



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DSECL ZG 522: Big Data Systems

Session 7: Hadoop MapReduce and YARN

Dr. Anindya Neogi

Associate Professor

anindya.neogi@pilani.bits-pilani.ac.in

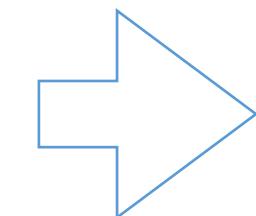
Topics for today

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



Sample MapReduce program (recap - session 5)

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	Mastercard	Carolina	Basildon	England	United Kingdom	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 States	01-02-2009 04:42	01-02-2009 07:49	39.195	-94.6819	
4	01-02-2009 13:08	Product1	1200	Mastercard	Federica	Astoria	OR	United States	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	HeAL	United States	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 States	9/24/08 15:19	01-04-2009 13:04	39.79	-75.2381	
8	01-04-2009 20:11	Product1	1200	Mastercard	Fleur	Peoria	IL	United States	01-03-2009 09:38	01-04-2009 19:45	40.69361	-89.5889	
9	01-02-2009 20:09	Product1	1200	Mastercard	adam	Martin	TN	United States	01-02-2009 17:43	01-04-2009 20:01	36.34333	-88.8503	
10	01-04-2009 13:17	Product1	1200	Mastercard	Renee	Eli	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-France	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 States	01-05-2009 02:23	01-05-2009 04:59	40.71417	-74.0064	
13	01-05-2009 05:39	Product1	1200	Amex	Heidi	Eindhoven	Noord-Brabant	Netherlands	01-05-2009 04:55	01-05-2009 08:15	51.45	5.466667	
14	01-02-2009 09:16	Product1	1200	Mastercard	Sean	Shavano	PTX	United States	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 States	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 States	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	
18	01-05-2009 11:27	Product1	1200	Visa	Isaac	Ottawa	Ontario	Canada	01-05-2009 00:25	01-05-2009 10:24	45.41557	76.7	



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>

Running and checking status (recap)

```
[root@centos-s-4vcpu-8gb-blr1-01 source]# ls -l
total 148
-rw-r--r--. 1 root root    44 Apr 18 01:11 Manifest.txt
-rw-r--r--. 1 root root 2966 Apr 18 01:12 ProductSalePerCountry.jar
drwxr-xr-x. 2 root root    96 Apr 19 00:57 SalesCountry
-rw-r--r--. 1 root root 1529 Apr 18 00:57 SalesCountryDriver.java
-rw-r--r--. 1 root root   746 Apr 18 00:56 SalesCountryReducer.java
-rw-r--r--. 1 root root 123637 Jun  6 16:56 SalesJan2009.csv
-rw-r--r--. 1 root root   661 Apr 18 00:56 SalesMapper.java
drwxr-xr-x. 3 root root   157 Jun  6 16:53 units
-rw-r--r--. 1 root root  219 Apr 18 21:10 units.csv
[[root@centos-s-4vcpu-8gb-blr1-01 source]# hadoop jar ProductSalePerCountry.jar /SalesJan2009.csv /output
21/06/06 17:27:37 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
21/06/06 17:27:38 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/06/06 17:27:39 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/06/06 17:27:39 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute
remedy this.
21/06/06 17:27:40 INFO mapred.FileInputFormat: Total input files to process : 1
21/06/06 17:27:40 INFO mapreduce.JobSubmitter: number of splits:2
21/06/06 17:27:40 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1622978356181_0002
21/06/06 17:27:41 INFO conf.Configuration: resource-types.xml not found
21/06/06 17:27:41 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
21/06/06 17:27:41 INFO resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = COUNTABLE
21/06/06 17:27:41 INFO resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTABLE
21/06/06 17:27:41 INFO impl.YarnClientImpl: Submitted application application_1622978356181_0002
21/06/06 17:27:41 INFO mapreduce.Job: The url to track the job: http://centos-s-4vcpu-8gb-blr1-01:8088/proxy/application_1622978356181_0002/
21/06/06 17:27:41 INFO mapreduce.Job: Running job: job_1622978356181_0002
21/06/06 17:27:49 INFO mapreduce.Job: Job job_1622978356181_0002 running in uber mode : false
21/06/06 17:27:49 INFO mapreduce.Job: map 0% reduce 0%
21/06/06 17:28:03 INFO mapreduce.Job: map 100% reduce 0%
```

```
[root@centos-s-4vcpu-8gb-blr1-01 source]# jps
7744 SecondaryNameNode
8357 ResourceManager
8581 NodeManager
13541 Jps
7238 DataNode
10428 MRAppMaster
6910 NameNode
```



MapReduce Application application_1622978356181_0002

Logged in as: dr.who

Cluster

Application

About Jobs

Tools

Active Jobs

Job ID	Name	State	Map Progress	Maps Total	Maps Completed	Reduce Progress	Reduces Total	Reduces Completed
job_1622978356181_0002	SalePerCountry	RUNNING	2	2		1	0	0

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

MapReduce stats (recap)

File System Counters

```
FILE: Number of bytes read=17747
FILE: Number of bytes written=660936
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=127535
HDFS: Number of bytes written=661
HDFS: Number of read operations=9
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
```

Job Counters

```
Launched map tasks=2
Launched reduce tasks=1
Data-local map tasks=2
Total time spent by all maps in occupied slots (ms)=14658
Total time spent by all reduces in occupied slots (ms)=7011
Total time spent by all map tasks (ms)=14658
Total time spent by all reduce tasks (ms)=7011
Total vcore-milliseconds taken by all map tasks=14658
Total vcore-milliseconds taken by all reduce tasks=7011
Total megabyte-milliseconds taken by all map tasks=15009792
Total megabyte-milliseconds taken by all reduce tasks=7179264
```

Map-Reduce Framework

```
Map input records=999
Map output records=999
Map output bytes=15743
Map output materialized bytes=17753
Input split bytes=180
Combine input records=0
Combine output records=0
Reduce input groups=58
Reduce shuffle bytes=17753
Reduce input records=999
Reduce output records=58
Spilled Records=1998
Shuffled Maps =2
Failed Shuffles=0
Merged Map outputs=2
GC time elapsed (ms)=585
CPU time spent (ms)=2510
Physical memory (bytes) snapshot=772923392
Virtual memory (bytes) snapshot=6393163776
Total committed heap usage (bytes)=527433728
```

Shuffle Errors

```
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
```

File Input Format Counters

```
Bytes Read=127355
```

File Output Format Counters

```
Bytes Written=661
```

Unravelling a MapReduce program - Driver (1)

```
package SalesCountry;  
  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.*;  
import org.apache.hadoop.mapred.*;  
  
public class SalesCountryDriver {  
  
    public static void main(String[] args) {  
        JobClient my_client = new JobClient();  
        // Create a configuration object for the job  
        JobConf job_conf = new  
        JobConf(SalesCountryDriver.class);  
  
        // Set a name of the Job  
        job_conf.setJobName("SalePerCountry");  
  
        // Specify data type of output key and value  
        job_conf.setOutputKeyClass(Text.class);  
        job_conf.setOutputValueClass(IntWritable.class);  
        . . .
```

package name of jar

hadoop libs

driver class that contains main()

Hadoop job with driver class
with SalesPerCountry as the application
name

Output data types defined based on
existing hadoop classes

Unravelling a MapReduce program - Driver (2)

...

```
// Specify names of Mapper and Reducer Class  
job_conf.setMapperClass(SalesCountry.SalesMapper.class);  
job_conf.setReducerClass(SalesCountry.SalesCountryReducer.class);
```

Mapper and Reducer classes in the jar pkg

```
// Set input and output directories  
// arg[0] = name of input directory on HDFS, and  
// arg[1] = name of output directory to be created  
// to store the output file.  
FileInputFormat.setInputPaths(job_conf, new Path(args[0]));  
FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));
```

Paths to read input and send output

```
my_client.setConf(job_conf);  
try {  
    // Run the job  
    JobClient.runJob(job_conf);  
} catch (Exception e) {  
    e.printStackTrace();  
}  
}
```

Send job for execution

Unravelling a MapReduce program - Mapper

```
package SalesCountry;  
import java.io.IOException;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapred.*;
```

```
public class SalesMapper extends MapReduceBase implements Mapper <LongWritable, Text,  
Text, IntWritable> {  
    private final static IntWritable one = new IntWritable(1);  
    // can add some data structure here for across map() instances  
    public void map(LongWritable key, Text value, OutputCollector <Text, IntWritable>  
output, Reporter reporter) throws IOException {  
  
        String valueString = value.toString();  
        String[] SingleCountryData = valueString.split(",");  
        output.collect(new Text(SingleCountryData[7]), one);  
    }  
}
```

← app jar package

← hadoop libs

map function w inputs

- **one or more <key, value> pairs**
- **output collector**
- ...

Map logic

<India, 1> <USA, 1> <India, 1>

Unravelling a MapReduce program - Reducer

```
package SalesCountry;
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;  
  
public class SalesCountryReducer extends MapReduceBase implements Reducer<Text, IntWritable,
Text, IntWritable> {
    // can add some data structure here for across reduce() instances
    public void reduce(Text t_key, Iterator<IntWritable> values,
OutputCollector<Text,IntWritable> output, Reporter reporter) throws IOException {
        Text key = t_key;
        int frequencyForCountry = 0;
        while (values.hasNext()) {
            // replace type of value with the actual type of our value
            IntWritable value = (IntWritable) values.next();
            frequencyForCountry += value.get();
        }
        output.collect(key, new IntWritable(frequencyForCountry));
    }
}
```

#reducers = 2

reduce() will be called as many times as #keys

Pkg and includes

reduce function w inputs

- key and value list,
e.g. <India, {1,1,1,1}>
- output collector
- ...

Reduce logic

- calculate frequency

For each reduce task, 1 output file created
Can control #reducer in driver

Topics for today

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



Map

1. RecordReader

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

2. Map

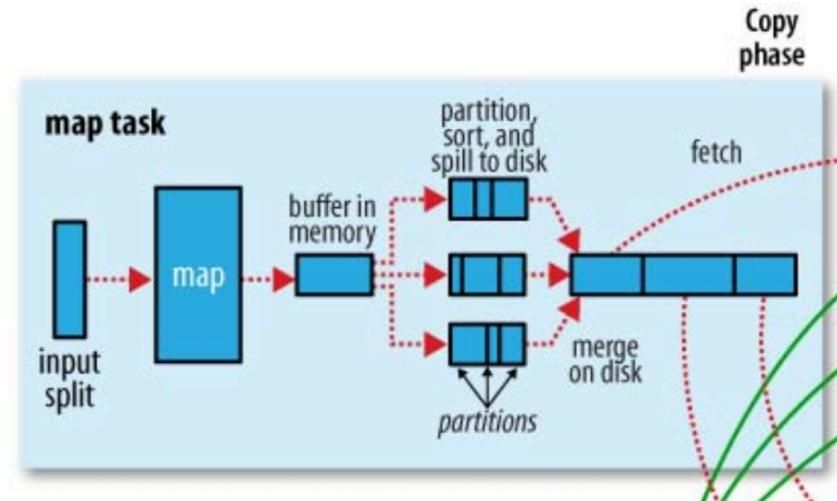
- ✓ **User defined function** to create key, value pair from input key, value pair

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)



Reduce

1. Shuffle and Sort

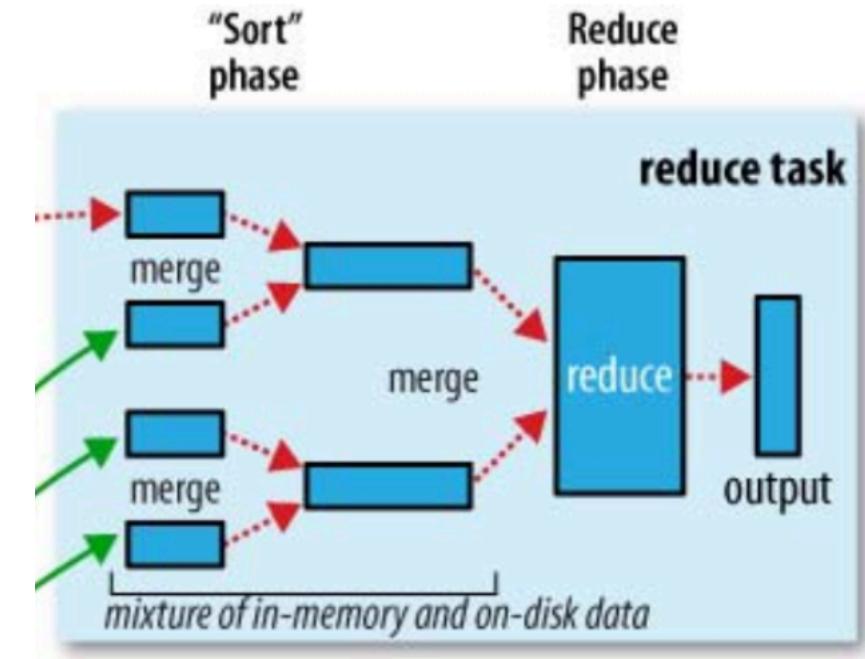
- ✓ Shuffle gets data from map node to reduce node or reduce task to happen where the relevant post-map data partition is
- ✓ Shuffling can start before Map is entirely complete
- ✓ Data is merge-sorted by key coming in from multiple maps

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

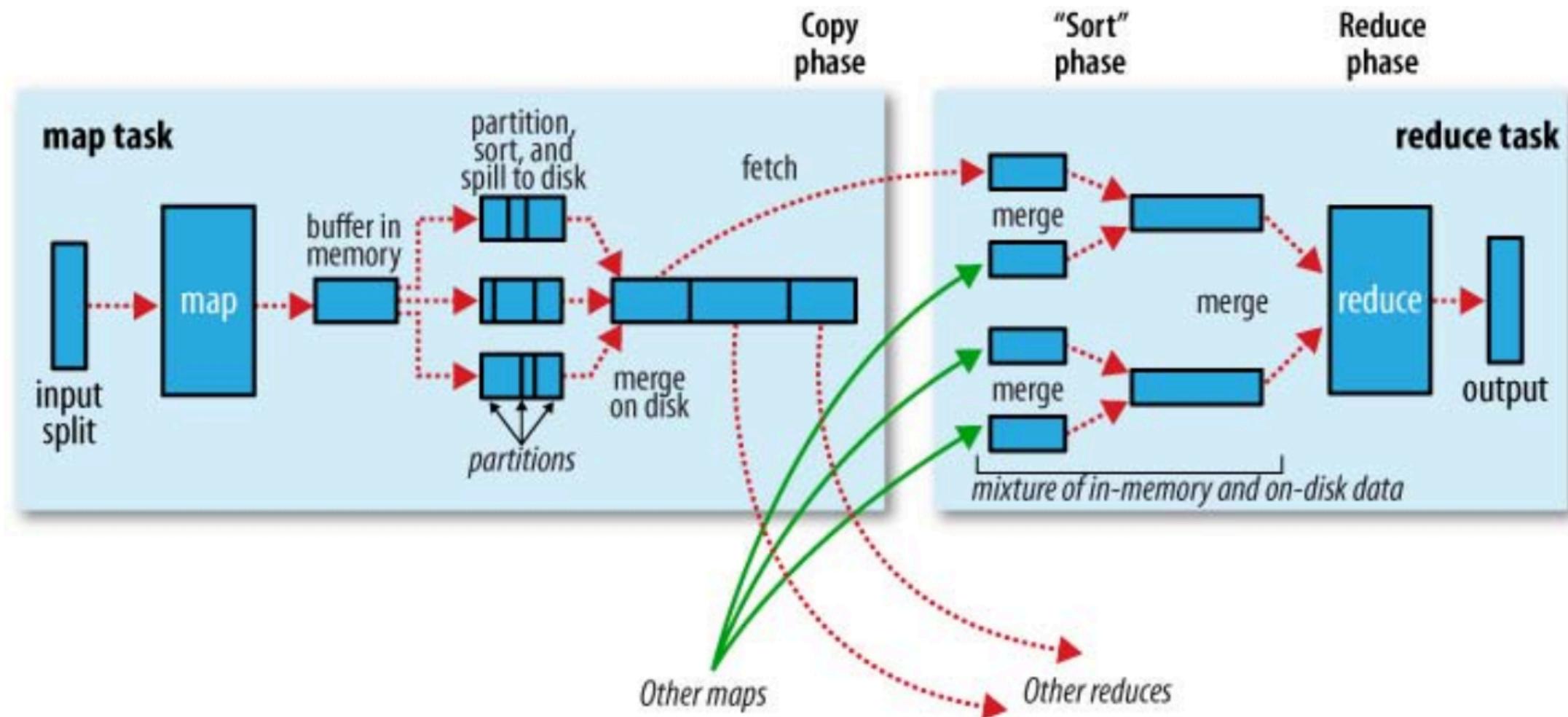
- ✓ Writes final output of reducer with separator of key, value and separator between pairs



MapReduce Programming Architecture

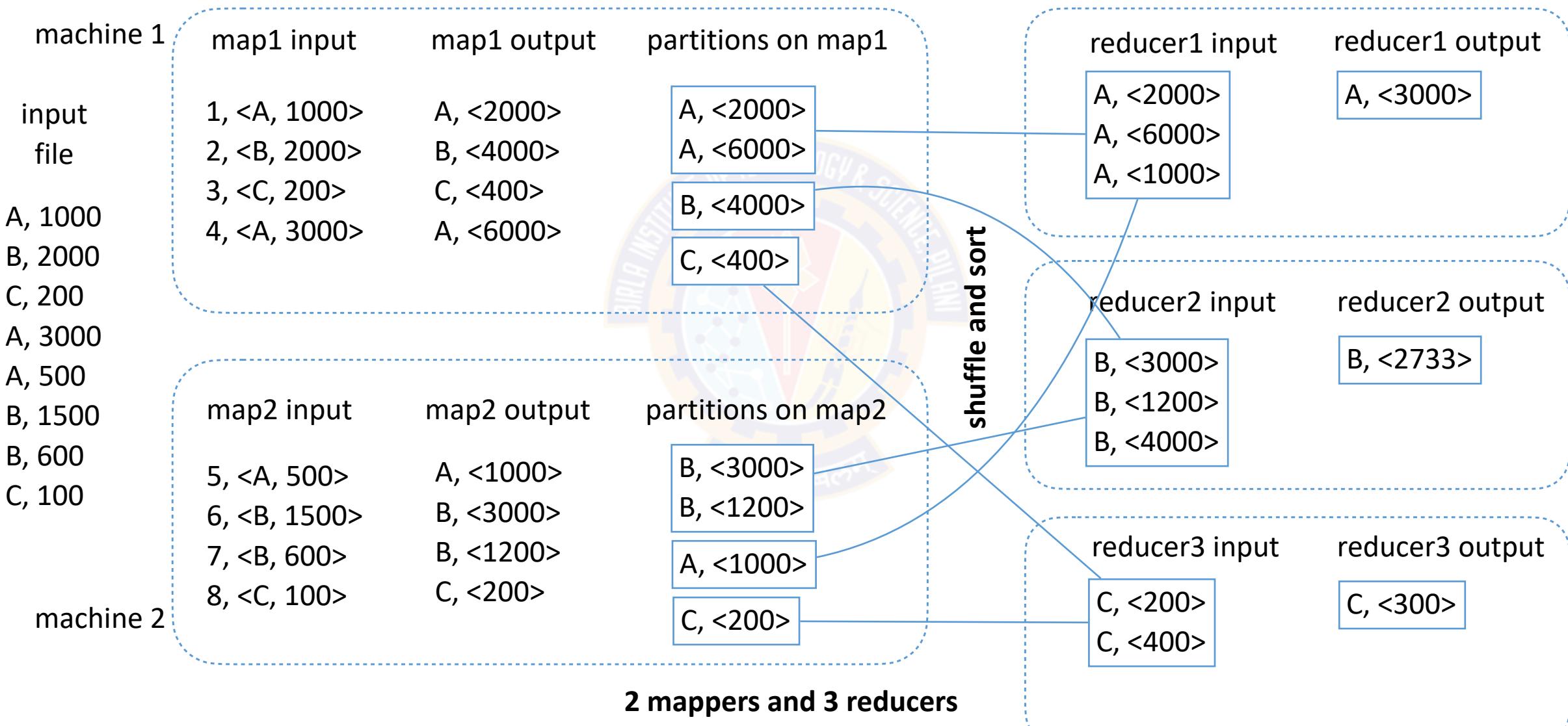
1. Input dataset is **split** into multiple pieces of data (several small sets)
2. Framework creates a **master** and several **worker** processes and executes the worker processes remotely
3. Several **Map tasks** work simultaneously and read pieces of data that were assigned to each map. Map worker 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.

MapReduce flow



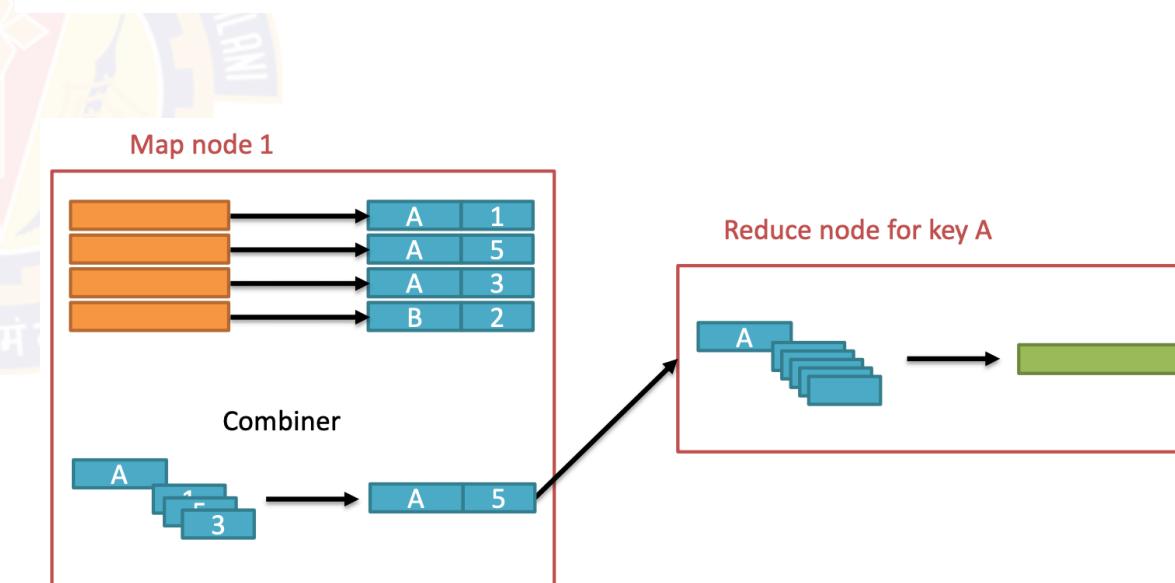
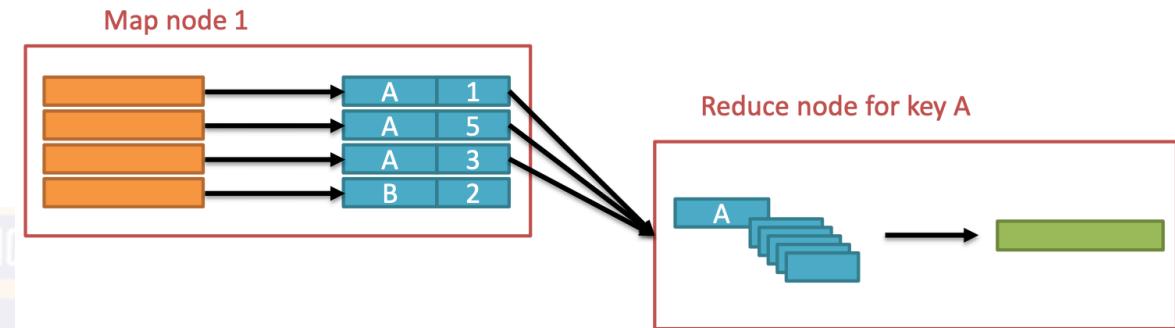
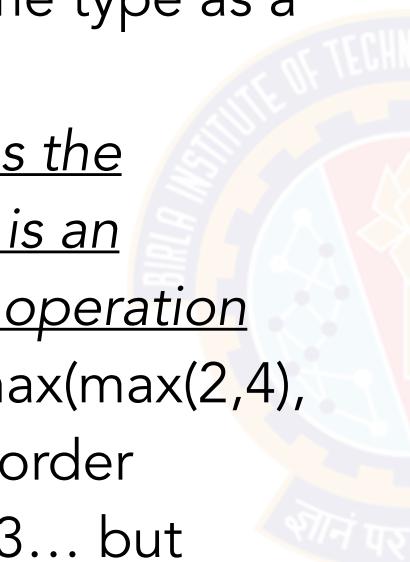
<https://stackoverflow.com/questions/22141631/what-is-the-purpose-of-shuffling-and-sorting-phase-in-the-reducer-in-map-reduce>

Map Reduce Example (input $\langle k, v \rangle$ find $\text{avg}(2v)$ for each k)



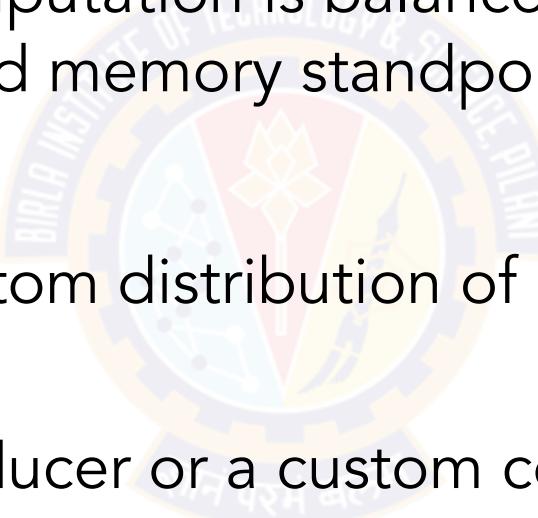
A word about Combiner

- Combiners can optimise the reduce by pre-processing on each node to compress data but output has to be same type as a map
- The reducer can also be set as the combiner class only if reduce is an associative and commutative operation
 - ✓ $\max(1,2, \max(3,4,5)) = \max(\max(2,4), \max(1,5,3))$ or any other order
 - ✓ $\text{avg}(1,2,\text{avg}(3,4,5)) = 2.33\dots$ but $\text{avg}(\text{avg}(2,4),\text{avg}(1,5,3)) = 3$
- 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

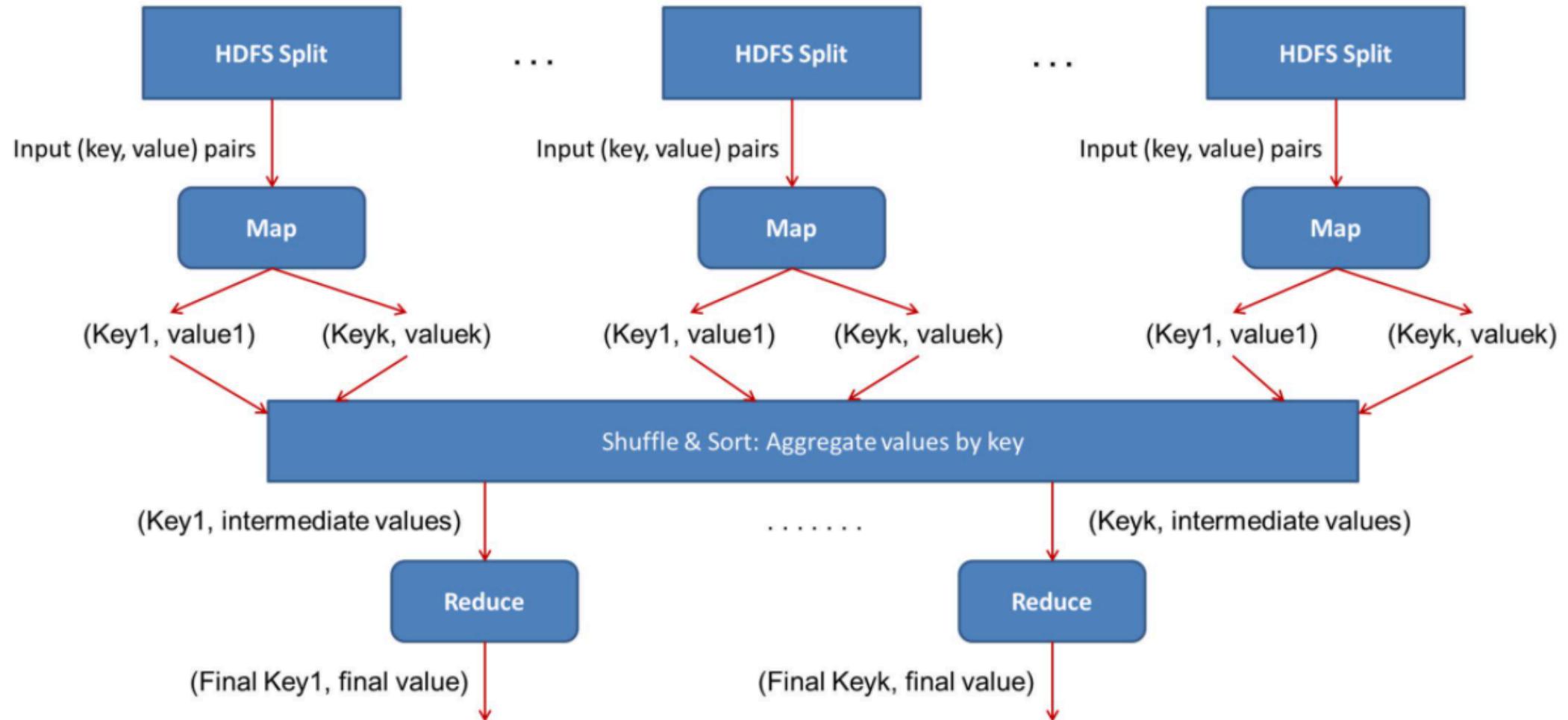
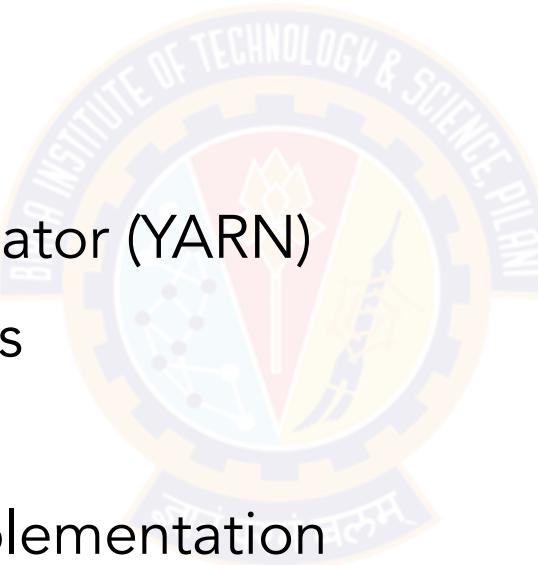


Image ref: <https://data-flair.training/blogs/hadoop-architecture/>

Topics for today

- Hadoop MapReduce
 - ✓ Walkthrough a MapReduce program
 - ✓ MapReduce runtime
 - ✓ **More examples**
- 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)

<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

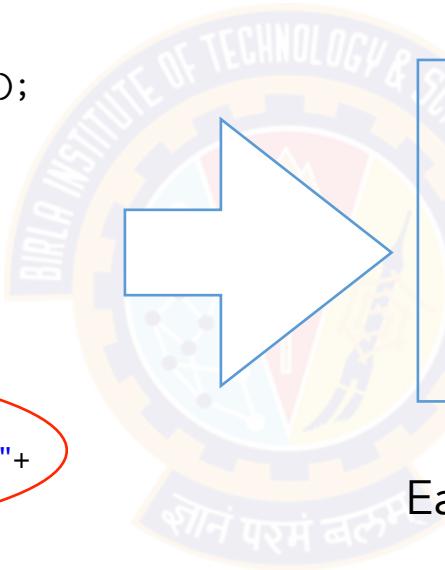


1971 Australia MIN	0.8412
1971 Australia MAX	0.899
1971 Austria MIN	23.638
1971 Austria MAX	25.873
1971 Belgium MIN	45.49
1971 Belgium MAX	49.73
1971 Canada MIN	0.9933
1971 Canada MAX	1.0248
1971 Denmark MIN	7.0665
1971 Denmark MAX	7.5067

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 ?

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>
```

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

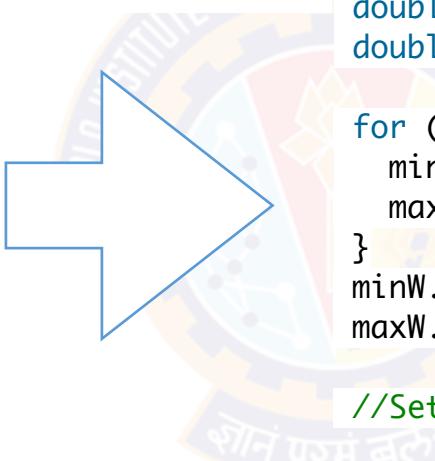
```
<key=K2, value=rate>  
<key=K2, value=rate>
```

...

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

...

#partitions = #keys = #reduce calls



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);  
}
```

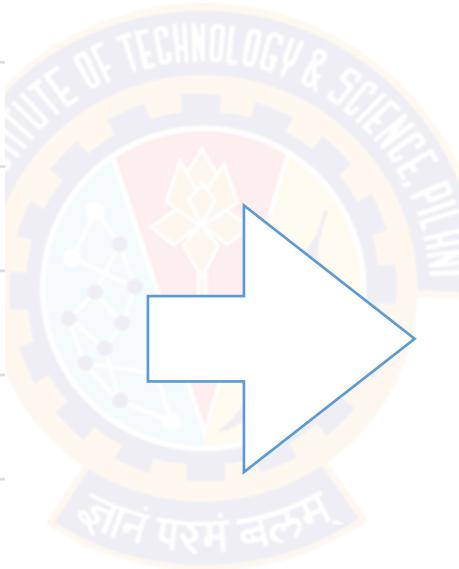
set #reducers = 1
if all results needed in
single node else
collect later and let
partitions run in
parallel

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

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

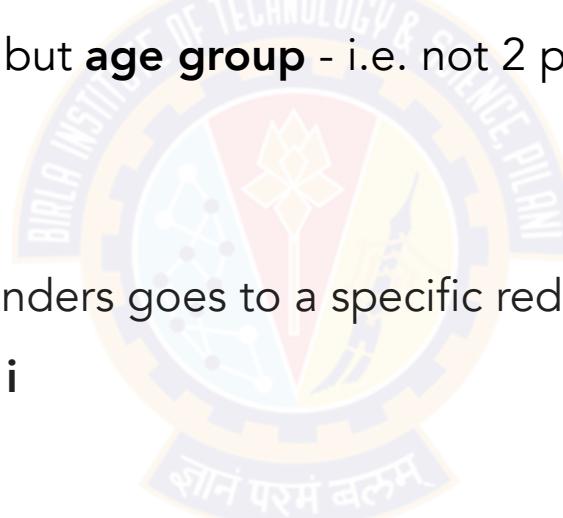
age group 2

Output in Part-00002

Female	51000
Male	50000

age group 3

Example 3: Custom partitioner instead of composite-key



Example 4: Top N keys given key-value pairs

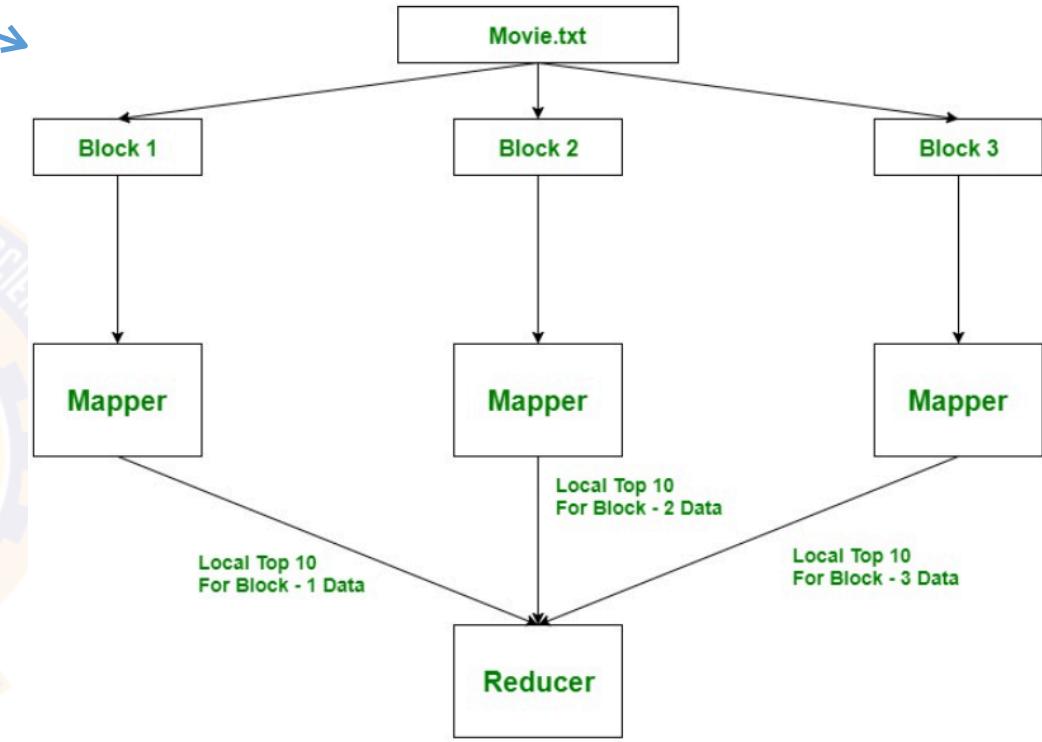
Find the Top 10 movies given <movie_name,#views>

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](#)

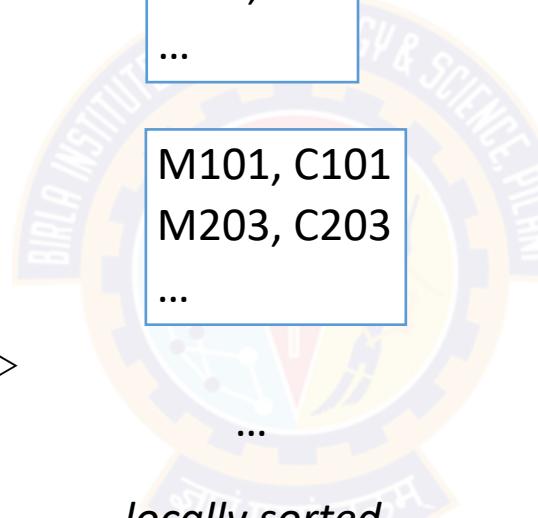
Example 4: Top N keys given key-value pairs

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

locally sorted
top N lists

```
Reducer class {
    TreeMap tmap

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

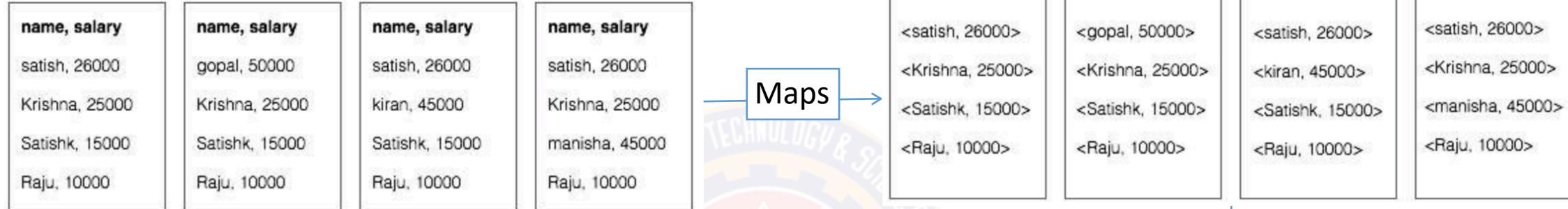
    for each Reduce :
        get all tmap entries <key, value>
        context.write(key, value)
    }
}
```

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

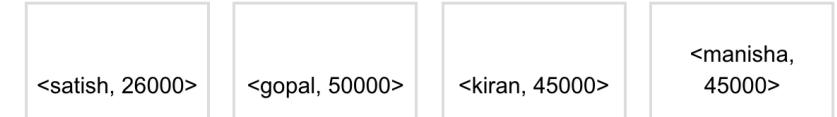
Find employee with maximum salary



Keep a local variable
in combiner class pick top element

Keep a local variable
in reducer class pick top element

Combiners (1 per Map node as local reducer)



Reducer (Set to 1 in Driver)

<gopal,50000>

Example 6

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

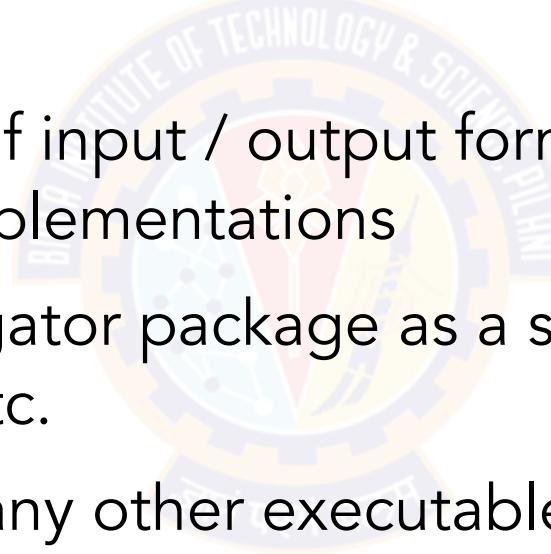
Advanced examples

- 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.
- Use to run python code or any other executable



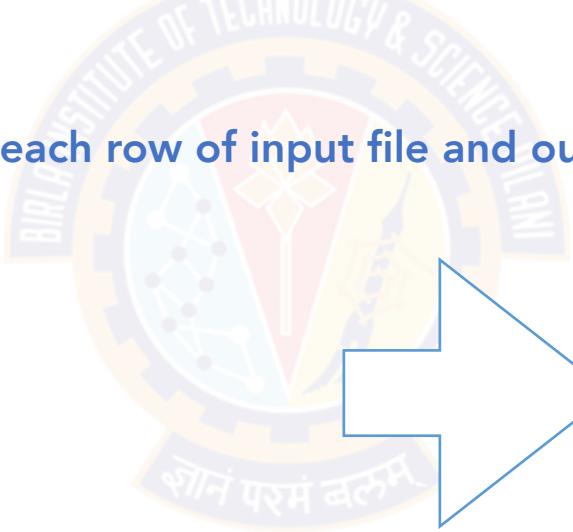
Hadoop streaming - Example

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)
```

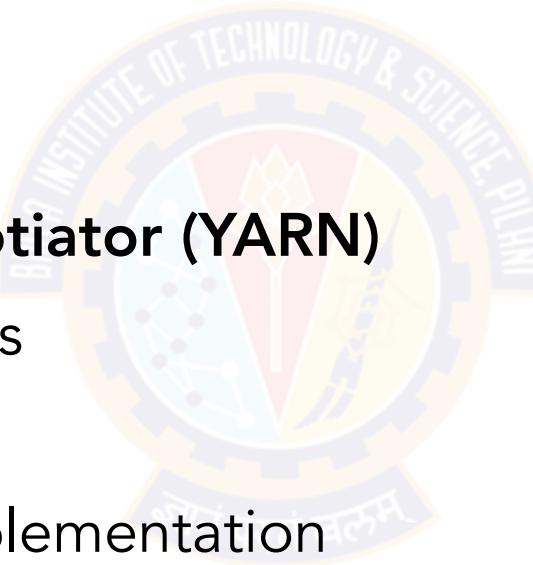

aggregate reducer counts
number of rows with same date

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
1/26/09	22

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

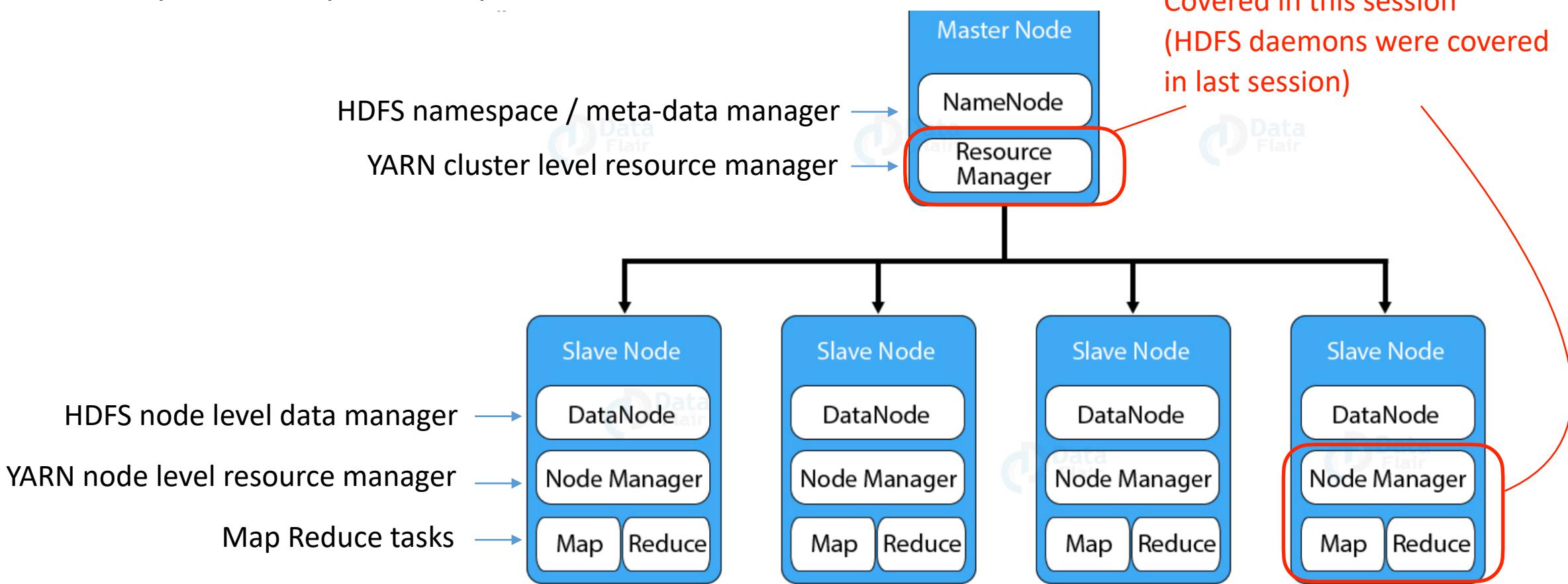
Topics for today

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



Hadoop 2 - Architecture

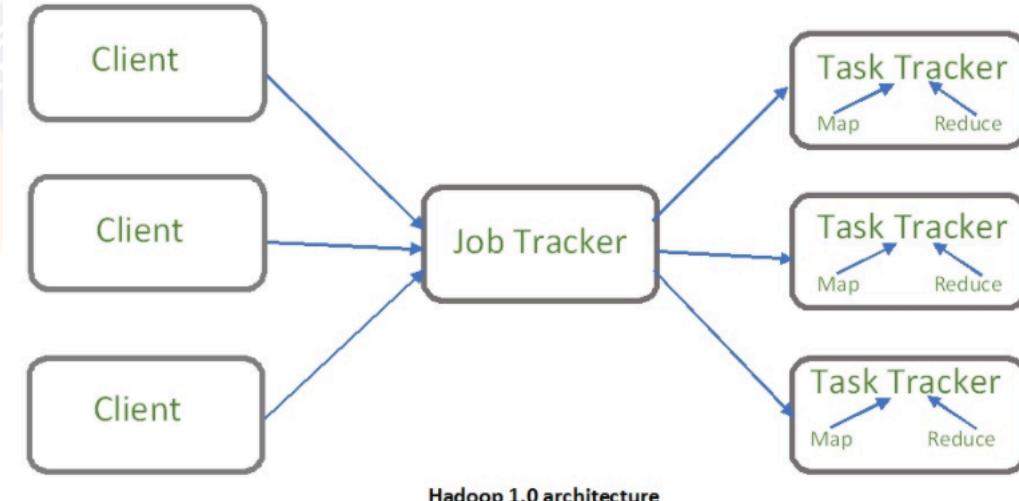
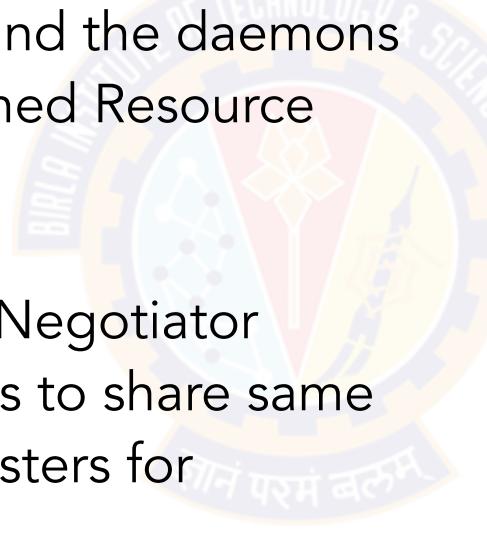
- Master-slave architecture for overall compute and data management
- Slaves implement peer-to-peer communication



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



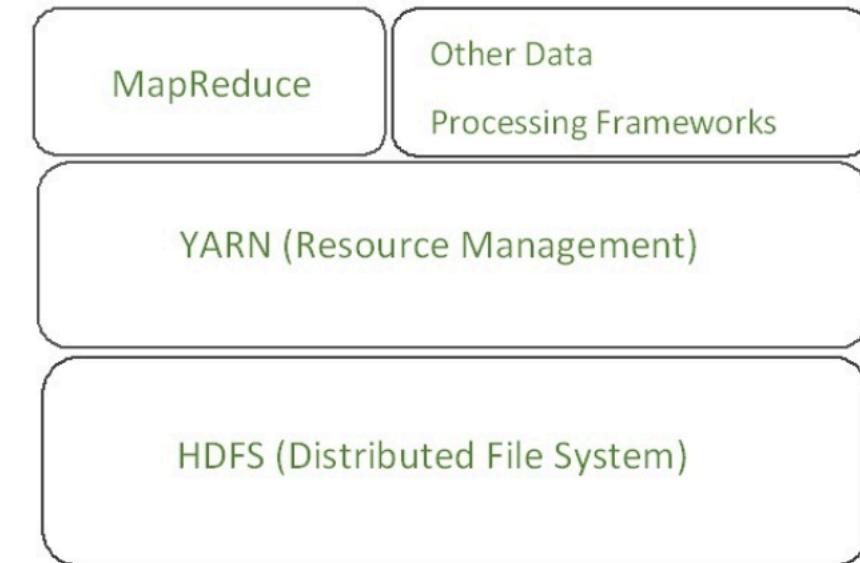
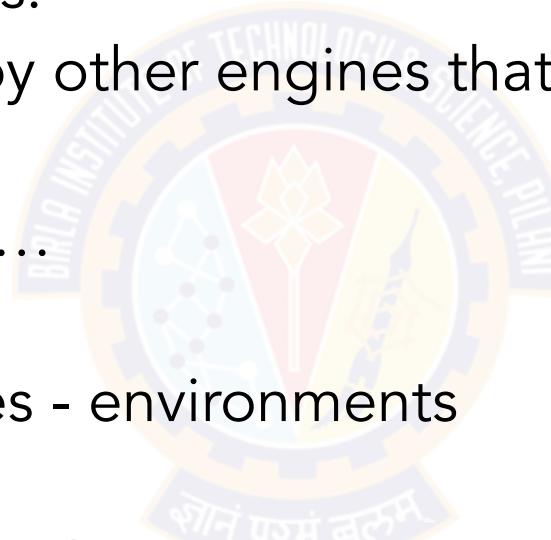
Hadoop 1.0 architecture

<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 utilisation
 - ✓ Multi-tenancy across various processing engines



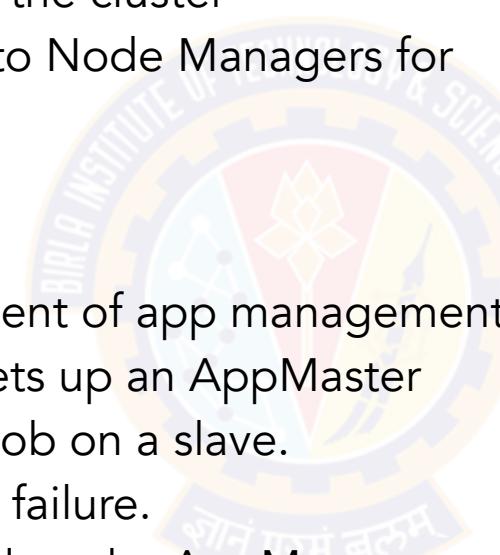
YARN components (1)

- **Client**

- ✓ Submits MR jobs

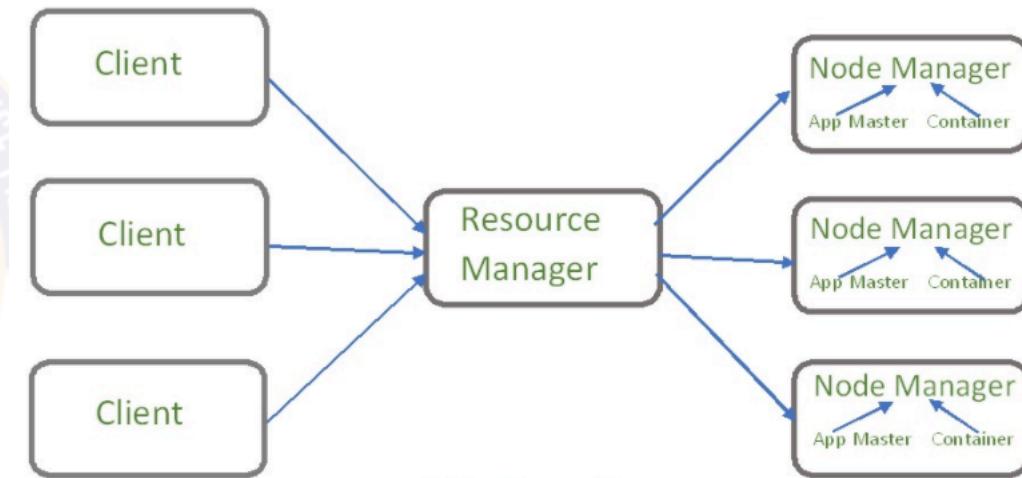
- **Resource manager**

- ✓ 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

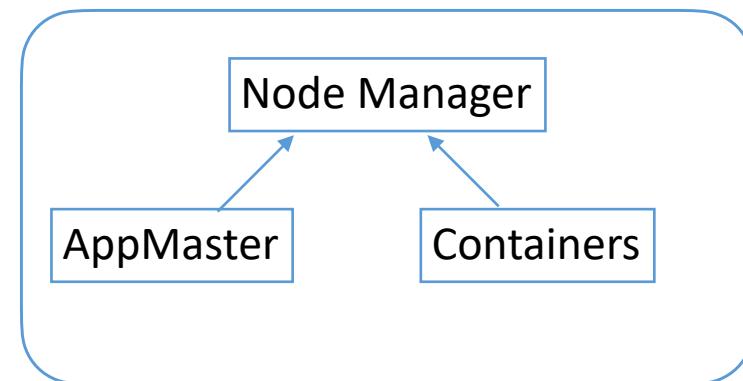
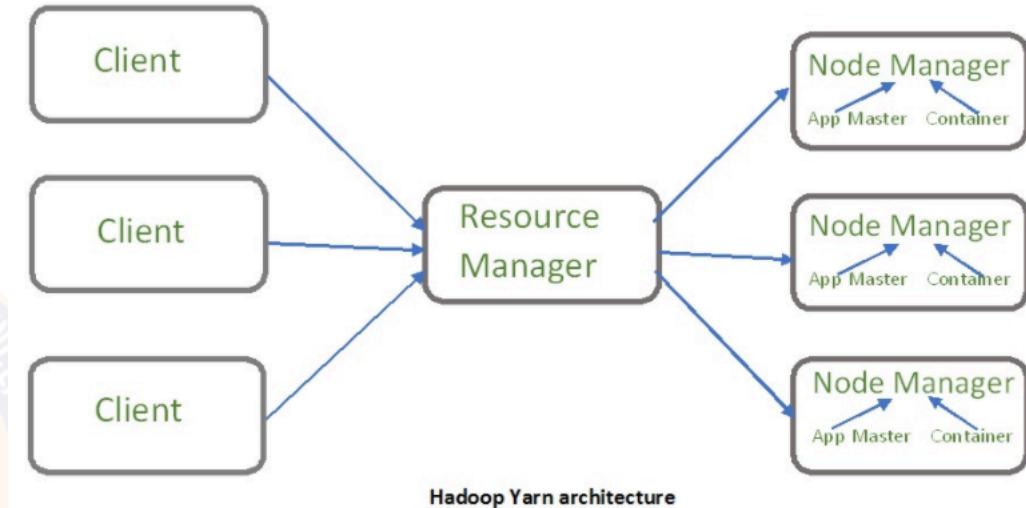
YARN components (2)

- **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



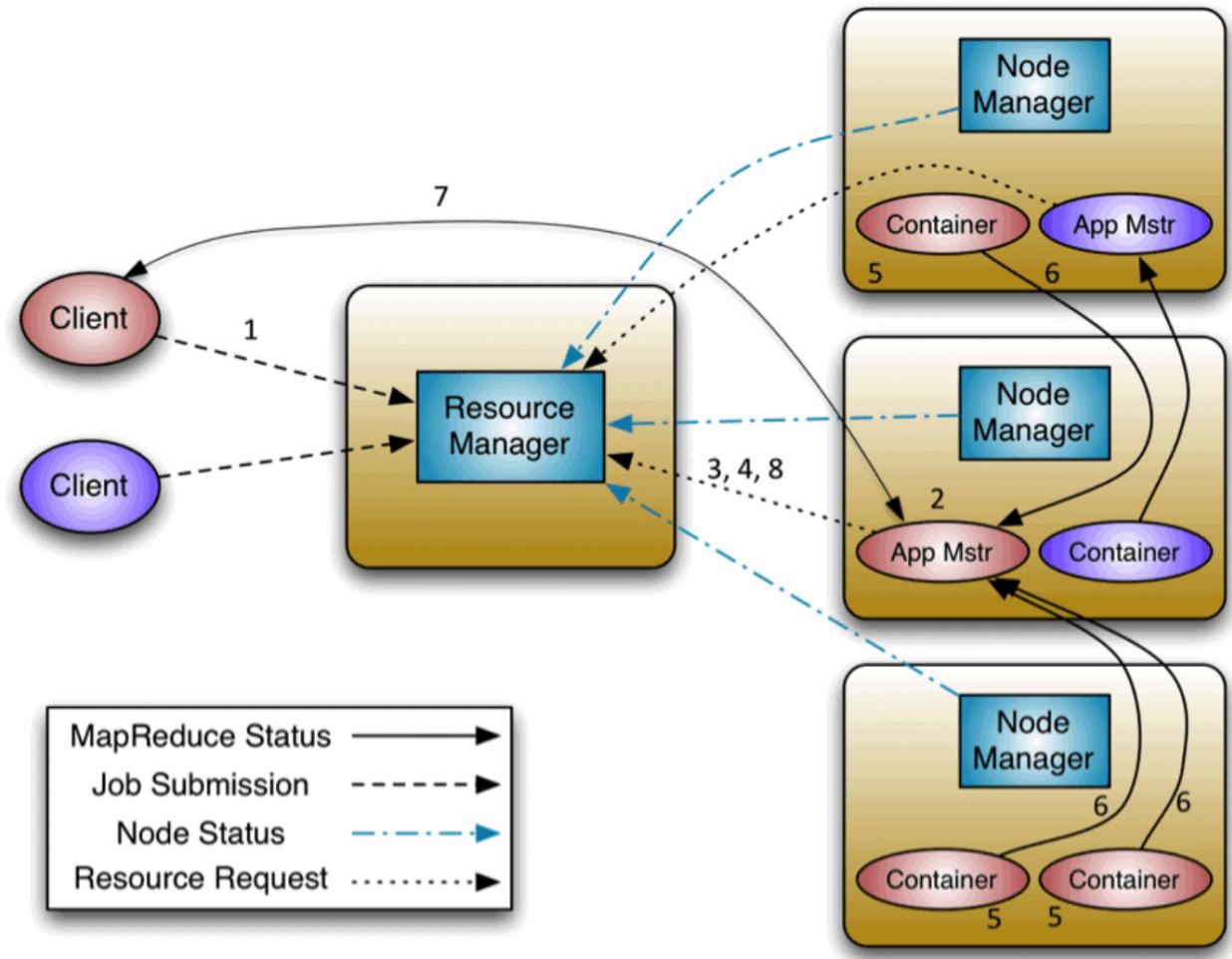
Topics for today

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



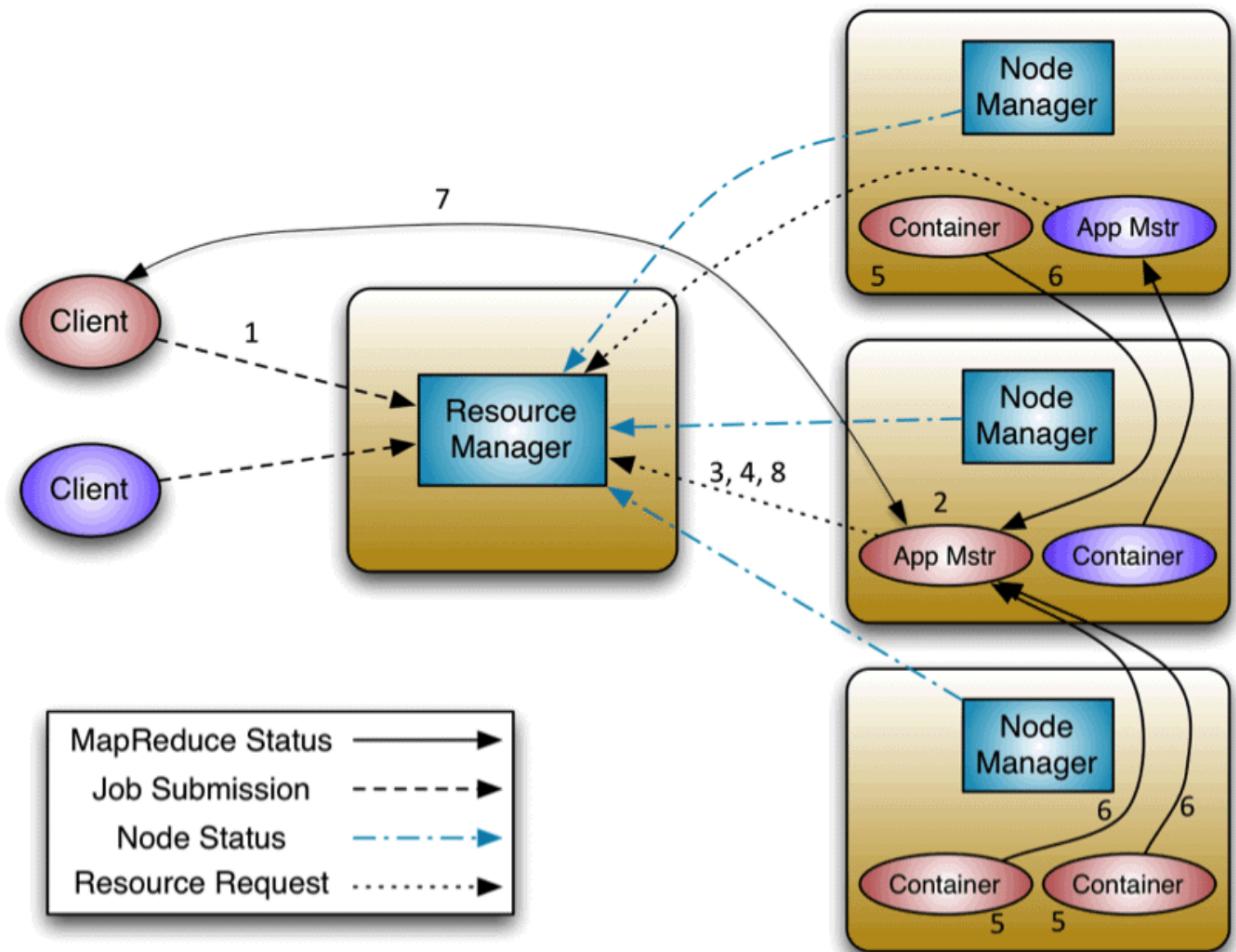
YARN workflow (1)

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.



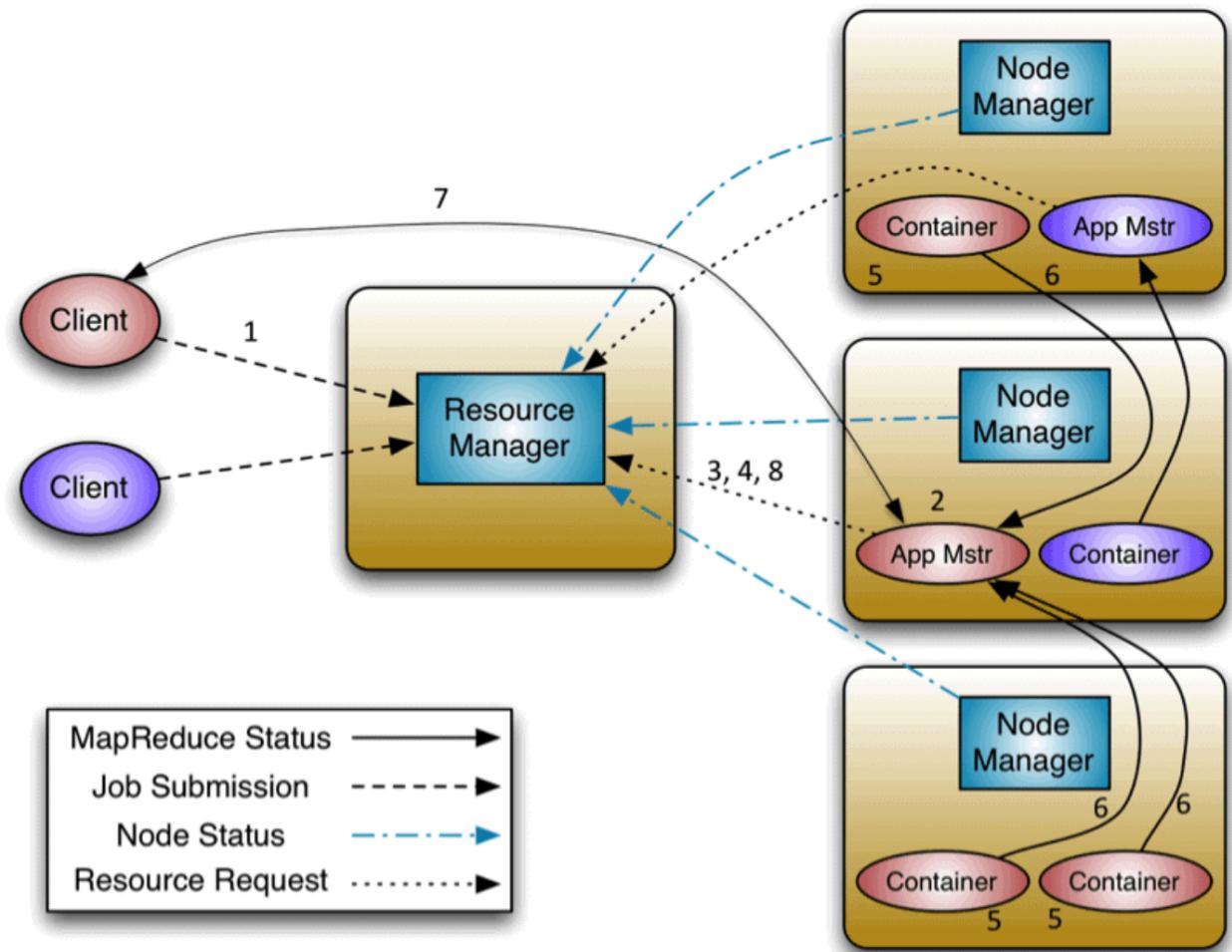
YARN workflow (2)

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)

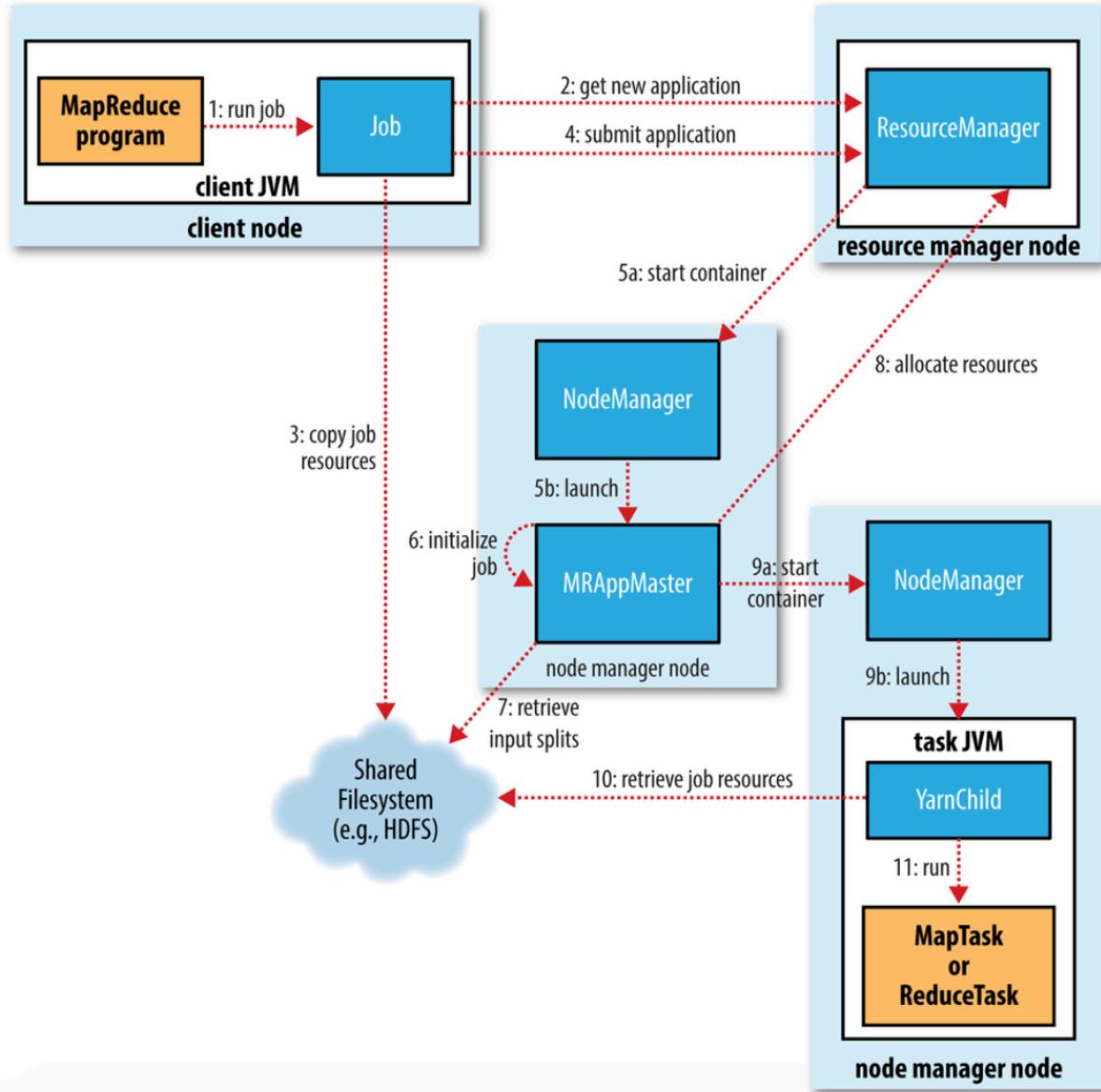
7. The client that submitted the app / job can directly **communicate with the AppMaster** for progress, status updates. via the application specific protocol.
8. On completion of the app / job, the **AppMaster de-registers** from ResourceManager and shuts down. So the containers allocated can be re-purposed.



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>

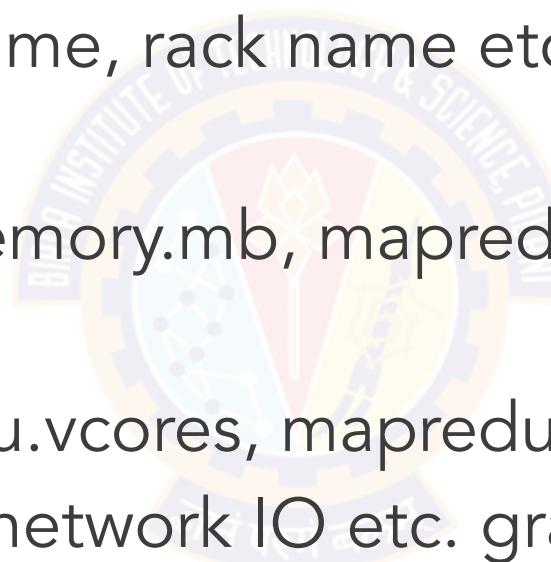
Topics for today

- Hadoop MapReduce
 - ✓ Walkthrough a MapReduce program
 - ✓ MapReduce runtime
 - ✓ More examples
- 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



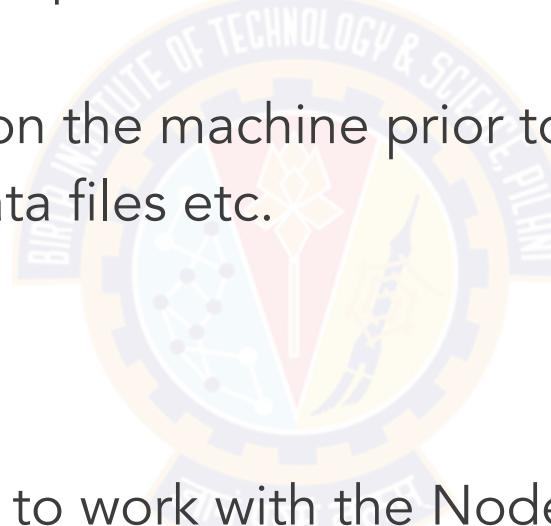
Resource requests

- 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 ?

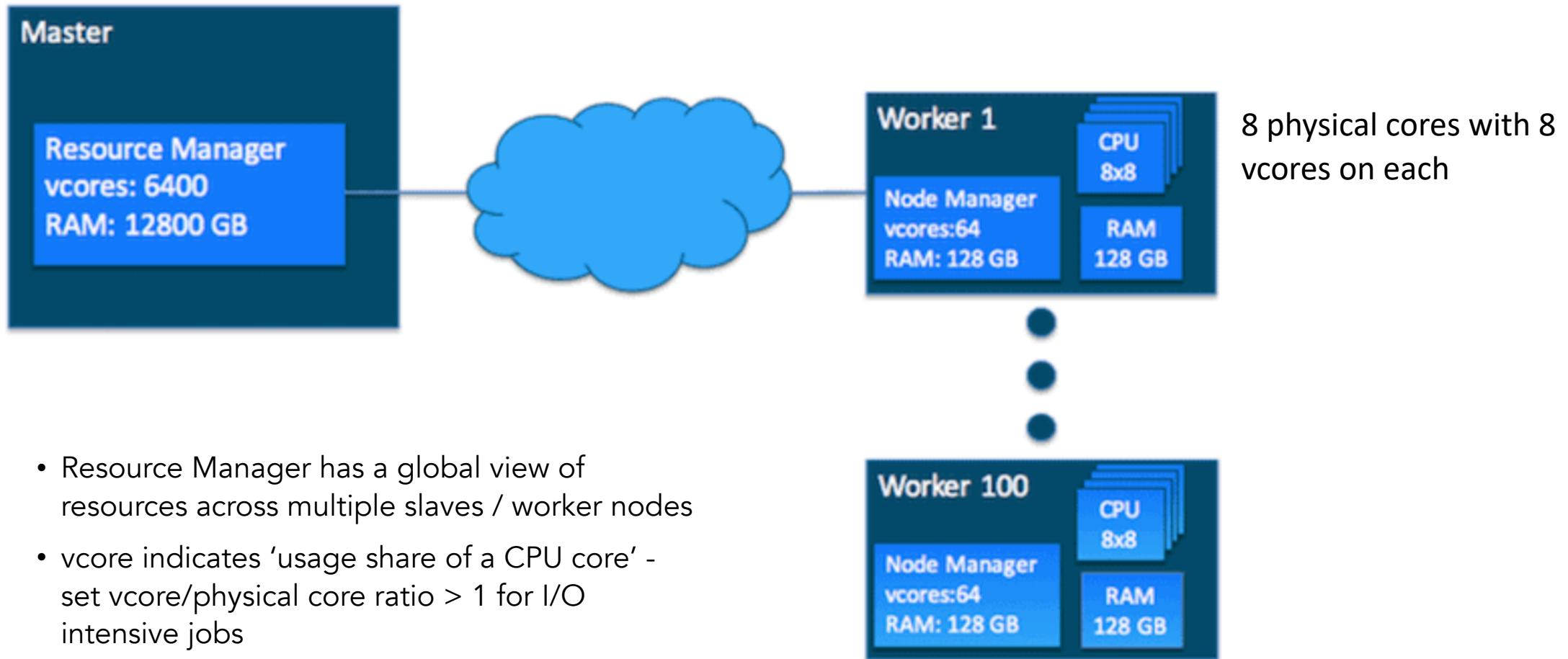
The YARN Container launch specification API is platform agnostic and contains:

- 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.

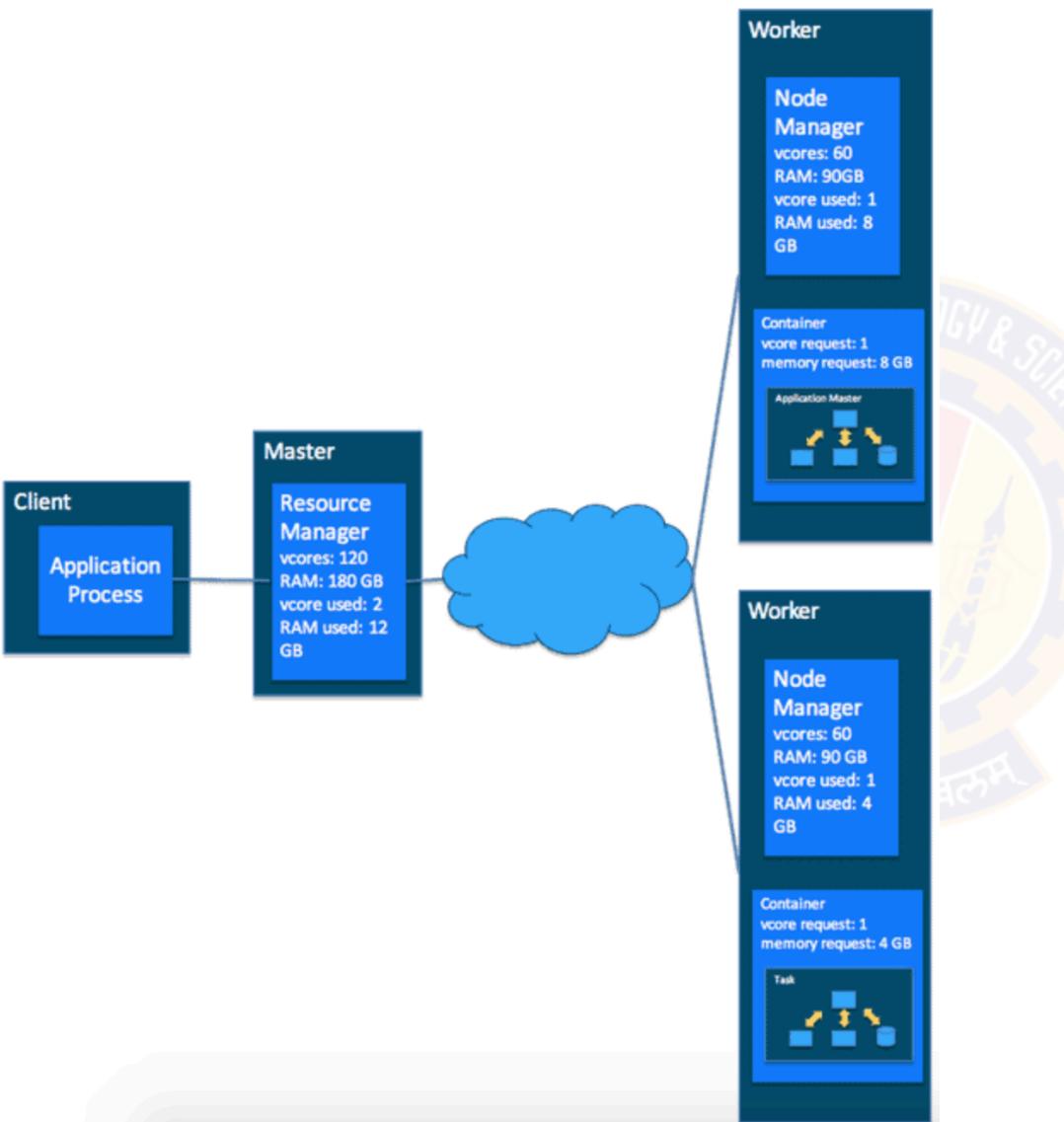


This allows the ApplicationMaster to work with the NodeManager to launch containers ranging from simple shell scripts to C/Java/Python processes on Unix/Windows to full-fledged virtual machines (e.g. KVMs)

Example: Global resource view

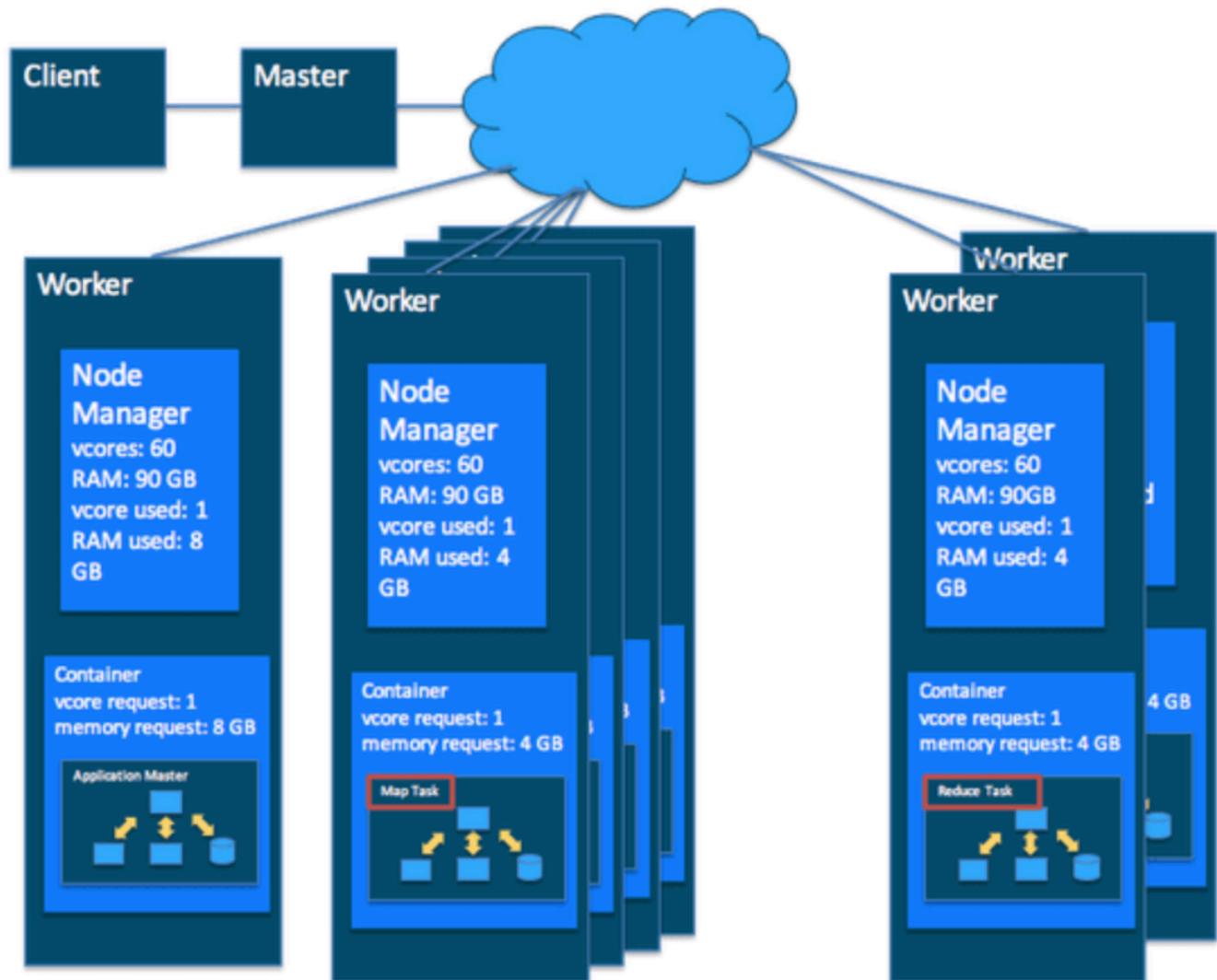


Example: Containers on slaves



- 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

Example: Map and Reduce tasks in containers



- Application specific tasks are started within the containers

Container executor

In conf/yarn-site.xml

```
<property>
  <name>yarn.nodemanager.container-executor.class</name>
  <value>org.apache.hadoop.yarn.server.nodemanager.LinuxContainerExecutor</value>
</property>

<property>
  <name>yarn.nodemanager.linux-container-executor.group</name>
  <value>hadoop</value>
</property>
```

Similarly there is also Windows container executor

An executor uses **setuid** for executing with the right permissions of the user.

Docker support:

- The Linux Container Executor (LCE) allows the YARN NodeManager to launch YARN containers to run either **directly on the host machine or inside Docker containers.**
- The application requesting the resources can specify for each container how it should be executed.
- When the LCE launches a YARN container to execute in a Docker container, the application can specify the Docker image to be used.

<https://hadoop.apache.org/docs/r2.9.2/hadoop-yarn/hadoop-yarn-site/DockerContainers.html>
<https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/SecureContainer.html>

Allocation enforcement mechanism

- Linux Control Groups (Cgroups) allow resource allocation among user defined processes
 - ✓ Resources can be CPU time, system memory, network bandwidth, block I/O, or combinations of these
- Cgroups are similar to processes in that:
 - ✓ They are hierarchical, and
 - ✓ Child cgroups inherit certain attributes from their parent cgroup.

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/resource_management_guide/ch01

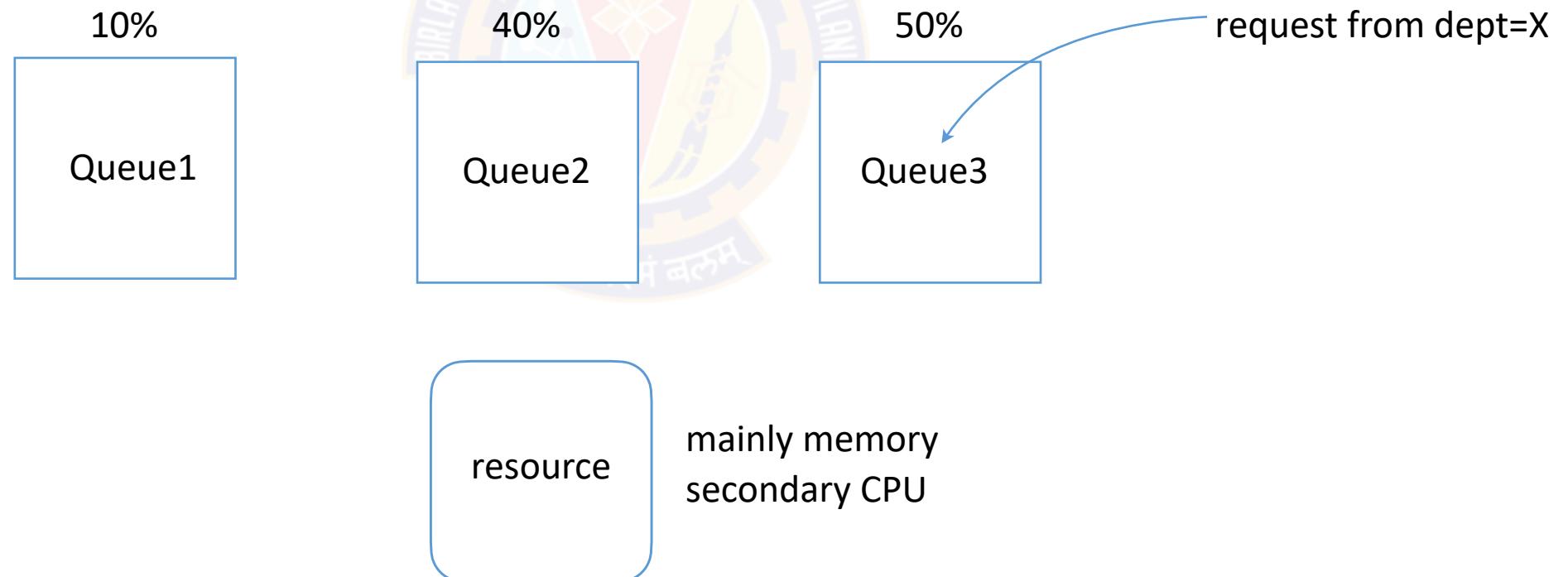
Topics for today

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



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.

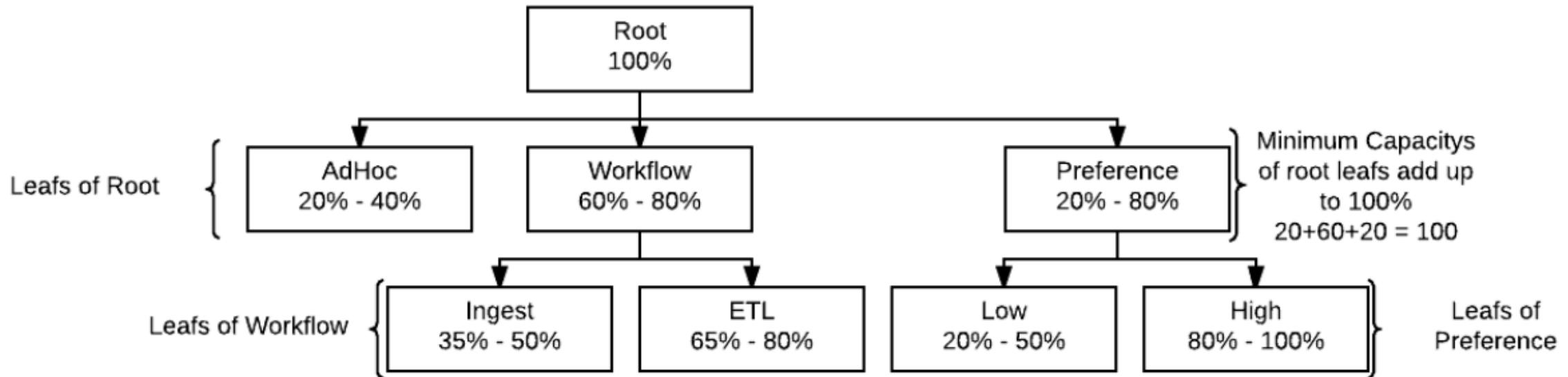


Resource Manager scheduling - Capacity scheduler

- **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
- **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

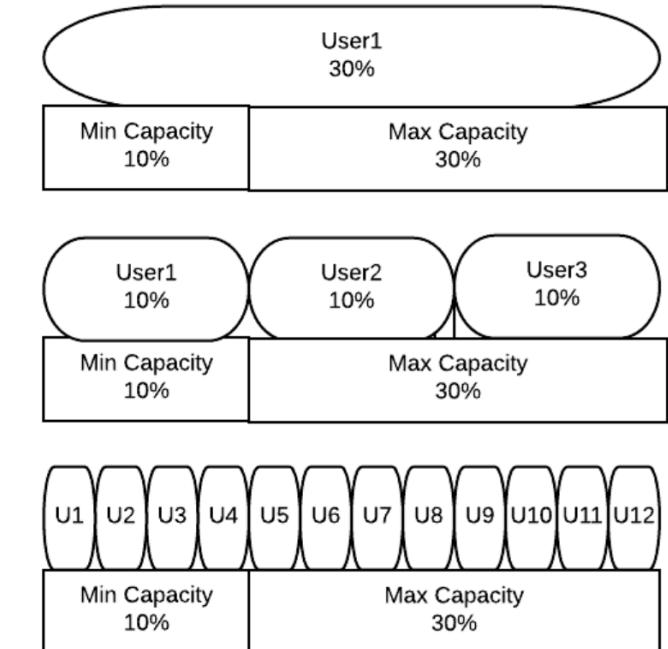
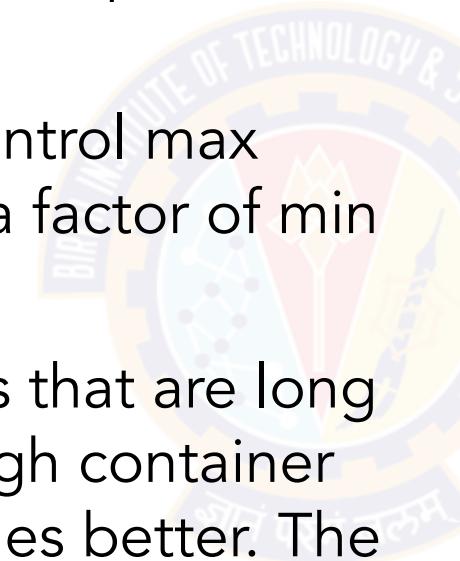
Capacity scheduler: Hierarchical queue allocations

- **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 not 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



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.

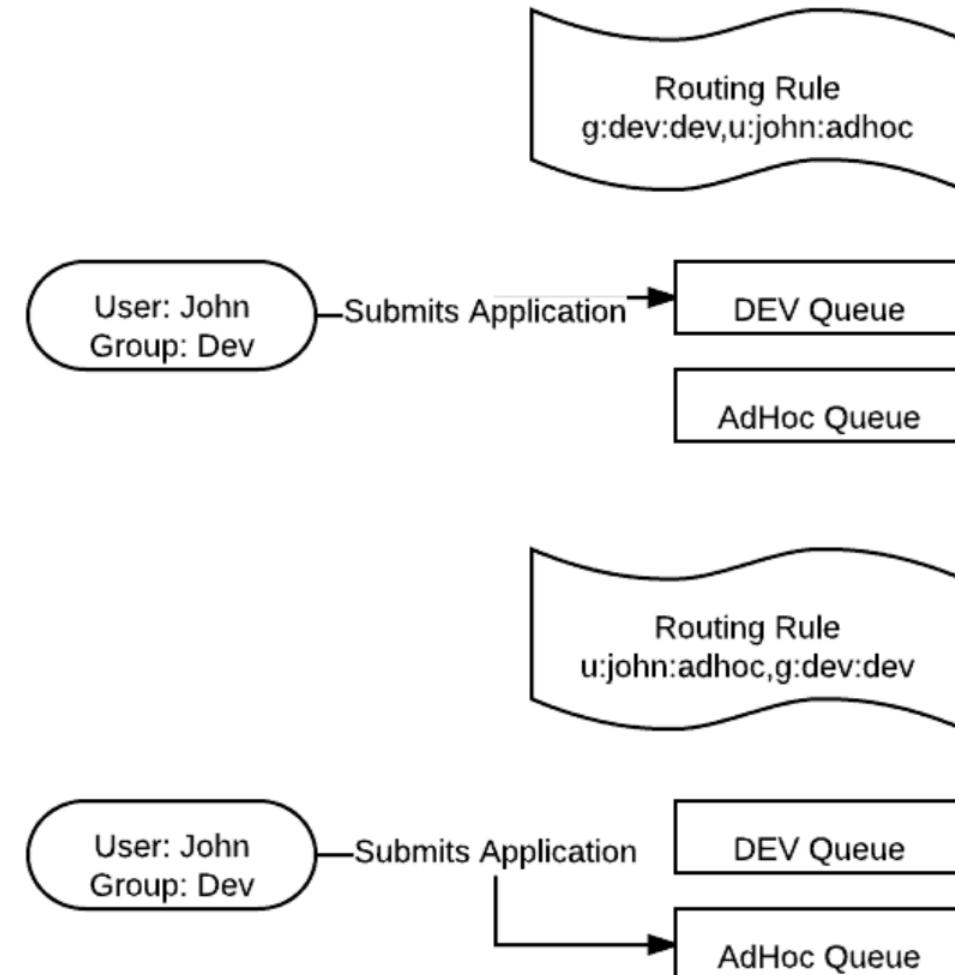


Capacity scheduler : Ordering policies within queue

- **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

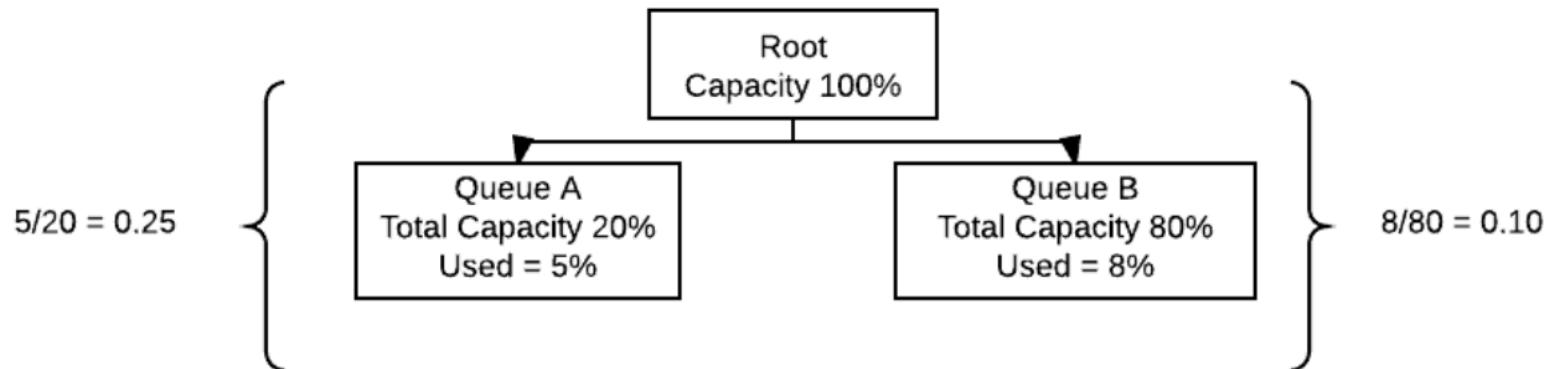
Capacity scheduler : Mapping to queue

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



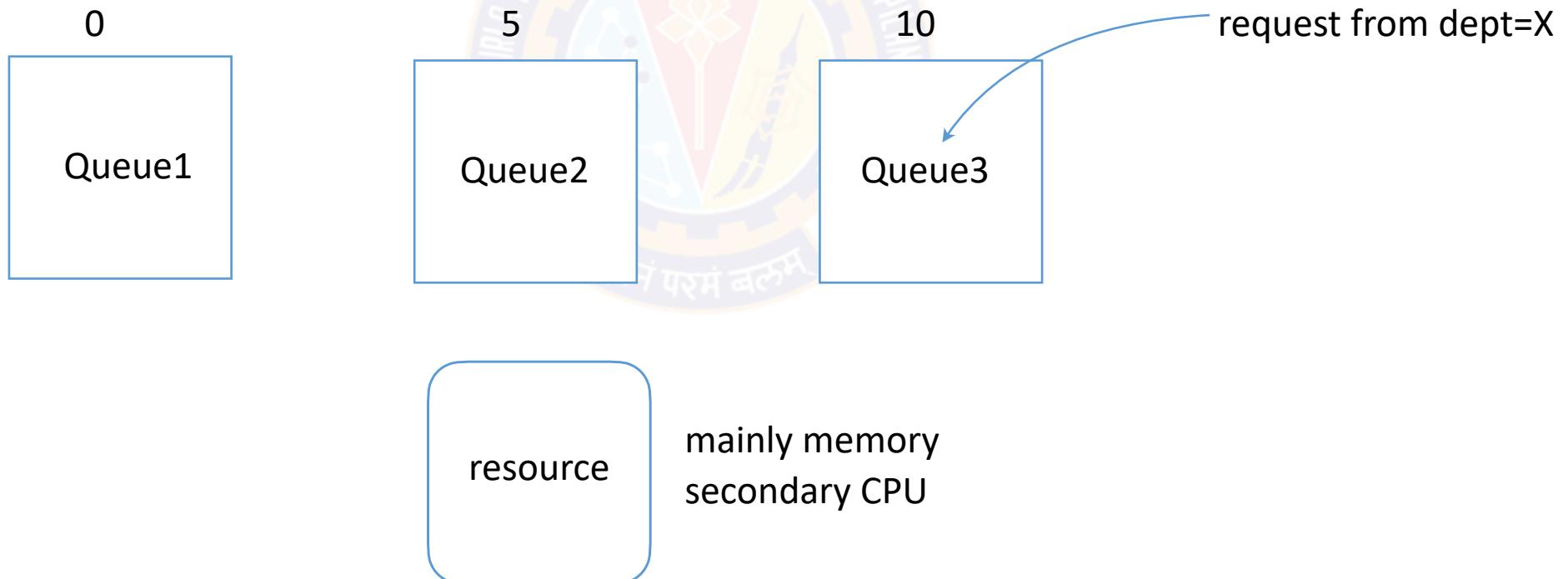
Capacity scheduler : Priority

- 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



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 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

Preemption across queues in Fair scheduler

- Scenario

- 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

- 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 anything unless it can fulfil entire allocation request

- Preemption is about only reclaiming min and not max allocation

- Within a queue one has to work with User limit factor or FIFO/FAIR ordering policies.

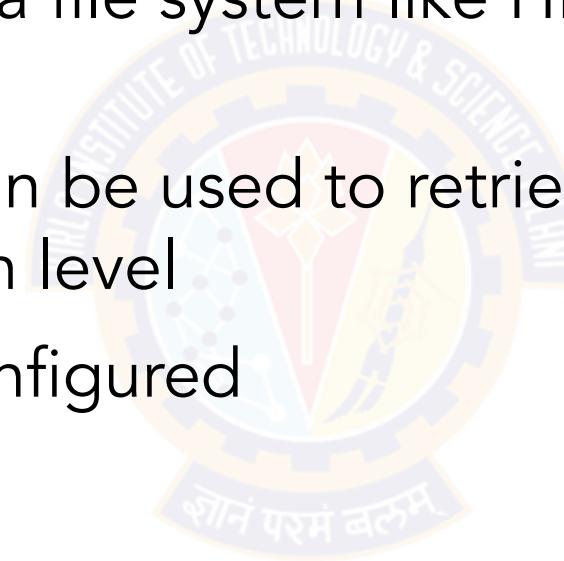
From below example, If queue1 is at 80% of its 50% share and need more resources and waiting for more than 60 seconds, queue1 is allowed to preempt containers from queue2. On a similar situation for queue2, it cannot preempt containers from queue1 because it does not have these parameters set.

```
<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>
```

[Reference link](#)

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

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 -appId <appid>`



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