Lecture-8
Big Data Systems(CCZG522/SEZG522)

BITS Pilani
Pilani Campus

Slides: Courtesy:.Prof. Anindya

innovate achieve lead

BITS Pilani
Pilani Campus

Second Semester

2024-25

Lecture-8 Contents

- Hadoop MapReduce
  - ✓ Walkthrough a MapReduce program
  - ✓ MapReduce runtime
  - ✓ More examples
- Yet Another Resource Negotiator (YARN)
  - ✓ Architectural components
  - ✓ Workflow
  - ✓ Resource scheduling
  - ✓ YARN sample commands

Sample MapReduce program (recap)

	A	B	C	D	E	F	G	H	I	J	K	L
1	Transaction_date	Product	Price	Payment_	Name	City	State	Country	Account_Created	Last_Login	Latitude	Longitude
2	01-02-2009 06:17	Product1	1200	Mastercar	carolina	Basildon	England	United Kir	01-02-2009 06:00	01-02-2009 06:08	51.5	-1.11667
3	01-02-2009 04:53	Product1	1200	Visa	Betina	Parkville	MO	United Sta	01-02-2009 04:42	01-02-2009 07:49	39.195	-94.6819
4	01-02-2009 13:08	Product1	1200	Mastercar	Federica e	Astoria	OR	United Sta	01-01-2009 16:21	01-03-2009 12:32	46.18806	-123.83
5	01-03-2009 14:44	Product1	1200	Visa	Gouya	Echuca	Victoria	Australia	9/25/05 21:13	01-03-2009 14:22	-36.1333	144.75
6	01-04-2009 12:56	Product2	3600	Visa	Gerd W	Cahaba He	AL	United Sta	11/15/08 15:47	01-04-2009 12:45	33.52056	-86.8025
7	01-04-2009 13:19	Product1	1200	Visa	LAURENCE	Mickleton	NJ	United Sta	9/24/08 15:19	01-04-2009 13:04	39.79	-75.2381
8	01-04-2009 20:11	Product1	1200	Mastercar	Fleur	Peoria	IL	United Sta	01-03-2009 09:38	01-04-2009 19:45	40.69361	-89.5889
9	01-02-2009 20:09	Product1	1200	Mastercar	adam	Martin	TN	United Sta	01-02-2009 17:43	01-04-2009 20:01	36.34333	-88.8503
10	01-04-2009 13:17	Product1	1200	Mastercar	Renee Elis	Tel Aviv	Tel Aviv	Israel	01-04-2009 13:03	01-04-2009 22:10	32.06667	34.76667
11	01-04-2009 14:11	Product1	1200	Visa	Aidan	Chatou	Ile-de-Fra	France	06-03-2008 04:22	01-05-2009 01:17	48.88333	2.15
12	01-05-2009 02:42	Product1	1200	Diners	Stacy	New York	NY	United Sta	01-05-2009 02:23	01-05-2009 04:59	40.71417	-74.0064
13	01-05-2009 05:39	Product1	1200	Amex	Heidi	Eindhover	Noord-Bra	Netherlan	01-05-2009 04:55	01-05-2009 08:15	51.45	5.466667
14	01-02-2009 09:16	Product1	1200	Mastercar	Sean	Shavano P	TX	United Sta	01-02-2009 08:32	01-05-2009 09:05	29.42389	-98.4933
15	01-05-2009 10:08	Product1	1200	Visa	Georgia	Eagle	ID	United Sta	11-11-2008 15:53	01-05-2009 10:05	43.69556	-116.353
16	01-02-2009 14:18	Product1	1200	Visa	Richard	Riverside	NJ	United Sta	12-09-2008 12:07	01-05-2009 11:01	40.03222	-74.9578
17	01-04-2009 01:05	Product1	1200	Diners	Leanne	Julianstov	Meath	Ireland	01-04-2009 00:00	01-05-2009 13:36	53.67722	-6.31917

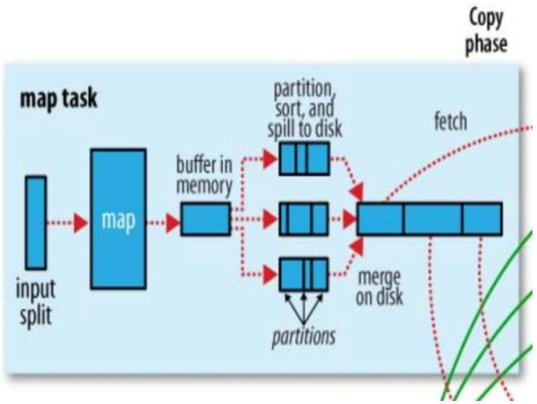
```
Argentina       1
Australia       38
Austria 7
Bahrain 1
Belgium 8
Bermuda 1
Brazil  5
Bulgaria        1
CO      1
Canada  76
Cayman Isls     1
China   1
Costa Rica      1
Country 1
Czech Republic  3
Denmark 15
Dominican Republic      1
Finland 2
France  27
Germany 25
Greece  1
Guatemala       1
Hong Kong       1
Hungary 3
Iceland 1
India   2
```

count tx by country

https://www.guru99.com/create-your-first-hadoop-program.html

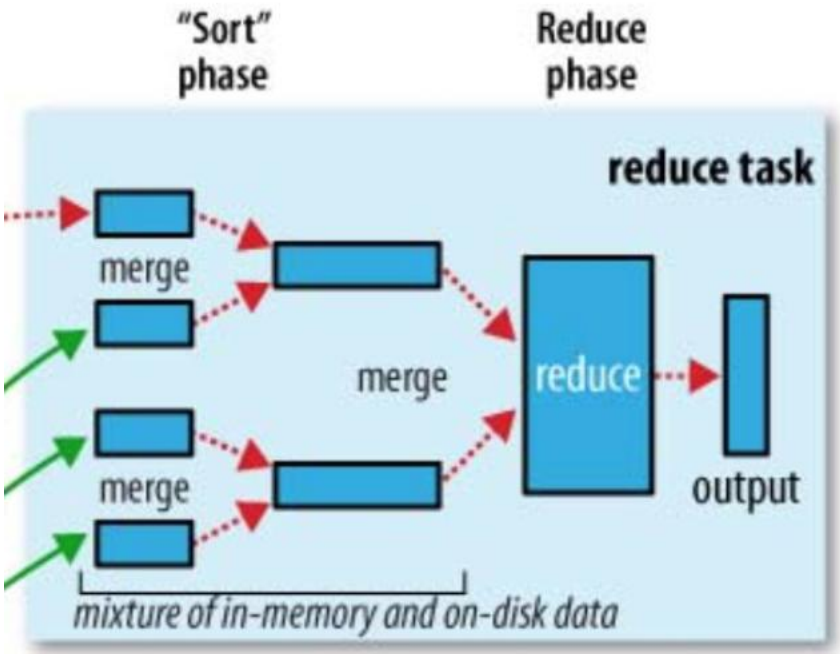
Map

1. **RecordReader**
   – Reads each record created by input splitter to pass key-value pair into Map
   – Could be text, binary etc.
   – Examples: LineRecordReader
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)



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}>
   - 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



MapReduce Programming Architecture

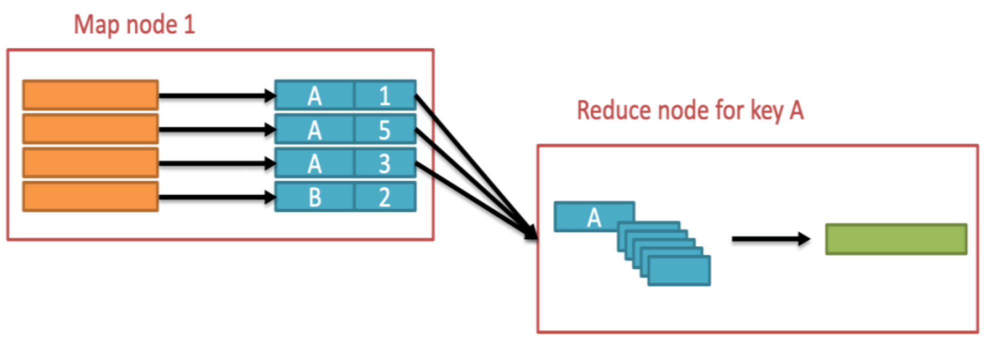
1. Input dataset is split into multiple pieces of data (several small sets)
2. Framework creates a master and several worker processes and executes the worker processes remotely
3. Several Map tasks work simultaneously and read pieces of data that were assigned to each map. Map worker uses the map function to extract only those data that are present on their server and generates key/value pair for the extracted data.
4. Map worker uses partitioner function to divide the data into regions. Partitioner decides which reducer should get the output of specified mapper.
5. When the map workers complete their work, the master instructs the reduce workers to begin their work.
6. 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.
7. Then it calls reduce function on every unique key. This function writes output to the file.
8. When all the reduce workers complete their work, the master transfers the control to the user program.

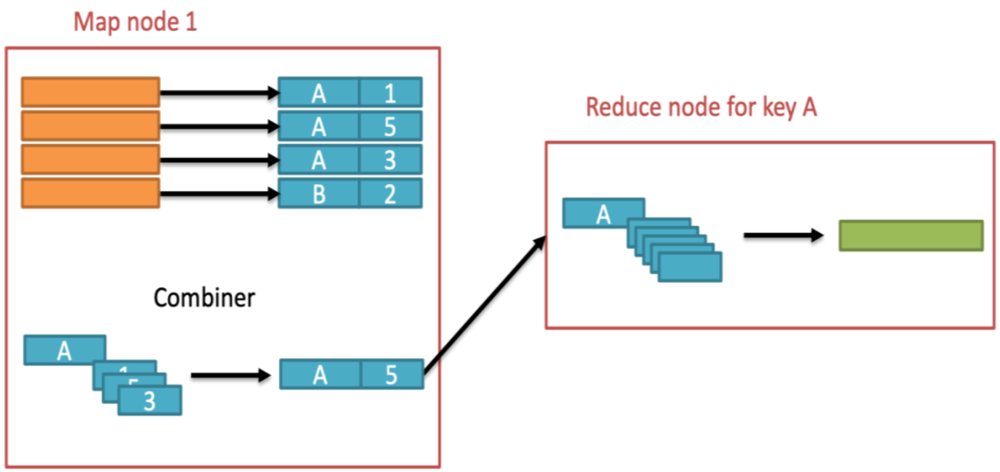
A word about Combiner

- Combiners can optimise the reduce by pre-processing on each node to compress data but output has to be same type as a map

- <u>The reducer can also be set as the combiner class only if reduce is an associative and commutative operation</u>
  - max(1,2, max(3,4,5)) = max(max(2,4), max(1,5,3)) or any other order
  - avg(1,2,avg(3,4,5)) = 2.33… but avg(avg(2,4),avg(1,5,3)) = 3

- If not C&A then optionally write a new combiner logic





Performance optimisations

We studied types of parallelism earlier

The goal for a parallel computation is balanced and maximum utilisation 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

MR job execution

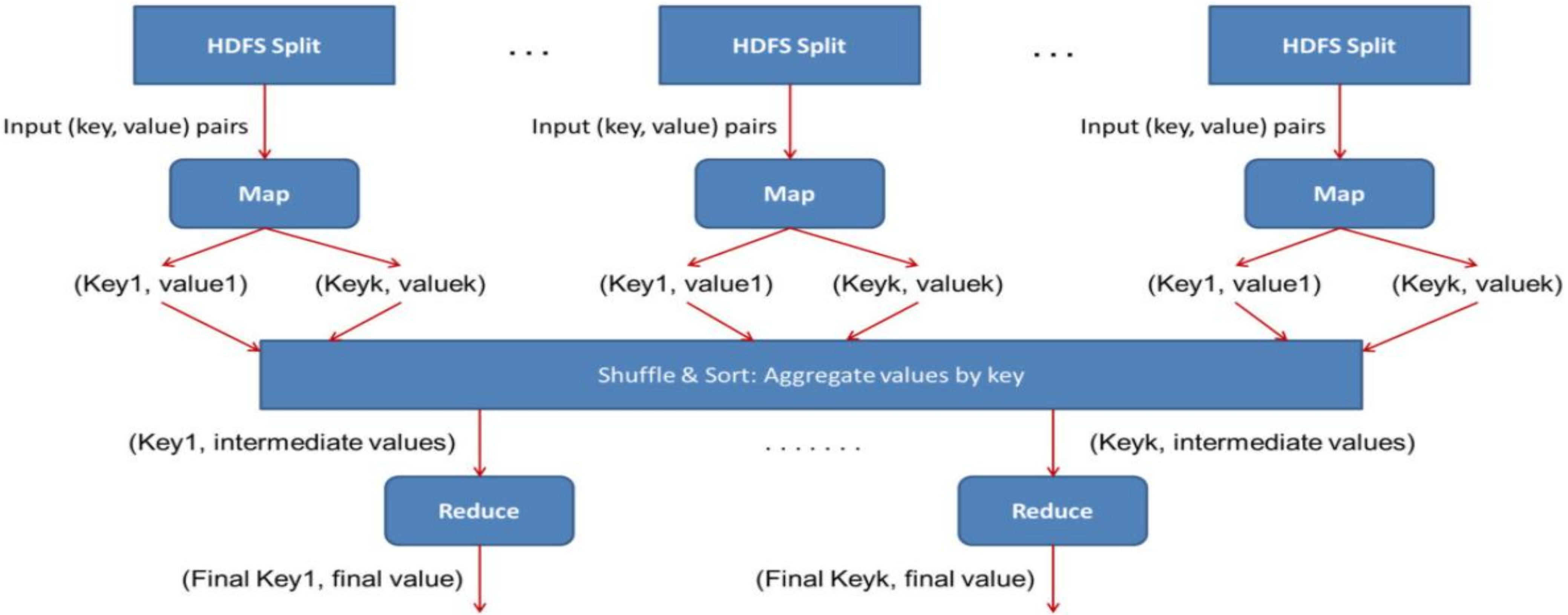


Image ref: https://data-flair.training/blogs/hadoop-architecture/

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()

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)

https://slogix.in/how-to-sort-the-data-using-mapreduce

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 ?

http://www.khalidmammadov.co.uk/finding-min-and-max-fx-rates-for-every-country-using-hadoop-mapreduce/

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 = #

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);
context.write(maxKey, maxW);
}
```

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

Example 3: Custom partitioning

Find max salary by gender and by age group

1201	gopal	45	Male	50000
1202	manisha	40	Female	51000
1203	khaleel	34	Male	30000
1204	prasanth	30	Male	31000
1205	kiran	20	Male	40000
1206	laxmi	25	Female	35000

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 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

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

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

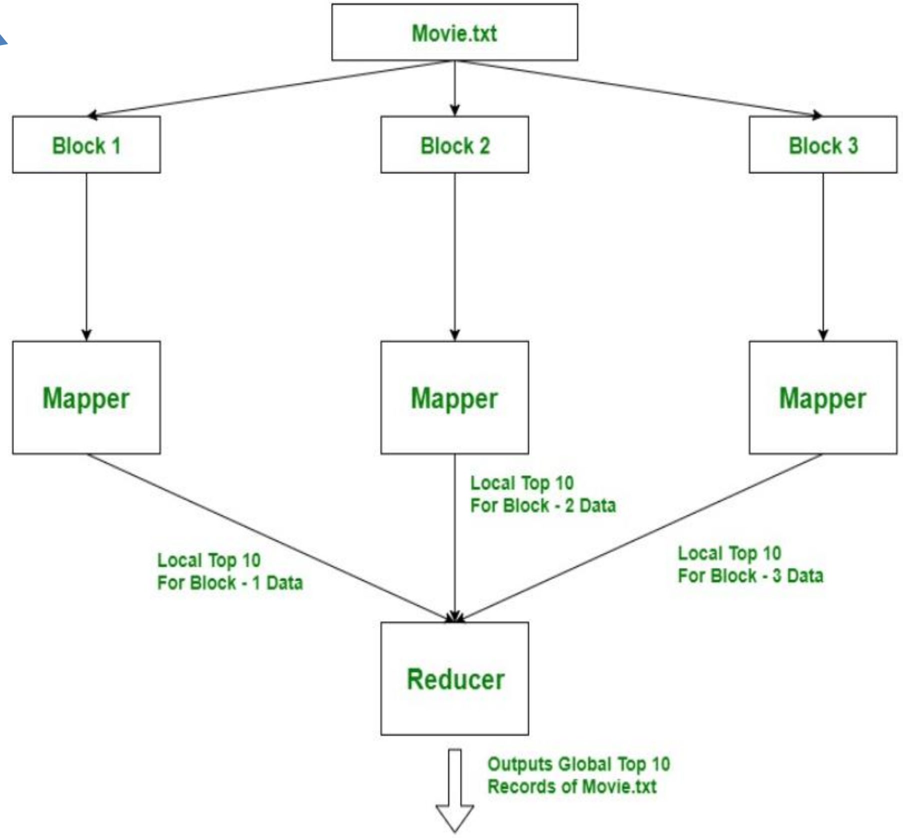
Example 4: Top N keys given key-value pairs

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
```

```
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 4: 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.

Example 5: Use combiners

Find employee with maximum salary

name, salary	name, salary	name, salary	name, salary
satish, 26000	gopal, 50000	satish, 26000	satish, 26000
Krishna, 25000	Krishna, 25000	kiran, 45000	Krishna, 25000
Satishk, 15000	Satishk, 15000	Satishk, 15000	manisha, 45000
Raju, 10000	Raju, 10000	Raju, 10000	Raju, 10000

Maps

<satish, 26000>	<gopal, 50000>	<satish, 26000>	<satish, 26000>
<Krishna, 25000>	<Krishna, 25000>	<kiran, 45000>	<Krishna, 25000>
<Satishk, 15000>	<Satishk, 15000>	<Satishk, 15000>	<manisha, 45000>
<Raju, 10000>	<Raju, 10000>	<Raju, 10000>	<Raju, 10000>

Combiners (1 per Map node as local reducer)

Keep a local variable in combiner class pick top element

<satish, 26000>	<gopal, 50000>	<kiran, 45000>	<manisha, 45000>

Reducer (Set to 1 in Driver)

Keep a local variable in reducer class pick top element

<gopal,50000>

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.
- Use to run python code or any other executable

Hadoop streaming - Example

Count the number of records for each date in the Sales transaction data

hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-2.10.1.jar
-D mapred.reduce.tasks=1 -input /SalesJan2009.csv -output /out **-mapper test.py**
-reducer aggregate -file ./test.py

test.py : extracts mm/dd/yyyy dates from each row of input file and outputs <date, 1>

```
import sys;

def generateLongCountToken(id):
   return "LongValueSum:" + id + "\t" + "1"

def main(argv):
   line = sys.stdin.readline();
   try:
      while line:
         line = line[:-1];
         fields = line.split(“ ");
         print generateLongCountToken(fields[0]);
         line = sys.stdin.readline();
   except "end of file":
      return None
if __name__ == "__main__":
    main(sys.argv)
```

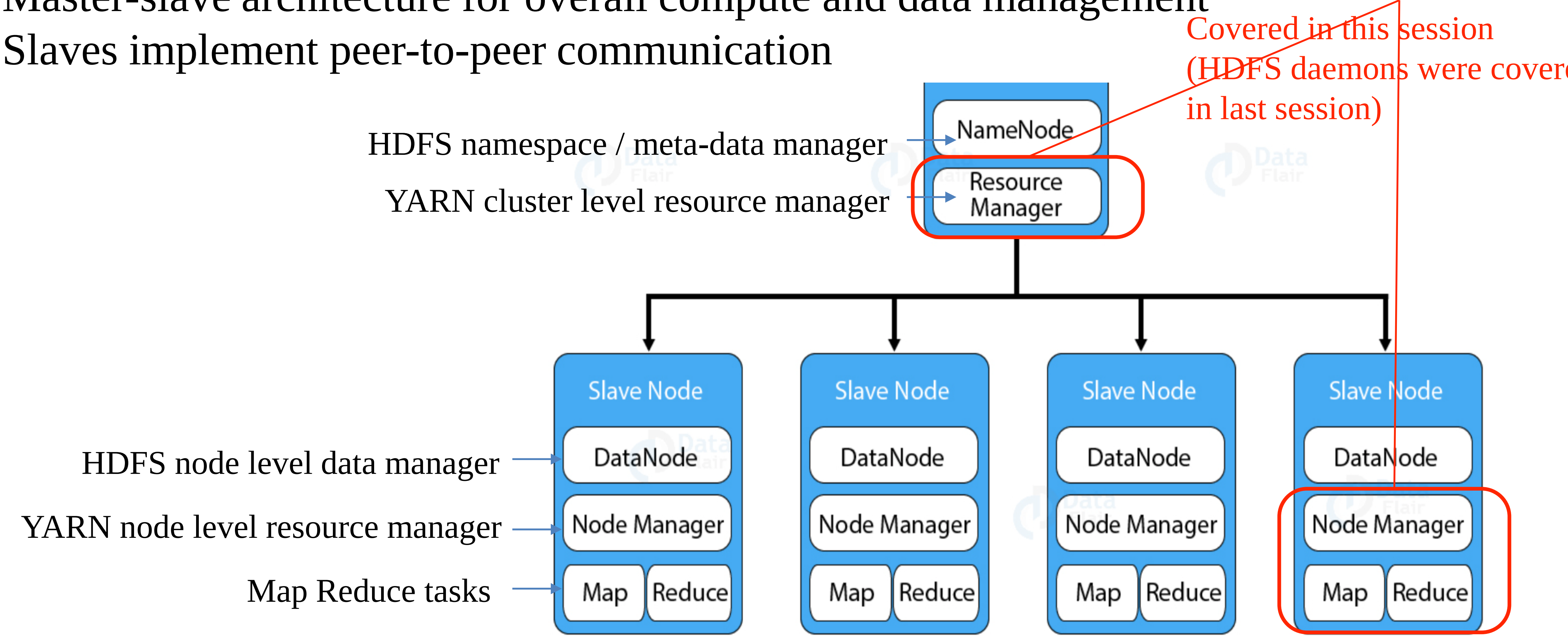
aggregate reducer counts
number of rows with same date

```
1/1/09  27
1/10/09 29
1/11/09 34
1/12/09 44
1/13/09 34
1/14/09 27
1/15/09 32
1/16/09 23
1/17/09 23
1/18/09 42
1/19/09 35
1/2/09  42
1/20/09 28
1/21/09 33
1/22/09 29
1/23/09 31
1/24/09 31
1/25/09 38
```

https://hadoop.apache.org/docs/r1.2.1/streaming.html

Hadoop 2 - Architecture

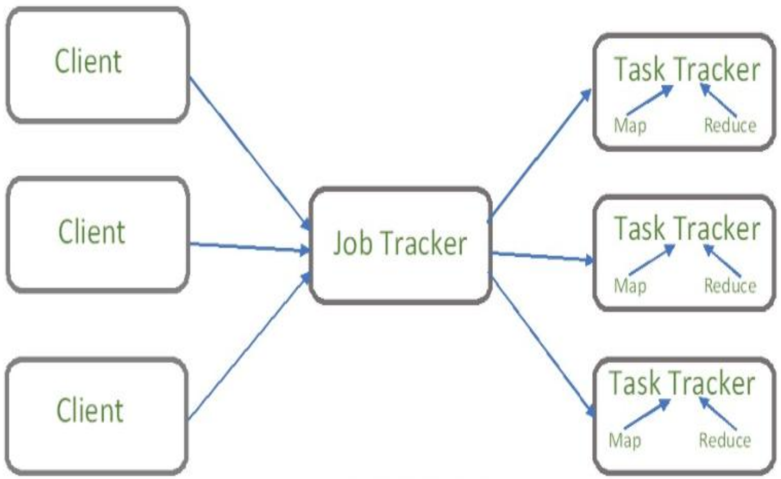
- Master-slave architecture for overall compute and data management
- Slaves implement peer-to-peer communication



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

Changes from Hadoop 1 - Why YARN ?

- In Hadoop 1: MapReduce included 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
- Can run Hadoop 1 jobs on Hadoop2 - backward compatibility



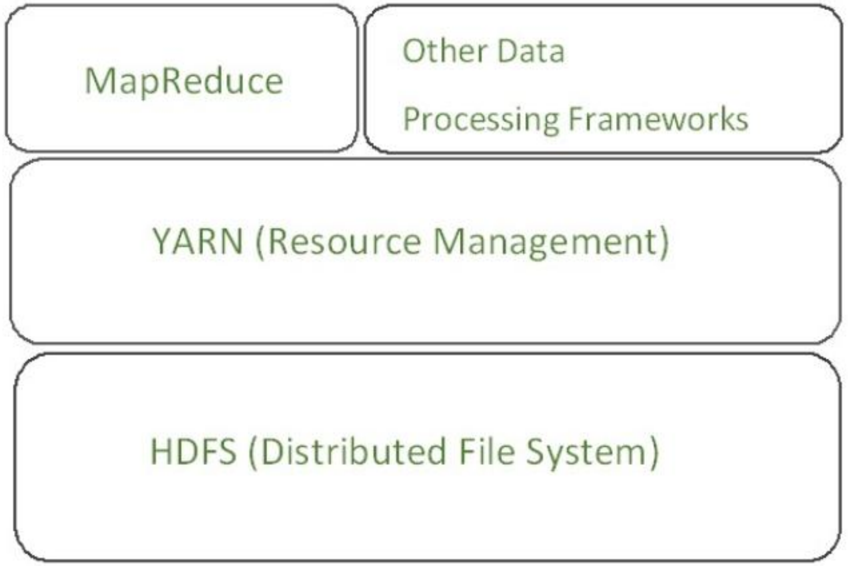
Hadoop 1.0 architecture

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

https://blog.cloudera.com/apache-hadoop-yarn-concepts-and-applications/

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
  - Scale across 1000s of nodes - environments tested with 10K+ nodes
  - Compatibility with MR and other engines
  - Dynamic cluster utilization
  - Multi-tenancy across various processing engines



Hadoop 2.0

YARN components (1)

Client

– Submits MR jobs

Resource manager

– Key role is to schedule resources in the cluster

– Takes request from client and talks to Node Managers for allocation

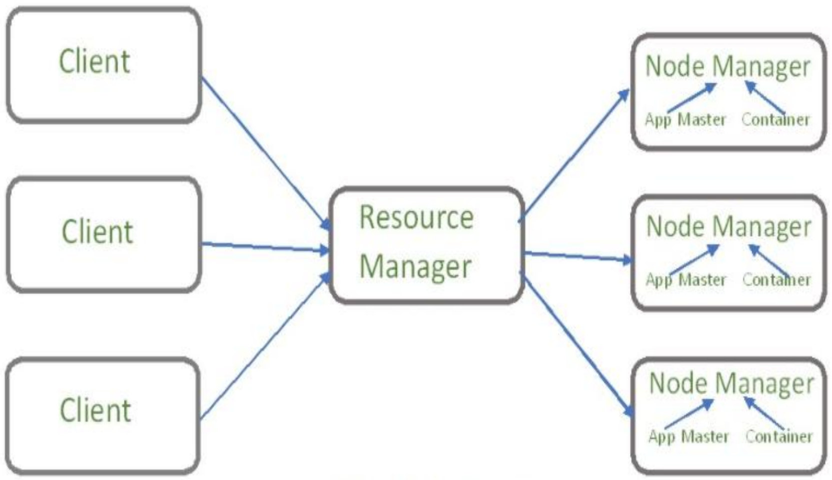
– 2 components:

- Scheduler : key function
- App Manager: Master component of app management
  – Accepts job request and sets up an AppMaster inside a container for each job on a slave.
  – Restarts the AppMaster on failure.
  – Main App specific work is done by AppMaster.

Node Manager

– Takes care of management and workflows on a node.

– Creating, killing containers, monitoring usage, log management



Hadoop Yarn architecture

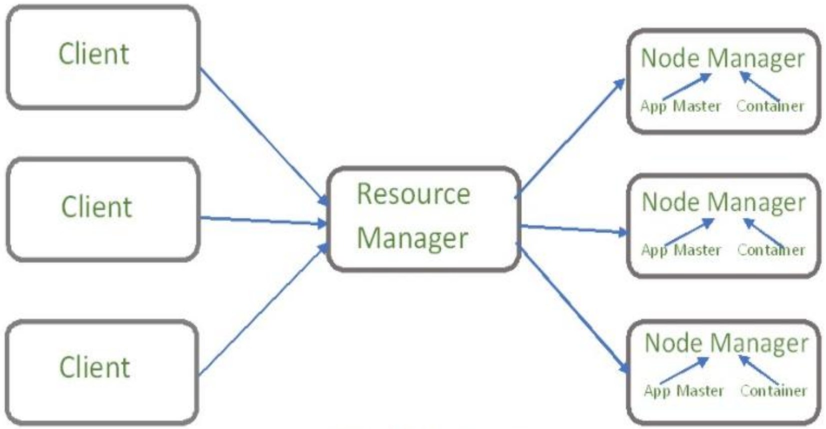
YARN components (2)

AppMaster

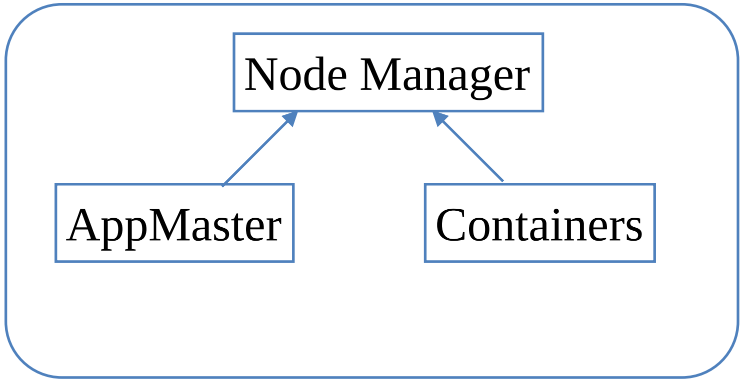
- Negotiates resources from Resource Manager per application for starting containers on nodes
- Sends periodic health status of application containers and tracks progress
- Talks via Node Manager for updates and usage reports to Resource Manager
- Clients can directly talk to AppMaster

Container

- CPU, Memory, Storage resources on a node
- Container Launch Context (CLC) - data structure that contains resource, security tokens, dependencies, environment vars

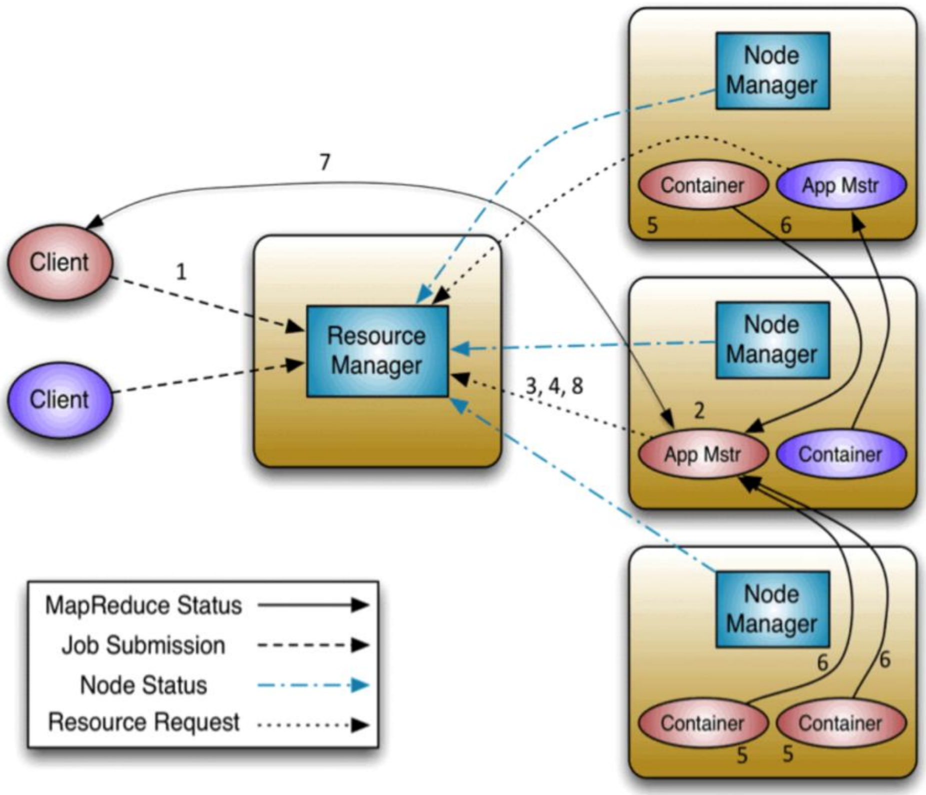


Hadoop Yarn architecture



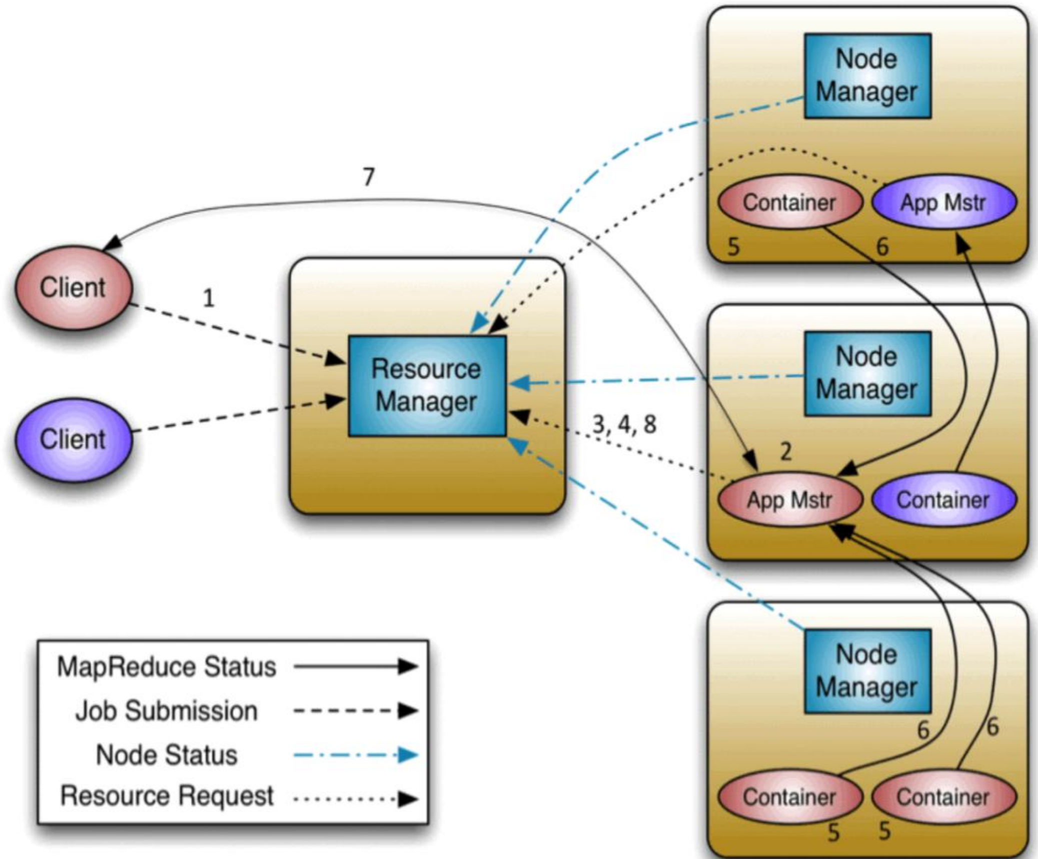
YARN workflow (1)

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.



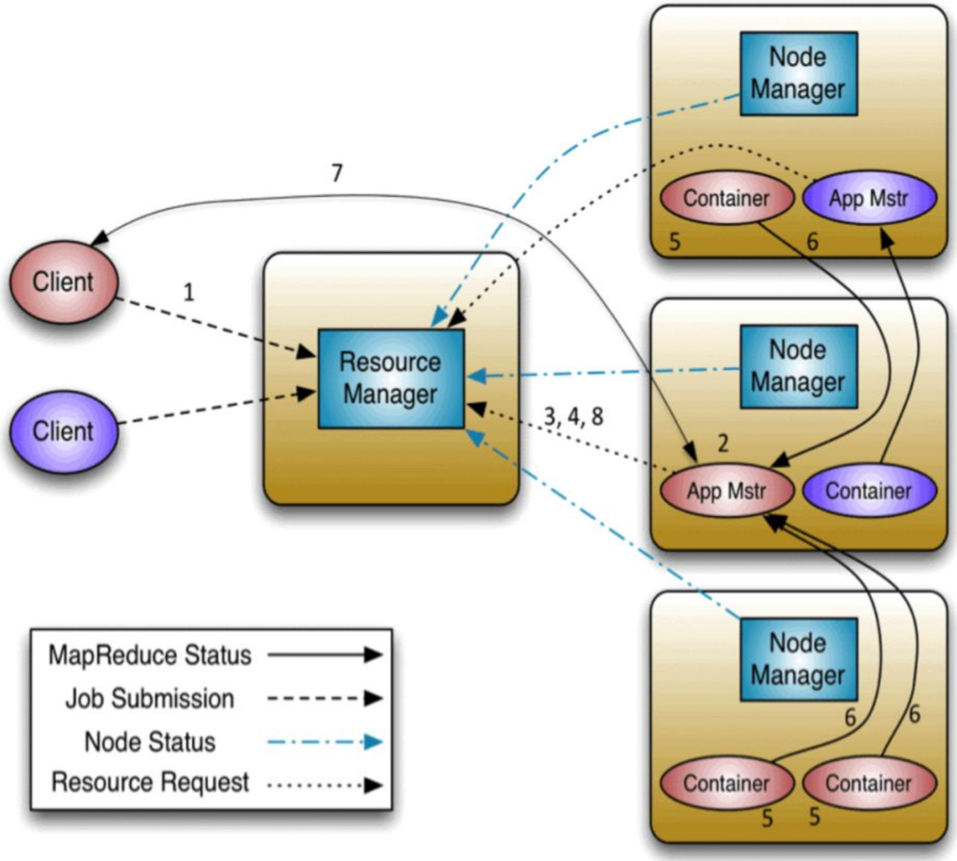
YARN workflow (2)

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 possible to have an AppMaster instance manage multiple applications - it is user specific code

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

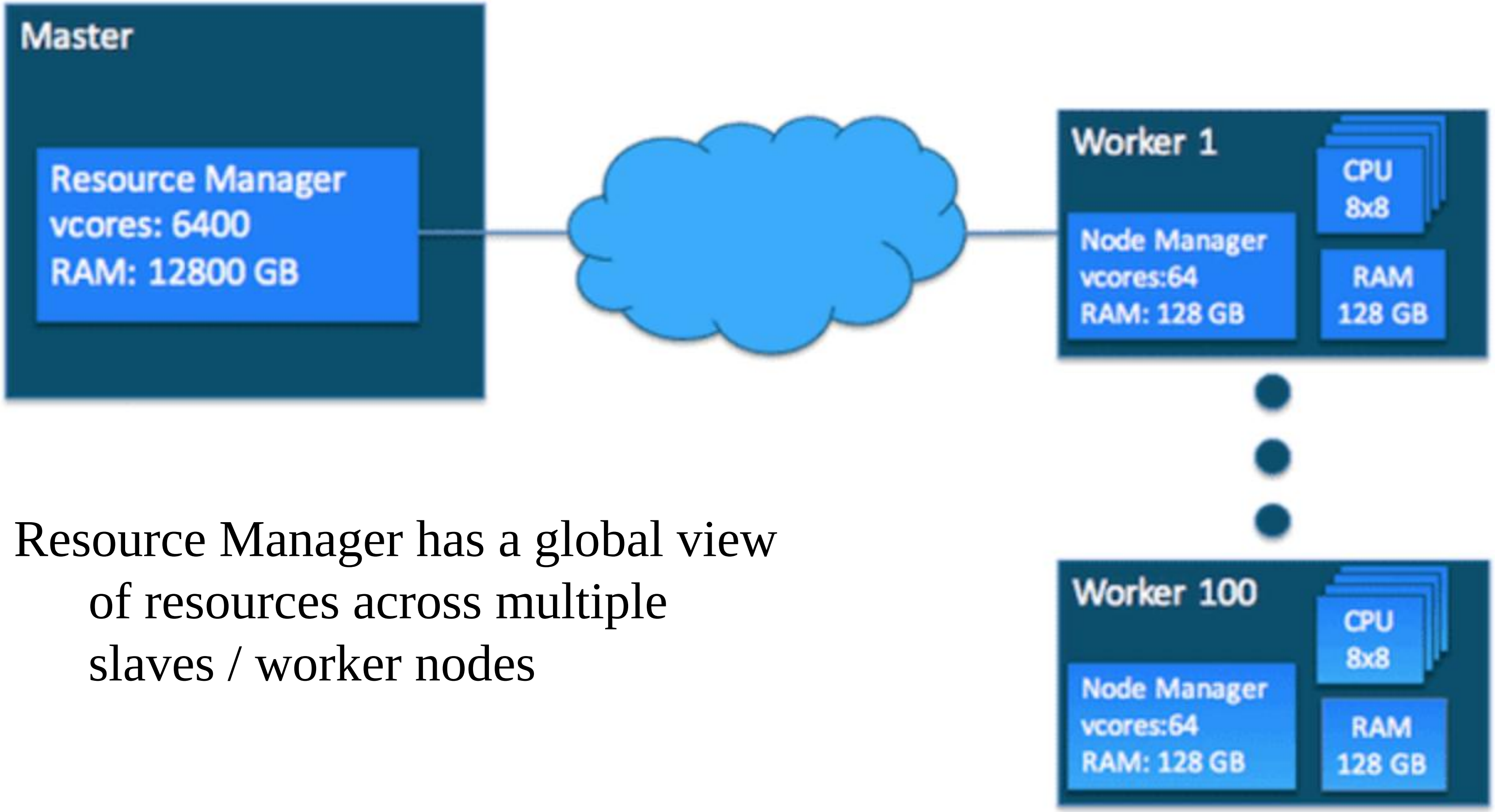
Resource requests

- 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.

Example: Global resource view



Resource Manager has a global view of resources across multiple slaves / worker nodes

https://blog.cloudera.com/untangling-apache-hadoop-yarn-part-1-cluster-and-yarn-basics/

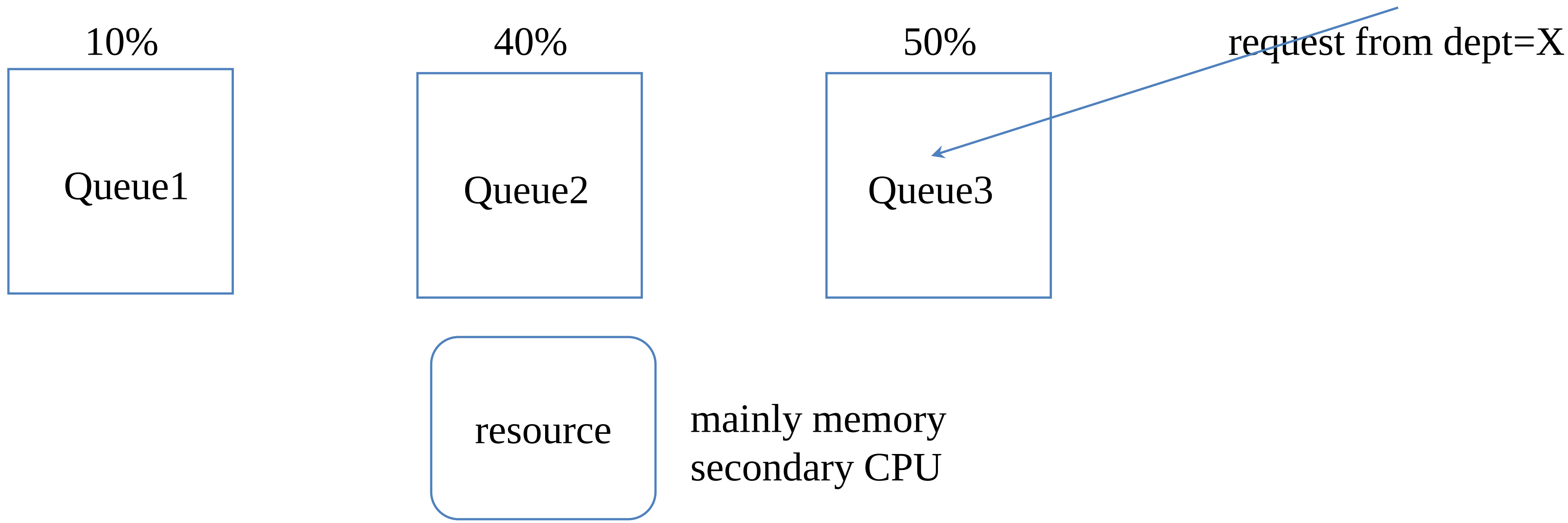
Example: Containers on slaves

- 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

https://blog.cloudera.com/untangling-apache-hadoop-yarn-part-1-cluster-and-yarn-basics/

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.

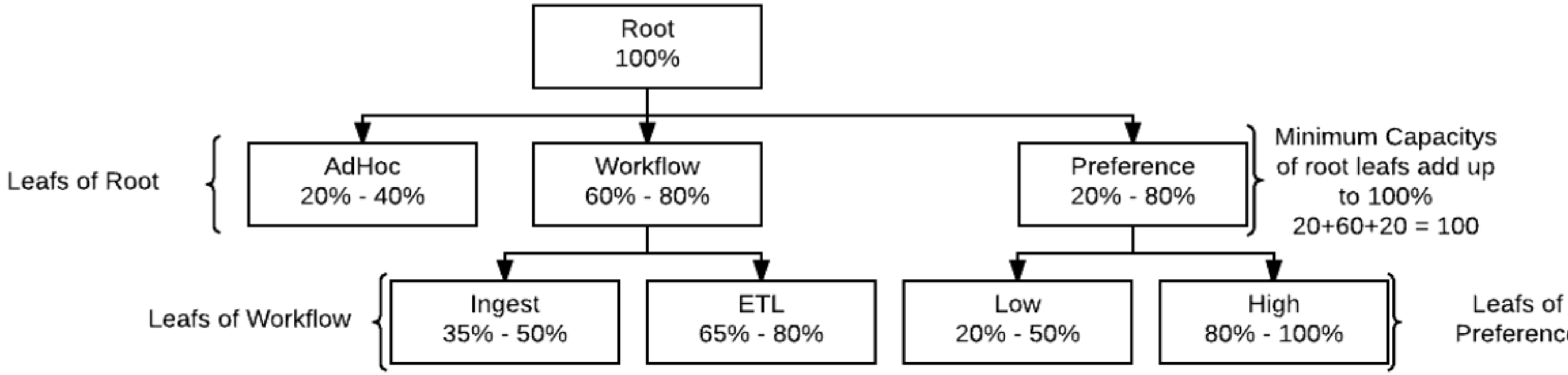


Resource Manager scheduling - Capacity scheduler

- Multiple organisations 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
- 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

Capacity scheduler: Hierarchical queue allocations

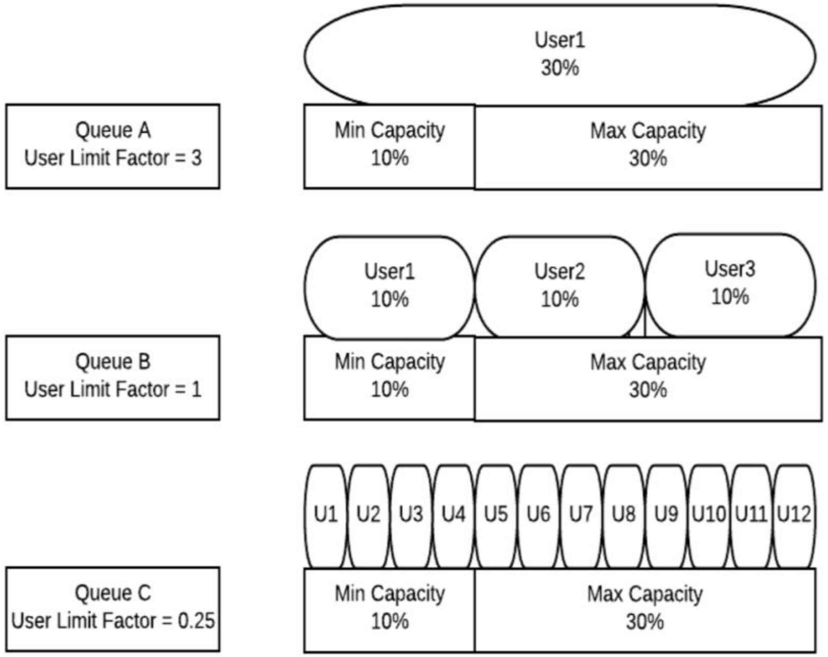
- Min capacity is what needs to be given to a queue even when cluster is at max usage
- Max capacity is an elastic like capacity that allows queues to make use of resources which are n being used to fill minimum capacity demand in other queues.
- e.g. Root->Preference->Low will get min 20% of the 20% min share of Root->Preference



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

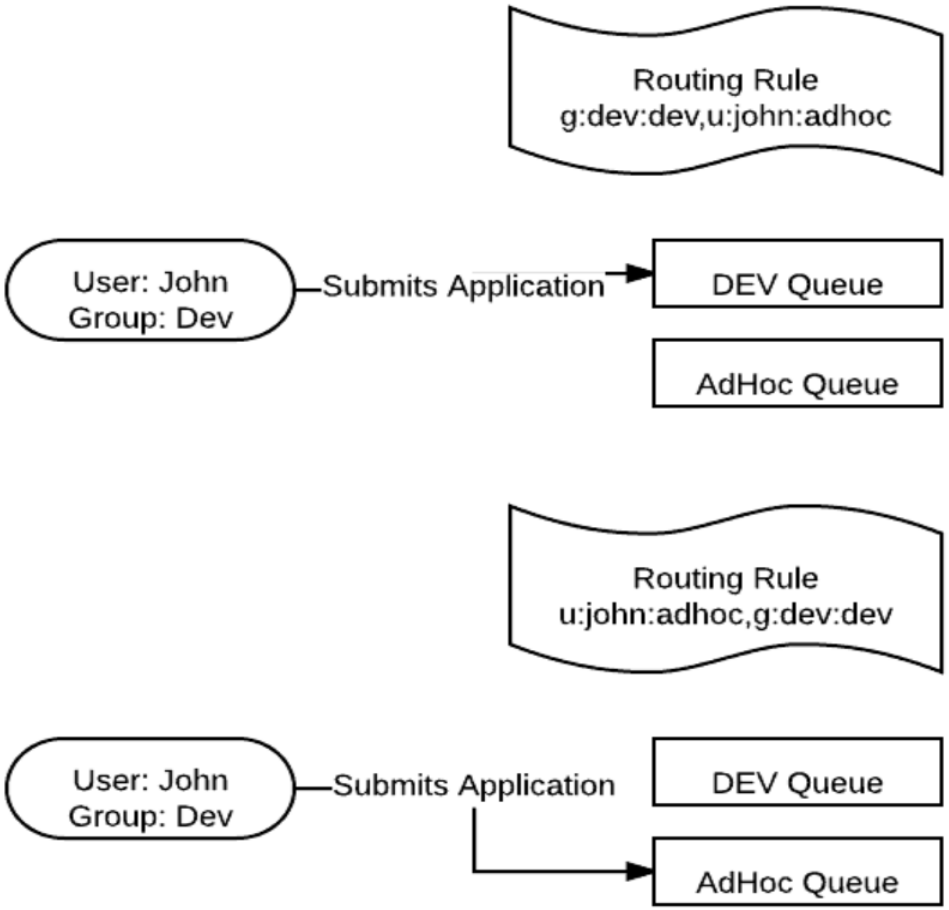
FAIR ordering

- 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

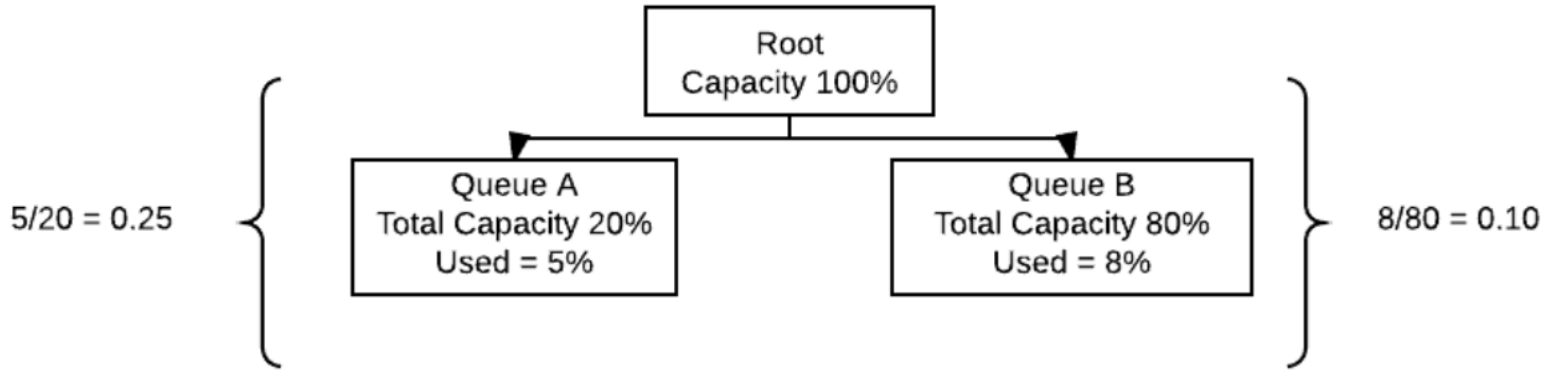
- User and group mapping to queue
- Depends on what order is specified for mapping in routing rules



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

Capacity scheduler : Priority

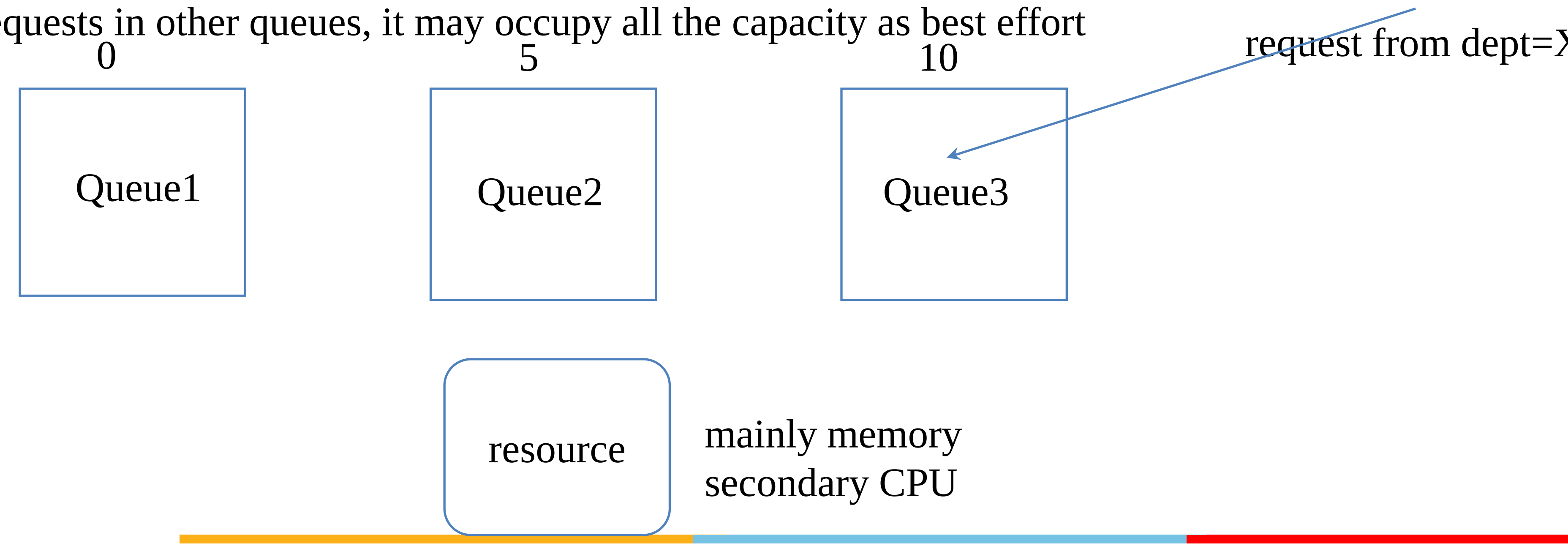
- Influence which queue gets new requests beyond resource utilisation driven
- Queue A relative capacity utilisation is 5 / 20 = 0.25
- Queue B relative capacity utilisation is 8 / 80 = 0.10
- So new requests will go to Queue B
- Increasing priority on Queue A will avoid that



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

Resource Manager scheduling - Fair 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



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

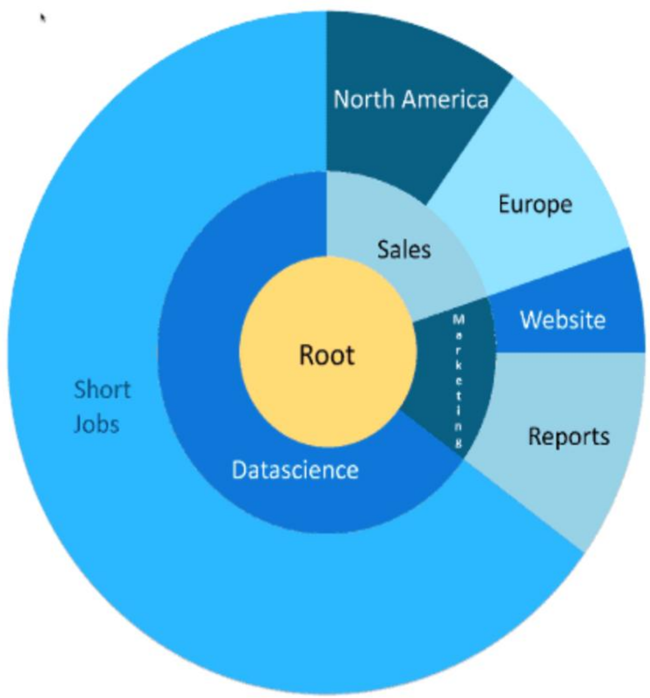
Fair share of a queue is determined by weight (0 means no fair share - just best effort)

- 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

Fair scheduler: Example




```xml
<?xml version="1.0"?>
<allocations>
 <queue name="marketing">
  <weight>3.0</weight>
  <queue name="reports">
   <weight>40.0</weight>
  </queue>
  <queue name="website">
   <weight>20.0</weight>
  </queue>
 </queue>
 <queue name="sales">
  <weight>4.0</weight>
  <queue name="northamerica">
   <weight>30.0</weight>
  </queue>
  <queue name="europe">
   <weight>30.0</weight>
  </queue>
 </queue>
 <queue name="datascience">
  <weight>13.0</weight>
  <queue name="short_jobs">
   <weight>100.0</weight>
  </queue>
  <queue name="best_effort_jobs">
   <weight>0.0</weight>
  </queue>
 </queue>
</allocations>
```

- Weights determine ratio of usage
  - Marketing : 3 (15%)
  - Sales : 4 (20%)
  - Datascience : 13 (65%)
- 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
- 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

https://blog.cloudera.com/untangling-apache-hadoop-yarn-part-3-scheduler-concepts/

Preemption

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 tries not to kill entire application to reclaim resources

- ✓ It tries to kill younger applications - initial stages
- ✓ Or kill reducers (not mappers that have done a lot of work already) because minimum work is lost

Preemption will not do not do anything unless it can fulfil entire allocation request

Preemption is about only reclaiming min and not max allocation

Preemption in latest editions of scheduler do not work across queues. Within a queue one has to work with User limit factor or FIFO/FAIR ordering policies.