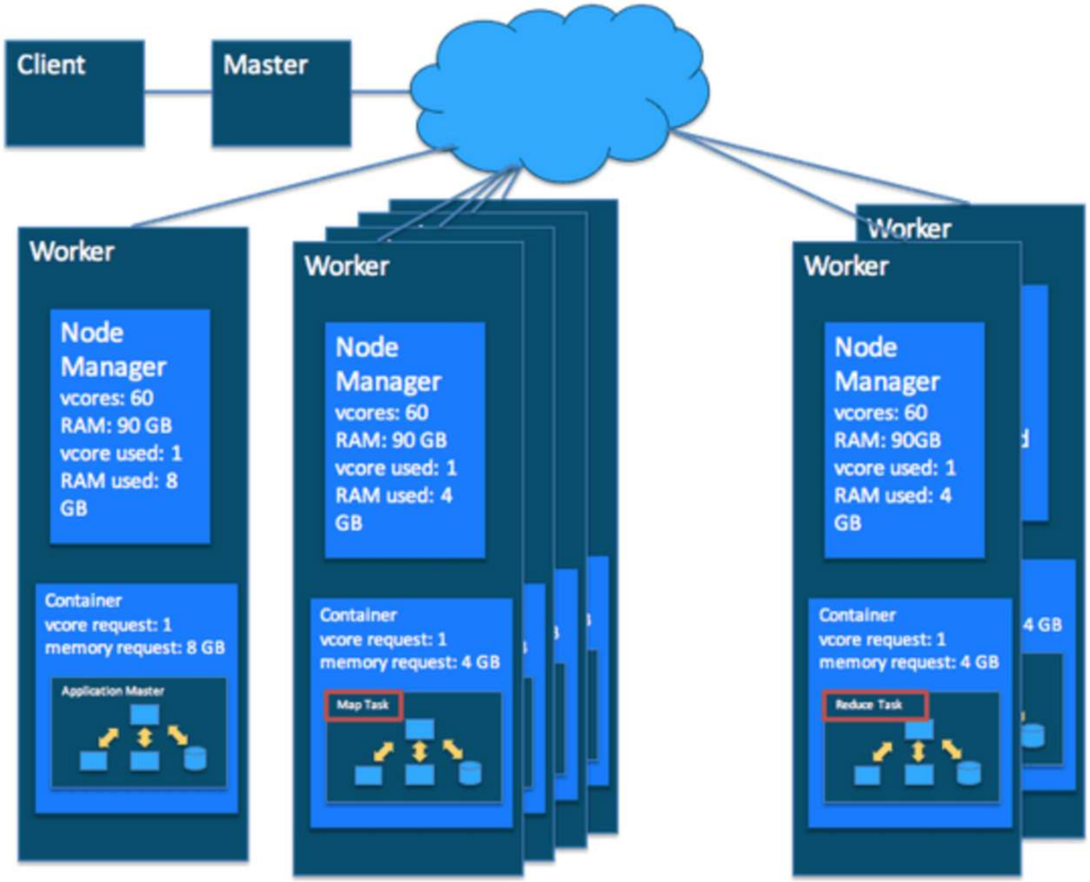
Yarn top command

Example: Containers on Workers



- Containers are allocated specific to application / job
- First container for an application has the Application Master
- Other containers to run tasks are negotiated by Application Master

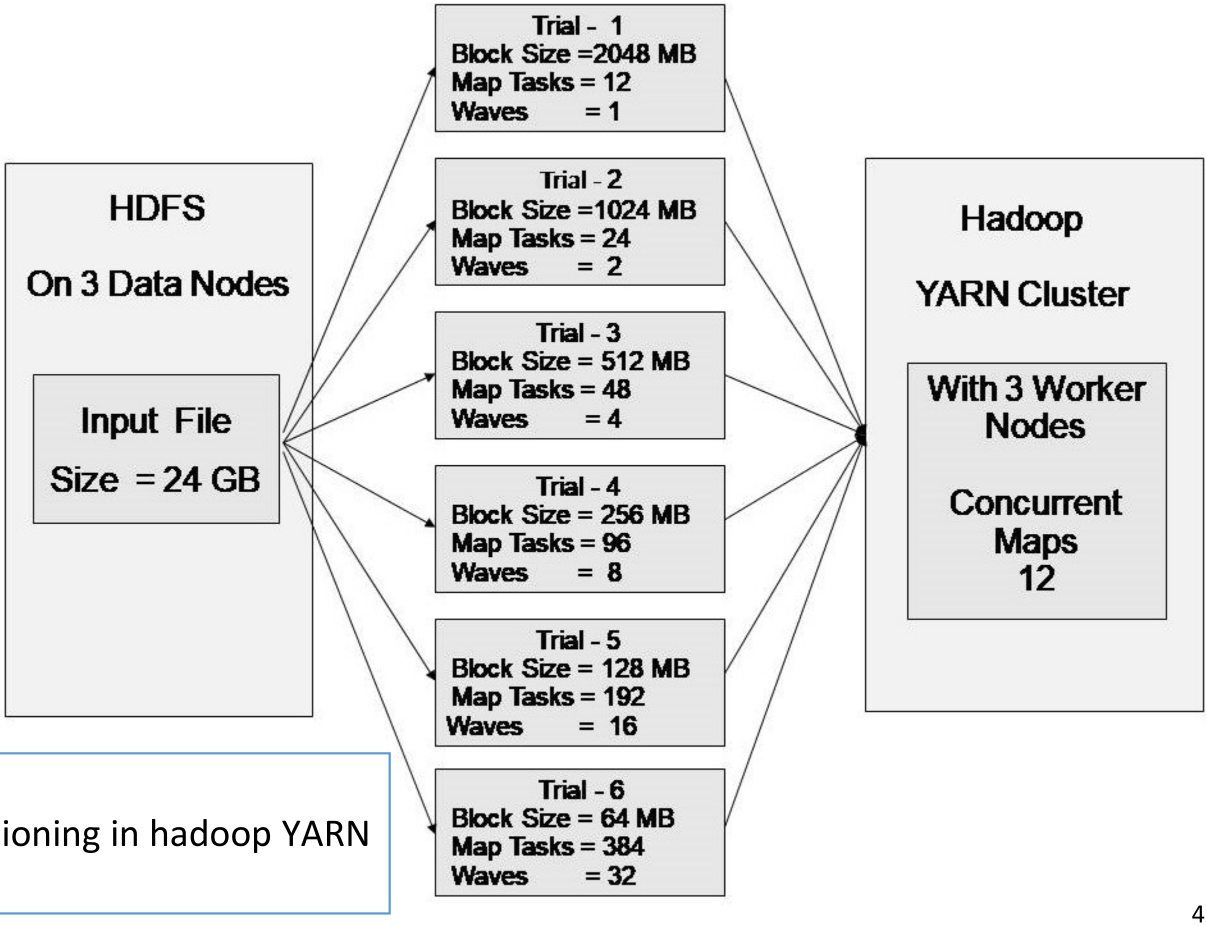
Example: Map and Reduce tasks in containers



- Application specific tasks are started within the containers

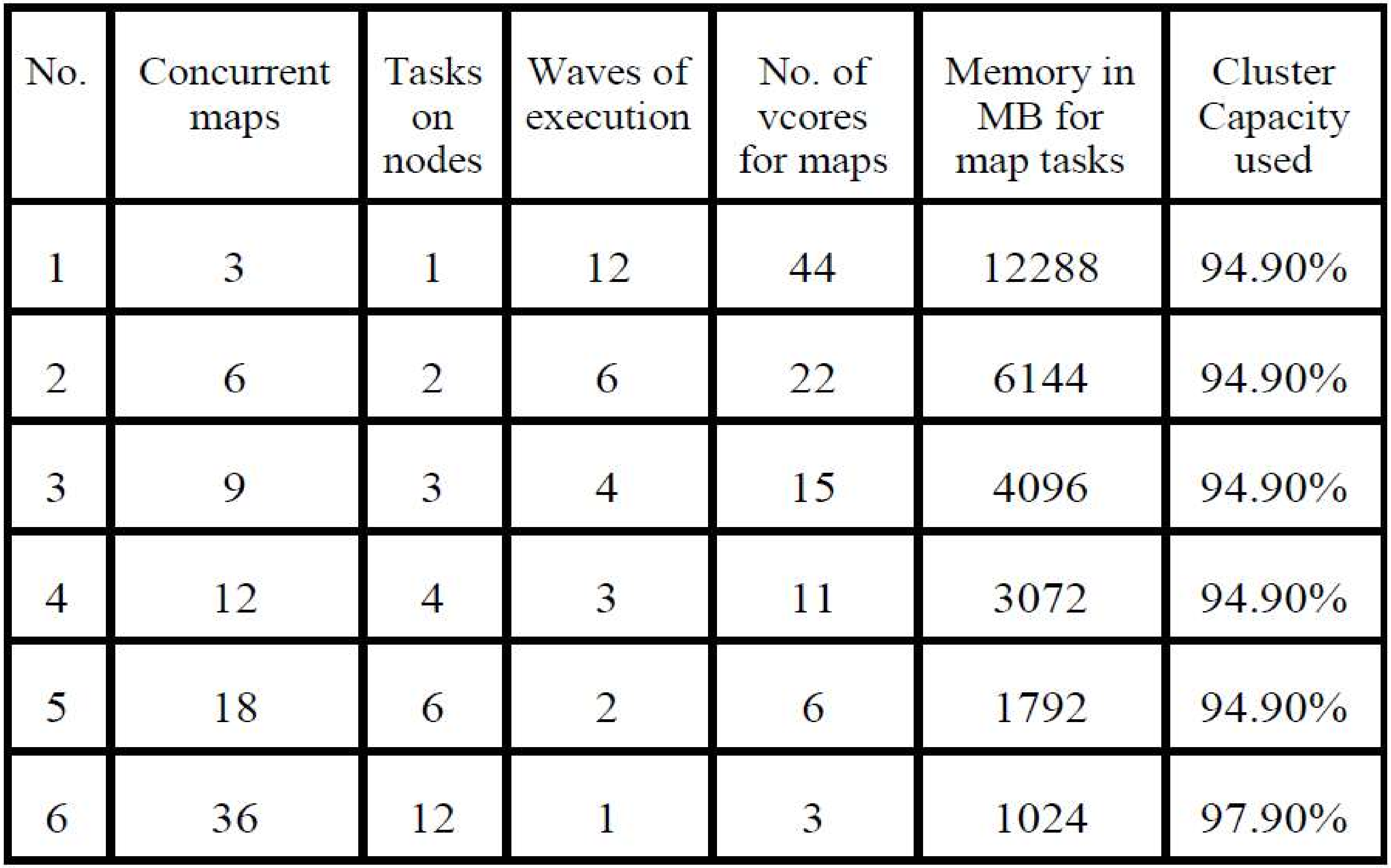
Block Size, Map tasks, Waves of execution

- No of concurrent containers depends on portion of the total resources allocated to it
- When no of map tasks > no of concurrent containers, map tasks executes in stages
- With containers having higher resource allocation, no of concurrent containers will be less
- With containers having less resource allocation, no of concurrent containers will be more..
- To guarantee maximum performance with minimum resources, we need to configure the YARN framework for optimum concurrency level in execution of containers.



Reference Paper:
Study of execution parallelism by resource partitioning in hadoop YARN
http://ieeexplore.ieee.org/document/8126000/

Concurrent maps on 3 nodes– File Size 9GB, Block size 256Mb



No.	Concurrent maps	Tasks on nodes	Waves of execution	No. of vcores for maps	Memory in MB for map tasks	Cluster Capacity used
1	3	1	12	44	12288	94.90%
2	6	2	6	22	6144	94.90%
3	9	3	4	15	4096	94.90%
4	12	4	3	11	3072	94.90%
5	18	6	2	6	1792	94.90%
6	36	12	1	3	1024	97.90%

Topics for today

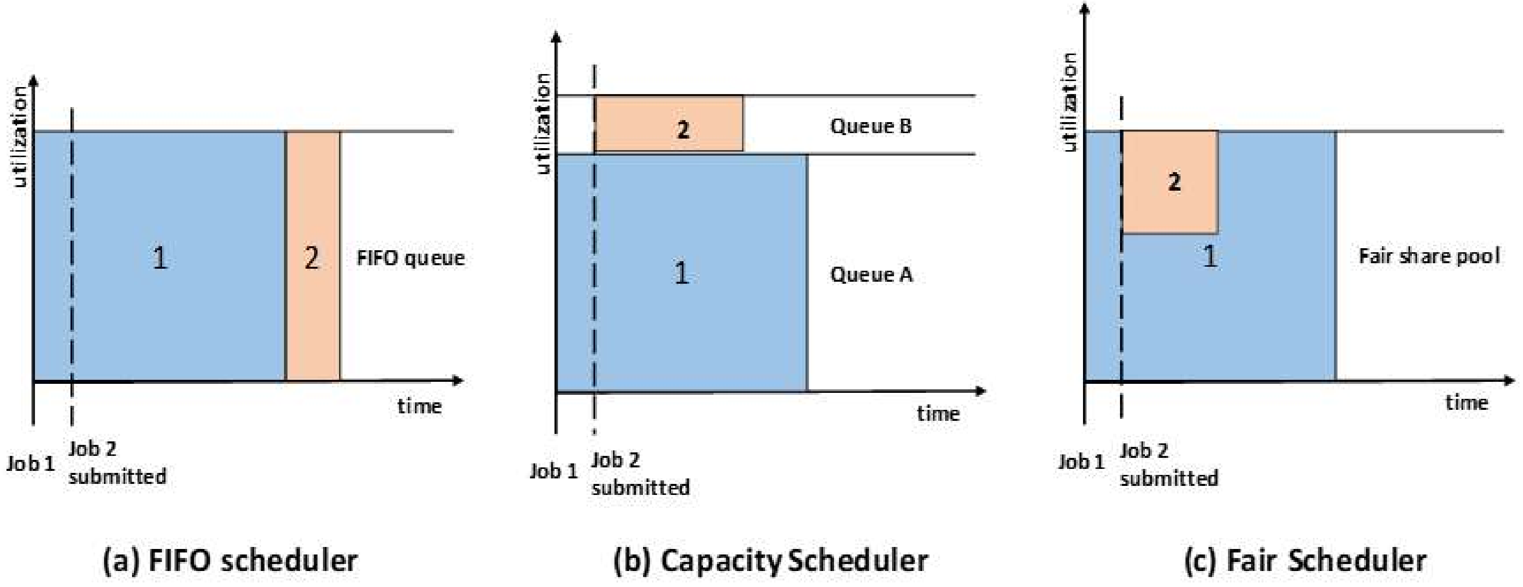
- Hadoop MapReduce
  - ✓ MapReduce runtime
  - ✓ More examples
  - ✓ Hadoop streaming
- Yet Another Resource Negotiator (YARN)
  - ✓ Architectural components
  - ✓ Workflow
  - ✓ Resource model and implementation
  - ✓ **Resource scheduling**
  - ✓ YARN sample commands

Schedulers in YARN

1. FIFO – First in, First Out
2. Capacity - Maintains separate queue for different types of jobs (Default on Apache Hadoop-3.3.5)
3. Fair share scheduler (Fair scheduler) - Dynamically balances resources between all accepted jobs

These are pluggable to YARN, Any one at a time

Source: [Hadoop: The Definitive Guide]

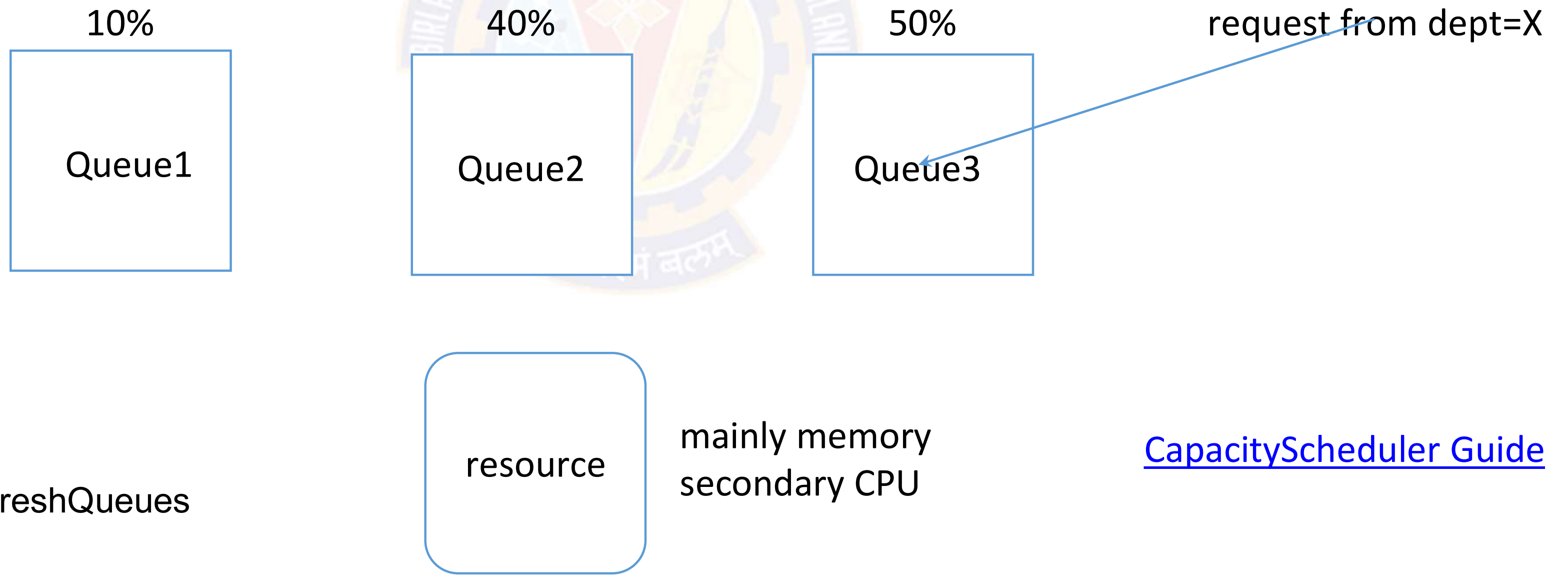


(a) FIFO scheduler (b) Capacity Scheduler (c) Fair Scheduler

Figure 1: YARN Schedulers' cluster utilization vs. time

Resource Manager scheduling - Capacity scheduler

- When container requests are made, which request do you satisfy ?
  - ✓ So put a queue where a request is entered, and a scheduler will pick requests from the queue
- Typically, multiple queues with different shares of resources (by capacity) because you want to partition system resources by some criteria, e.g. organisation / app type etc.
- **`mapred queue [-list] | [-info <job-queue-name> [-showJobs]]`**



mainly memory
secondary CPU

yarn rmadmin -refreshQueues

[CapacityScheduler Guide]

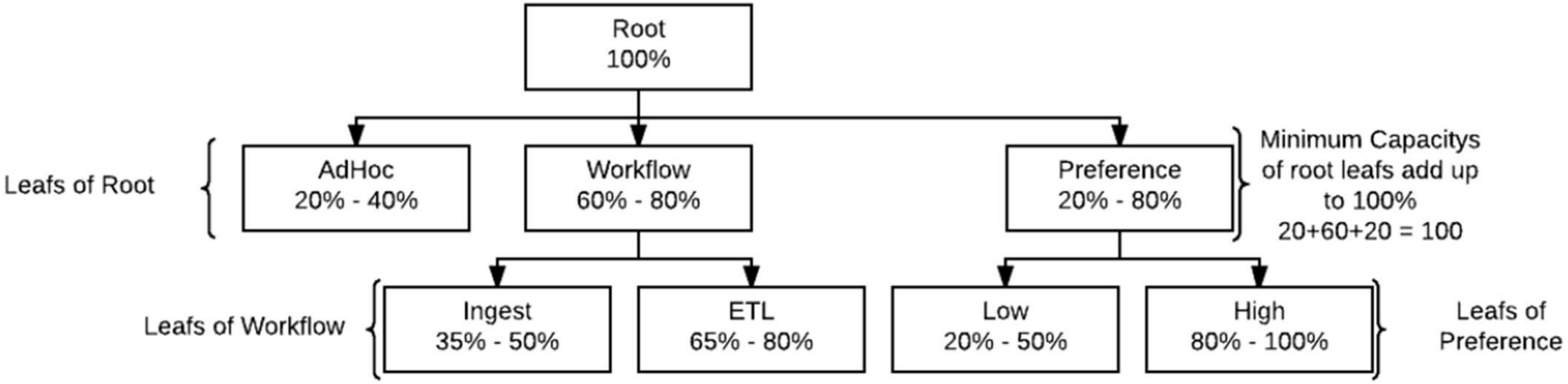
Resource Manager scheduling - Capacity scheduler

- Multiple organizations can share a cluster
  - ✓ Individual org level **capacity reservations** are strictly met with isolation to have multi-tenancy
- Implements **hierarchical queues** where first level is between orgs and second level queue is within an org
  - ✓ Enables capacity to be first shared by users within org before any spare capacity (from set limits) is used by other orgs
- Security done by ACLs for job submission in relevant queues
- **Elasticity to use spare capacity** with pre-emption capability to stop jobs when higher priority jobs come in or the spare capacity is no longer available
- Configurable scheduling
  - ✓ Resource based scheduling for applications that use some resource more, e.g. memory intensive
  - ✓ Users can map jobs to specific queues and define placement rules, e.g. user/group/app can determine where the resources are allocated
  - ✓ Priority scheduling - higher value means high priority
  - ✓ Applications can ask for absolute value of resources

```
yarn.resourcemanager.scheduler.class =  org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler
```

Capacity scheduler: Hierarchical queue allocations

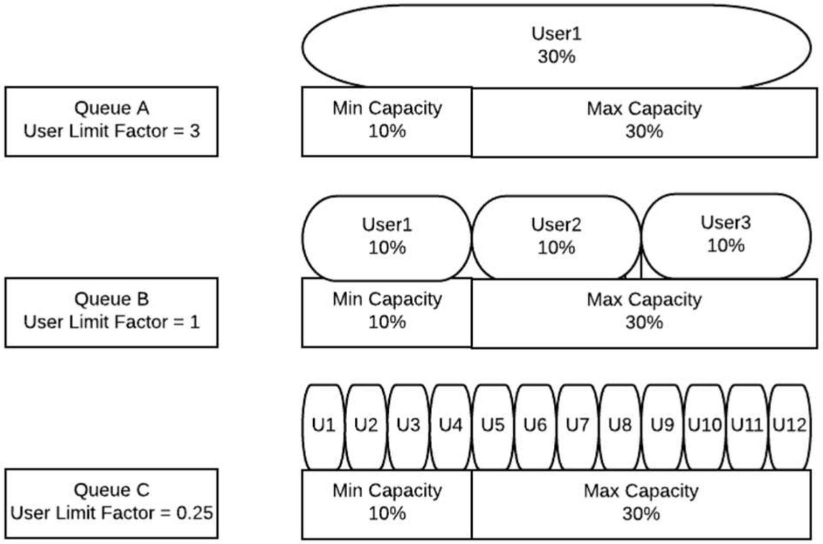
- Min capacity - what needs to be given to a queue even when cluster is at max usage
- Max capacity - an elastic like capacity that allows queues to make use of resources which are not being used to fill minimum capacity demand in other queues.



https://blog.cloudera.com/yarn-capacity-scheduler/

Capacity scheduler : User level allocations

- Min user percentage : Smallest amount of resources a single user can get access to - varies between min and max of queue capacity.
- User limit factor: Used to control max resources given to a user as a factor of min user percentage.
- *Note: Be aware of containers that are long lived vs short lived. Latter (high container churn) helps to balance queues better. The former use limit factors, pre-emption, or even dedicated queues without elastic capacity.*



https://blog.cloudera.com/yarn-capacity-scheduler/

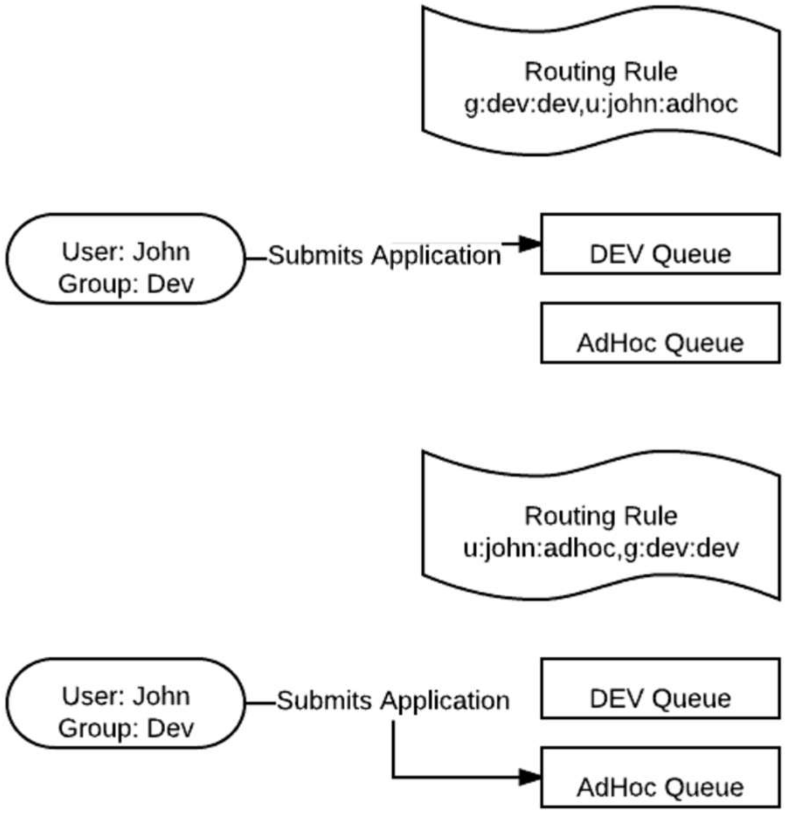
Capacity scheduler : Ordering policies within queue

- FIFO ordering
  - ✓ Ordering based on arrival time
  - ✓ Large applications can block each other and cause user discontent
  - ✓ There is no preemption within a queue anyway
- FAIR ordering
  - ✓ Give resources to smallest requests first and new applications with least resources to get started (like shortest job first)
  - ✓ Works well when there is good container churn within a queue

https://blog.cloudera.com/yarn-capacity-scheduler/

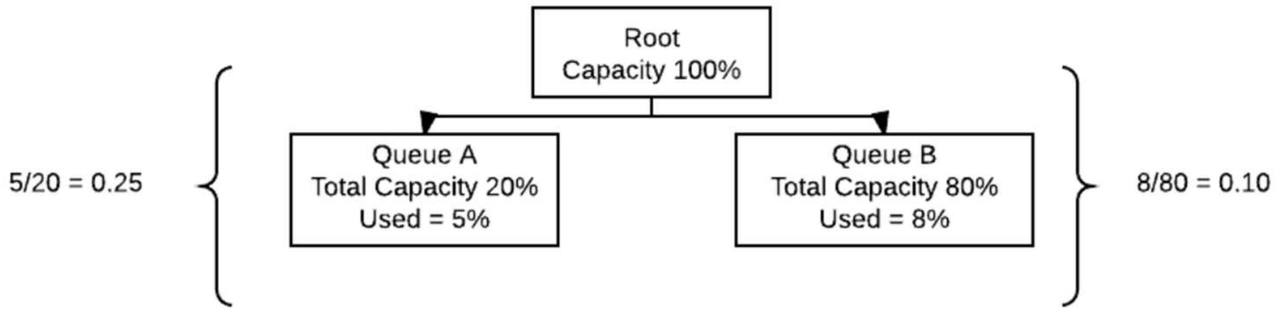
Capacity scheduler : Mapping to queue

- User and group mapping to queue
- Depends on what order is specified for mapping in routing rules



https://blog.cloudera.com/yarn-capacity-scheduler/

Capacity scheduler : Priority

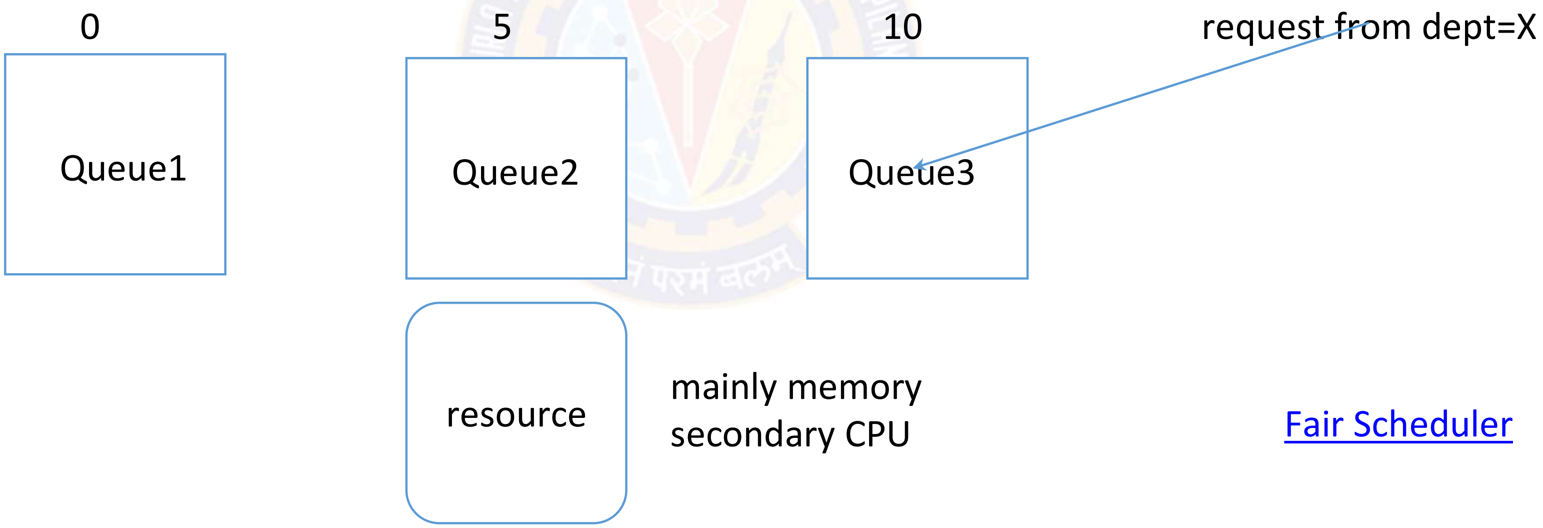


- Influence which queue gets new requests beyond resource utilization driven
- Queue A relative capacity utilization is 5 / 20 = 0.25
- Queue B relative capacity utilization is 8 / 80 = 0.10
- So new requests will go to Queue B
- Increasing priority on Queue A will avoid that

https://blog.cloudera.com/yarn-capacity-scheduler/

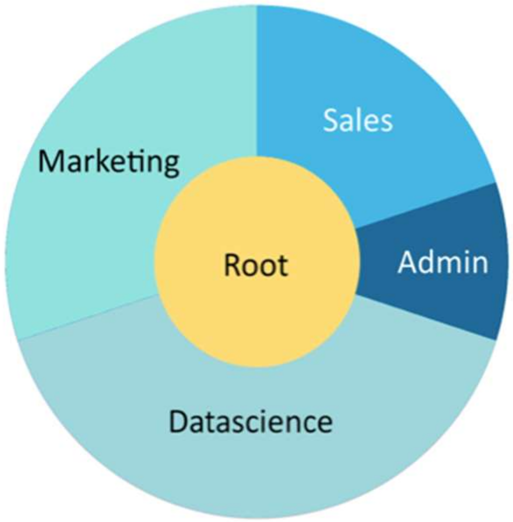
Resource Manager scheduling - Fair Share scheduler

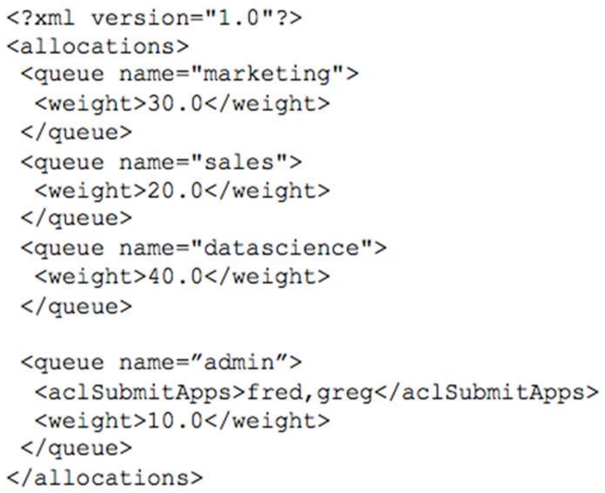
- **Weight replaces min%-max% capacity share**
- So, sum of weights don’t need to add up to 1 or 100% unlike capacity
- Weight is a measure of what a queue considers as a fair share it needs and weights make sense as ratios of each other
  - ✓ They can be used to calculate approx capacity allocations
- 0 means best effort basis allocation is good enough - it doesn’t imply 0% capacity
  - ✓ **If no requests in other queues, it may occupy all the capacity as best effort**



Fair Scheduler

Fair scheduler: Example





```
<?xml version="1.0"?>
<allocations>
 <queue name="marketing">
  <weight>30.0</weight>
 </queue>
 <queue name="sales">
  <weight>20.0</weight>
 </queue>
 <queue name="datascience">
  <weight>40.0</weight>
 </queue>

 <queue name=”admin”>
  <aclSubmitApps>fred,greg</aclSubmitApps>
  <weight>10.0</weight>
 </queue>
</allocations>
```

- Weights determine ratio of usage
  - ✓ Marketing : 30
  - ✓ Sales : 20
  - ✓ Datascience : 40
  - ✓ Admin : 10
- Weights need not add up to 100
- Change weights to change the fair share
- Best effort can be weight 0 - which means no fair share, but will get what is remaining if there is spare capacity available in the cluster
- So, min and max as in Capacity scheduler need not be set
- Weights under a hierarchy further indicate fair share within the fair share of the parent

Resource Manager scheduling - Fair scheduler

- Ensures that all apps get a “fair” share on the average of the cluster - more popular approach
- Primary resource is memory but CPU + memory can also be configured with one as the dominant resource for deciding fairness
- Works well where shorter jobs are not starved and longer jobs also get to finish without getting unduly delayed when there are many short jobs
- Apps are assigned to queues and unless specified, all apps will go to default queue
- Fair share scheduler works within a queue and also across the queues
- Fair share of a queue is determined by weight (0 means no fair share - just best effort)
  - ✓ No capacity min-max specified - just a single weight
  - ✓ Queue level: A queue can be assigned a minimum share (determined by weight), e.g. a queue for production apps can be assigned a queue with a certain minimum share of the cluster. Excess can be shared with other queues.
  - ✓ App level: Works with app priorities where a weight can be assigned to an app to compute fair share (enable yarn.scheduler.fair.sizebasedweight=TRUE)
    - Weight is a function of natural log of memory demanded by the app
- A limit can also be put per queue / per user on how many apps will be picked up for scheduling
- Queues can be organized hierarchically
- Plugging-in Fair share scheduler

```
In yarn-site.xml
<property>
    <name>yarn.resourcemanager.scheduler.class</name>
    <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler</value>
</property>
```

Preemption across queues in Capacity / Fair scheduler

- Scenario
  - ✓ Application A is demanding more resources and using elastic capacity of a queue Q1
  - ✓ So Q1 gets unused min allocation from queue Q2
  - ✓ Suddenly Q2 needs its min capacity back because of new applications
- Preemption enables a queue to reclaim elastic resources to bring back its min allocation without making new applications wait till the older application tasks using elastic capacity finishes
- Preemption does not save the context of the task from where it be restarted
- Preemption is done by killing the task (There will be wastage of computing power)
- Preemption tries not to kill tasks of entire application to reclaim resources
  - ✓ It tries to kill younger tasks – tasks in initial stages of execution
  - ✓ Or kill reducers (not mappers that have done a lot of work already) because minimum work is lost
- Preemption will not do anything unless it can fulfil entire allocation request
- Preemption only reclaims min allocation & NOT max allocation (Capacity Scheduler)
- Within a queue one has to work with <u>User limit factor or FIFO/FAIR ordering policies.</u>

From below example, If queue1 is at 80% of its 50% share and need more resources and waiting for more than 60 seconds, queue1 is allowed to preempt containers from queue2. On a similar situation for queue2, it cannot preempt containers from queue1 because it does not have these parameters set.

```
<allocations>
  <queue name="queue1">

<fairSharePreemptionTimeout>60</fairSharePreemptionTimeout>

<fairSharePreemptionThreshold>0.80</fairSharePreemptionThreshold>
        <weight>1.0</weight>
  </queue>
  <queue name="queue2">
        <weight>1.0</weight>
  </queue>
  .
  .
  .
</allocations>
```

Difference between Capacity scheduler and Fair scheduler

	Capacity	Fair
	• Allows sharing a large cluster while giving each department a minimum capacity guarantee. • Available resources in the Hadoop cluster are partitioned among multiple departments who collectively fund the cluster based on computing needs. • Set up by queues for each dept with specified amount of capacity. • Jobs within the same queue get scheduled in FIFO order. • Added benefit is that an organization can access any excess capacity not being used by others. • Provides elasticity for the organizations in a cost-effective manner	• A method of assigning resources to jobs such that all jobs get, on average, an equal share of resources over time. • When there is a single job running, that job uses the entire cluster. • When other jobs are submitted, tasks slots that free up are assigned to the new jobs, so that each job gets roughly the same amount of CPU time.
	Default in Apache Hadoop 3.3.5	

Log management in YARN

- Unlike Hadoop 1, YARN makes sure that through Node Manager, local logs are moved to a file system like HDFS in configurable directories
- YARN commands / UI can be used to retrieve these logs that are at node level or application level
- Log retention can be configured

Some YARN commands

Display yarn jobs:	yarn top
Display status of a specific queue:	yarn queue -status BDS_Q1
Start an application:	yarn app start <app name>
Start a JAR file:	yarn jar <jar> <main class> <args>
Stop an application:	yarn app stop <appid>
Change priority:	yarn app -updatePriority <priority> -appID <appid>
Know which applications are running:	yarn app -list -appStates RUNNING
Get logs from an application:	yarn logs -appld <appid>
Changing Queue Configuration:	yarn rmadmin -refreshQueues

https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YarnCommands.html

Summary

- Real world use cases with MapReduce model of computing
- Hadoop streaming
- YARN – Yet another resource negotiator
- Pluggable schedulers in YARN
- Yarn command interface

Next Session:
Hadoop ecosystem tech: Pig, Hive, HBase