Costa Rica Big Data School: Introduction to MapReduce and Hadoop for Big Data

Weijia Xu
Research Scientist, Group Manager
Data Mining & Statistics
Texas Advanced Computing Center
University of Texas at Austin
Dec. 2018

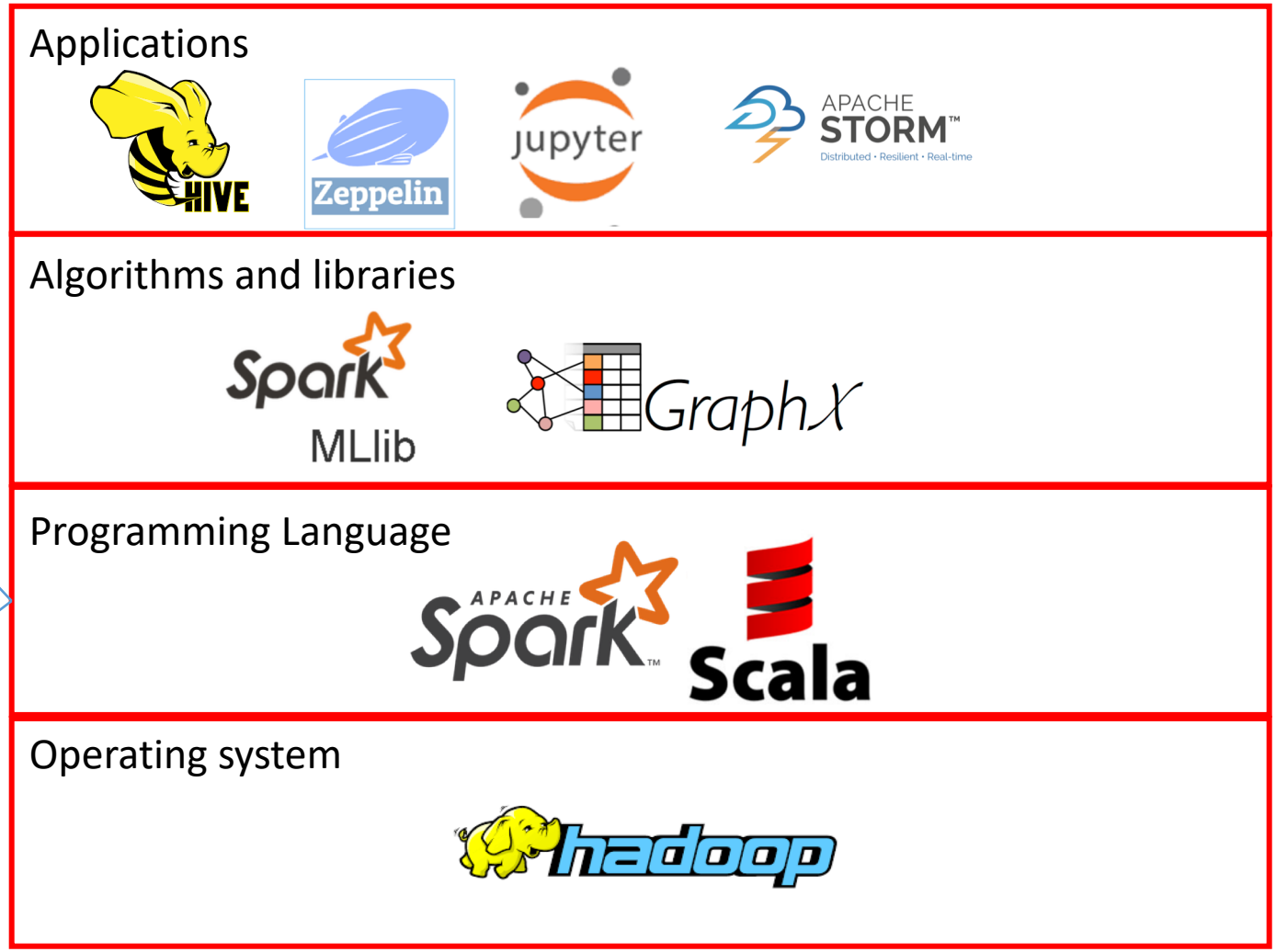
TACC

About me

- UT alumni
  - Ph.D. in Computer Science
- Now a research scientist at Texas Advanced Computing Center manage Data Mining & Statistics Group (DMS)
- DMS Group
  - Support Data and Machine Learning resources at TACC.
  - Collaboration with domain experts in diverse fields

Data Analysis and Machine Learning Tools

Caffe


Applications
HIVE
Zeppelin
jupyter
APACHE STORM™
Distributed • Resilient • Real-time
Algorithms and libraries
Spark MLlib
GraphX
Programming Language
APACHE Spark™
Scala
Operating system
hadoop


python


R

What is big data?

- Big data is
  - a lot of data
  - accumulating very fast
  - consisting of different types
  - often with a lot of noises

Volume
Velocity
Variety
Veracity

- How big is big data?

- Big data is “when you have too many data than you can handle”.

Let's consider a simple example, "wordcount"

- The problem: counting occurrences of unique words from text
  - The input:
    - A text file
  - The output:
    - a list of

      word, number of times the word occurred

- How would you solve it?

- But what about the input text file is
  - 1TB
  - 1PB…

What's the largest text data set you know?

- Google n-gram dataset
  - http://storage.googleapis.com/books/ngrams/books/datasetsv2.html
  - Basically a word count for all books Google has
  - But for 1~5 gram, per language, per year, per book, per page …
  - 2.2 TB compressed, ~9 TB when uncompressed in text
- ClueWeb09
  - https://lemurproject.org/clueweb09/
  - Text from ~1 billion web pages
  - 5TB compressed, ~25TB uncompressed
- How many web pages in the world?
  - ~5 billions?

Challenges with Big Data Analysis

- More computational resources required
  - Storage, triple the size of the raw data to store the intermediate files, output etc.
  - Memory, algorithm may need to hash more information in heap. e.g. the pair-wise distance matrix among data points
- The process would take much longer
  - Typical hard drive read speed is about 150MB/sec,
    - Reading 1TB ~ 2 hours
  - Complexity of the analysis matters
    - Complexity, time of a algorithm to finish with respect to the problem size.
      - e.g. wordcount is a “linear” problem.
    - Analysis could require processing time quadratic to the size of the data
      - Analysis took 1 second for 1GB data, would require 11 days to finish for 1TB data

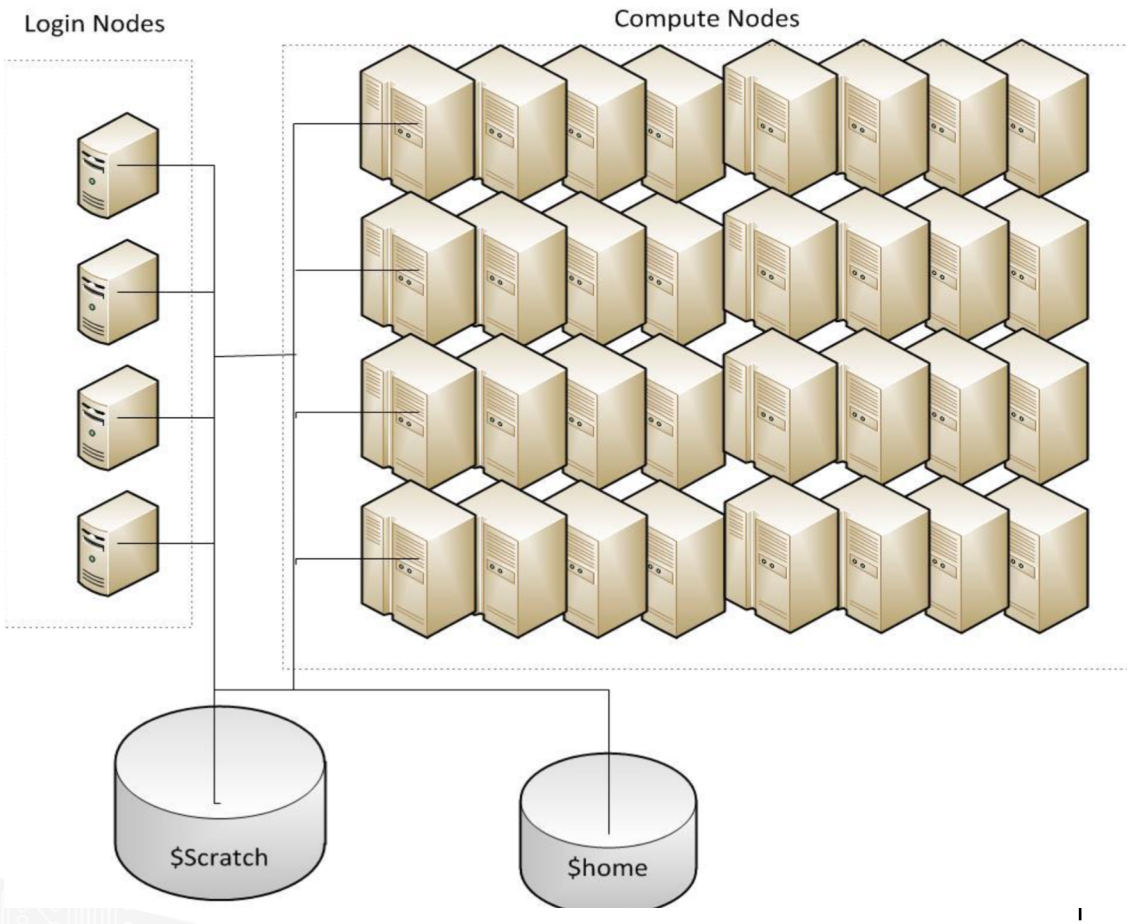
Solution?

- Let's use more computers to do it in parallel.
  - Great and simple idea

- But many practical issues
  - How to distribute data
  - How to track the data
  - How to coordinate works of concurrent processes
  - How to scale with different resources
  - How to handle failures and errors
  - …

Solution?

- Traditional HPC and HTC

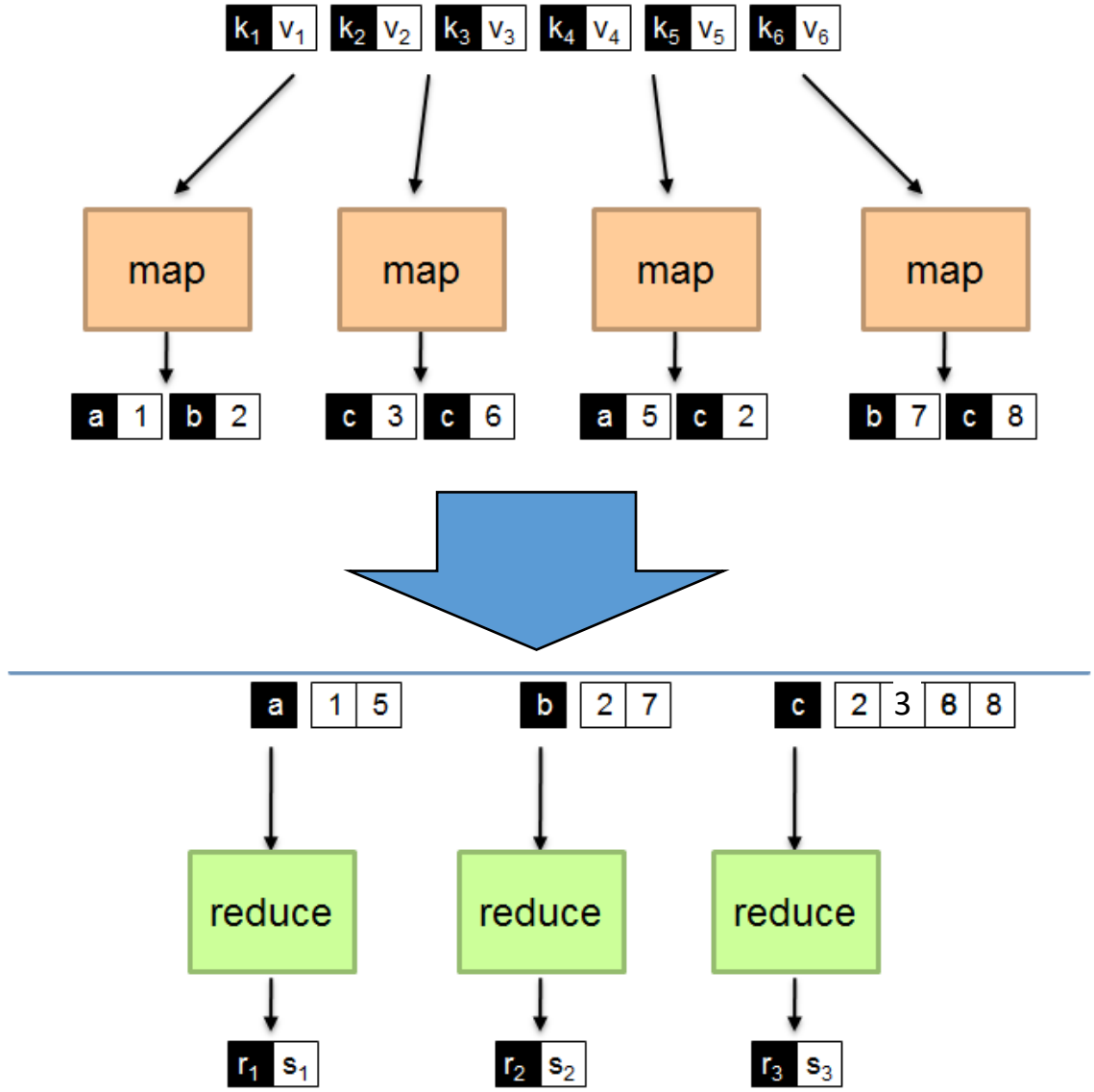


MapReduce Model for Big Data

- A programing model proposed and used by Google.
- Serving as a platform for customized computation over large scale of data.
- At the core, users implement interface of two main functions:

```
map   (in_key, in_value) ->
         (out_key, intermediate_value) list
reduce (out_key, intermediate_value list) ->
         out_value list
```

- Data are treated as Key/Value pairs.


k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1
b 2
c 3
c 6
a 5
c 2
b 7
c 8
a 1 5
b 2 7
c 2 3 6 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

WordCount with MapReduce

Read text files and count how often words occur.

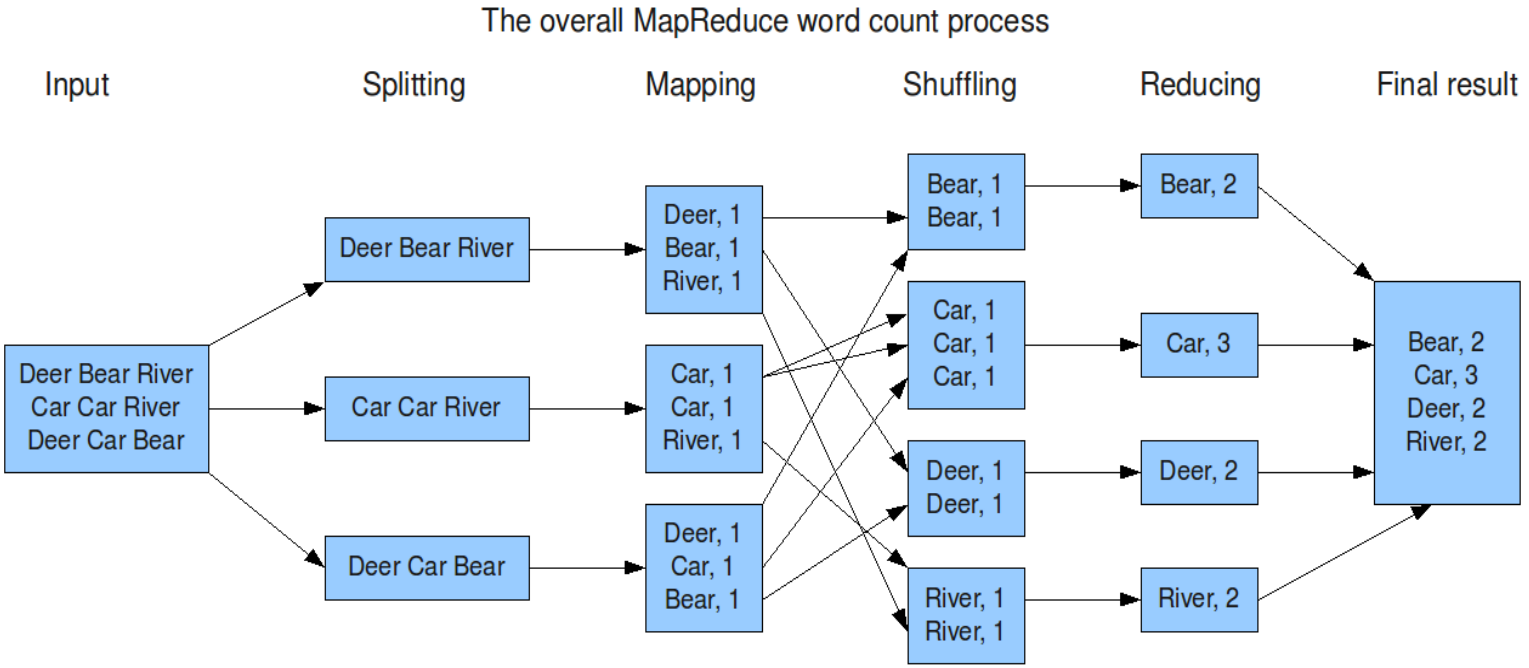
- The input is text files
- The output is a text file
  - each line: word, count

Map:

- Produce pairs of (word, count)

Reduce:

- For each key (word), sum up the value (counts).


The overall MapReduce word count process
Input
Splitting
Mapping
Shuffling
Reducing
Final result
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, 1
Bear, 1
Car, 1
Car, 1
Car, 1
Deer, 1
Deer, 1
River, 1
River, 1
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

Mapper - Java

Maps input key-value pairs to a set of intermediate key-value pair

Class for Individual tasks to run.

One mapper task per *InputSplit.*

Map function are automatically called per key Value pair.

```
public static class Map extends Mapper<Object, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map( Object key, Text value, Context context)
      throws IOException {
      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
      }
    }
}
```

Reducer - Java

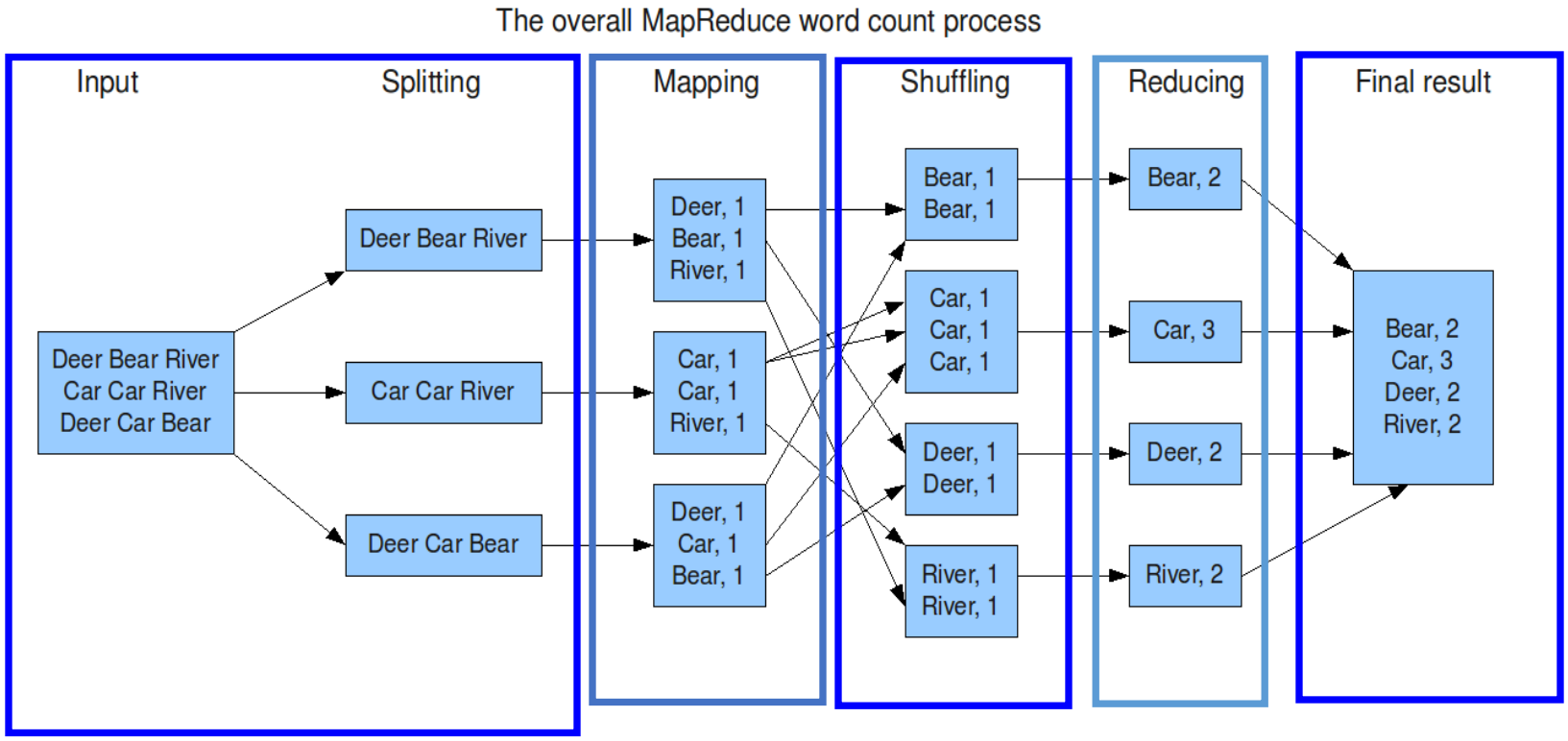
Reduces the set of values of the same key to a smaller set.

- Each reducer will process subset generated by partitioner
- Each reducer will generate an output file

```
public static class Reduce extends extends Reducer<Text, IntWritable,  Text,
    IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values,
                Context context)  throws IOException {
      int sum = 0;
      while (values.hasNext()) {
            sum += values.next().get();
      }
      result.set(sum);
      context.write(key, result);
    }
}
```

WordCount main - Java

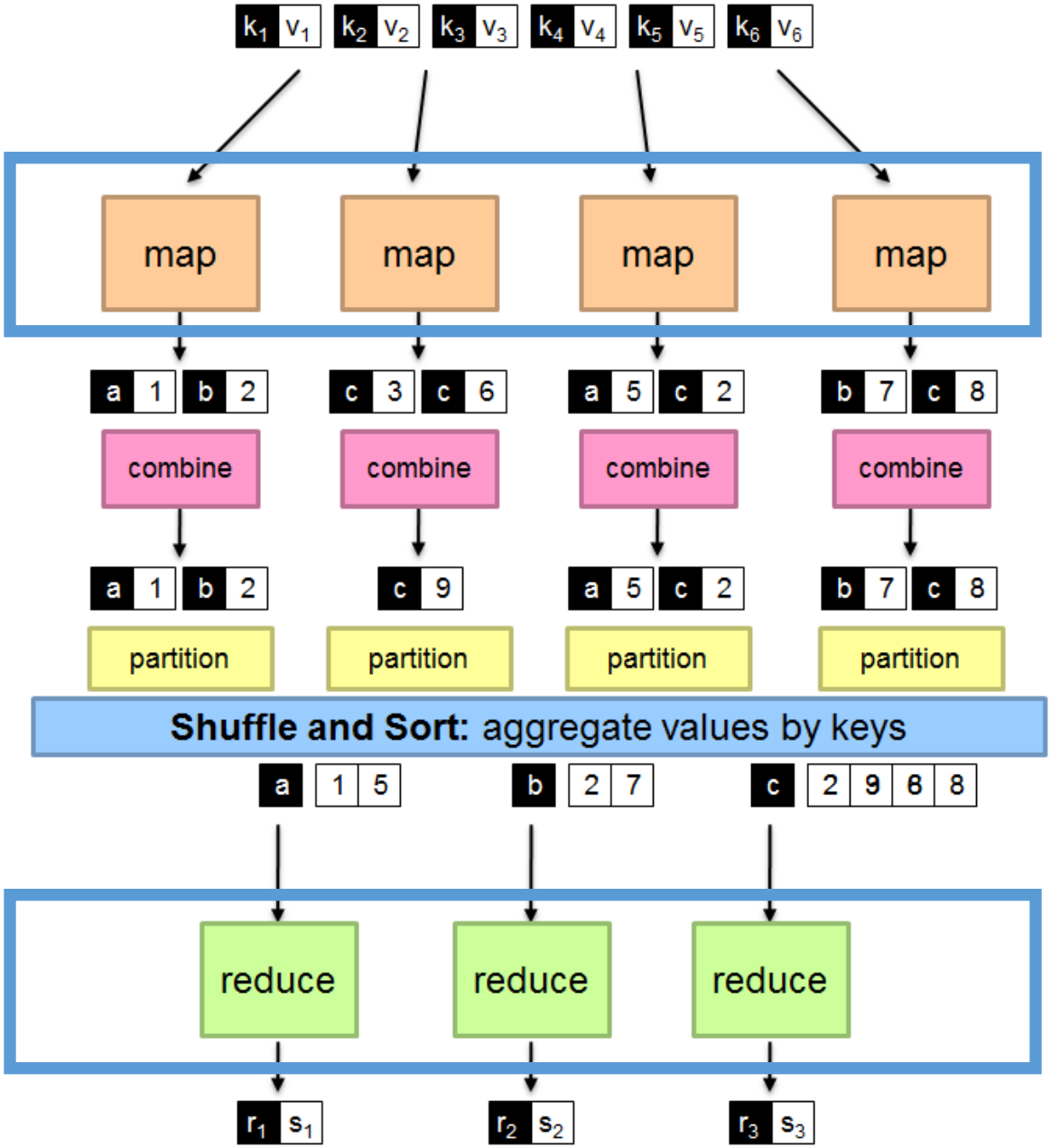
```
public static void main(String[] args) throws Exception {
        Job job = Job.getInstance(new Configuration(), "word count");

        job.setJarByClass(WordCount.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```



- MapReduce model separates the parallel process from the analysis.

Simple?

- But ...
  - Where is the parallelism?
  - How the data is split?
  - Who assign the work to each process?
  - How map results given to reduce?
  - Where is the output?
  - …


k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
combine
combine
combine
combine
a 1 b 2
c 9
a 5 c 2
b 7 c 8
partition
partition
partition
partition
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 6 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

“Big” Ideas in MapReduce

- Scale out instead of scale up:
  - The same code can run with 1 node or 100 nodes.
- Hide system details from user
  - Providing abstraction in writing parallel code.
    - Mapper and reducer
    - Partitioner and combiner
  - Isolates developer from (and independent from) system hardware details
  - Once all required components are specified, data is automatically “sliced” and processed in parallel.

“Big” Ideas in MapReduce

- Move computations to data.
  - Do not use/assume high bandwidth inter-connection between nodes.
  - Avoid and reduce the need of data transfer over the network is often the bottleneck.
- Processing data in sequential order, avoid random access
  - Same idea as applied in the RDBMS
  - However, assume data can be processed any order,
  - All data are packed in to blocks (default at 64MB).

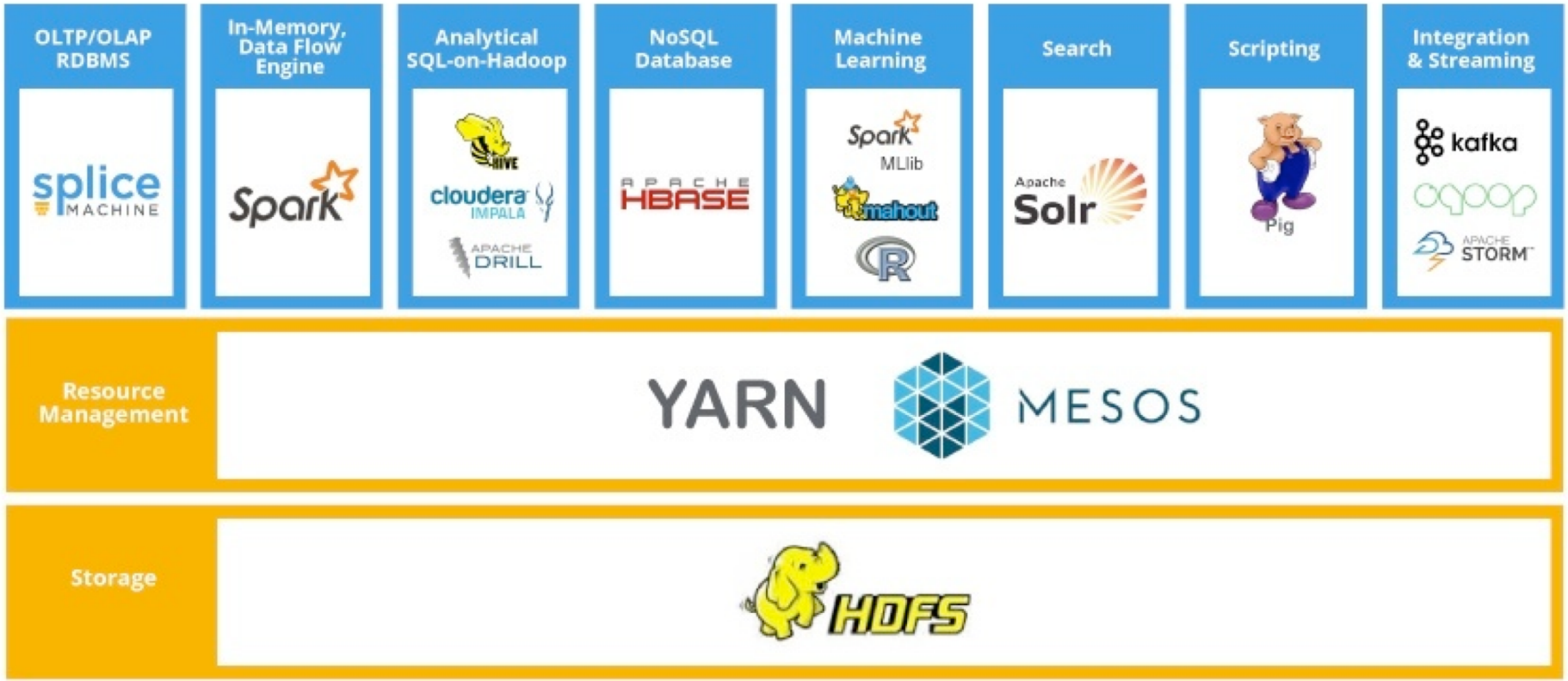
From MapReduce to Hadoop

- MapReduce is a neat programming model, but we still need an actual system to realize those advantages

- Hadoop is an open source implementation of MapReduce programming model in JAVA with interface to other programming language such as C/C++, python.

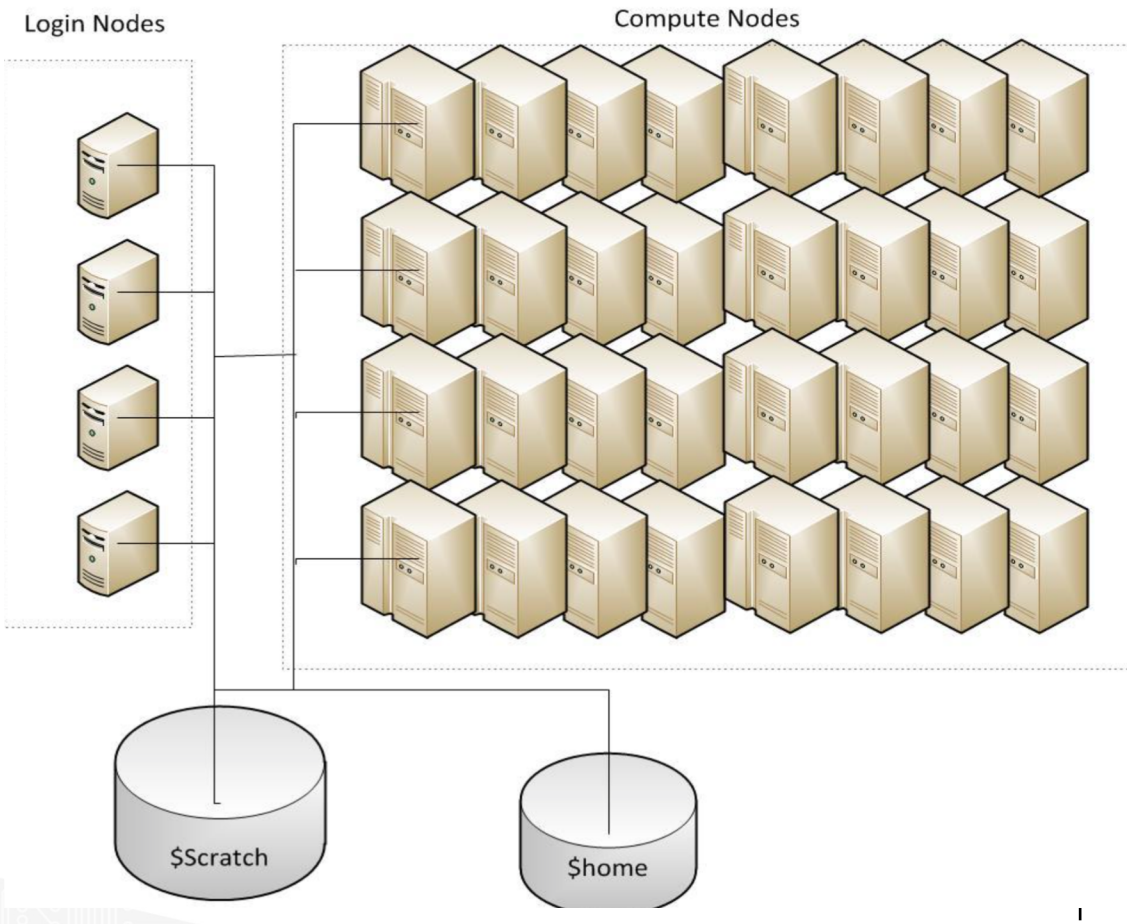
- But Hadoop is much more than a simple library.

Hadoop EcoSystem

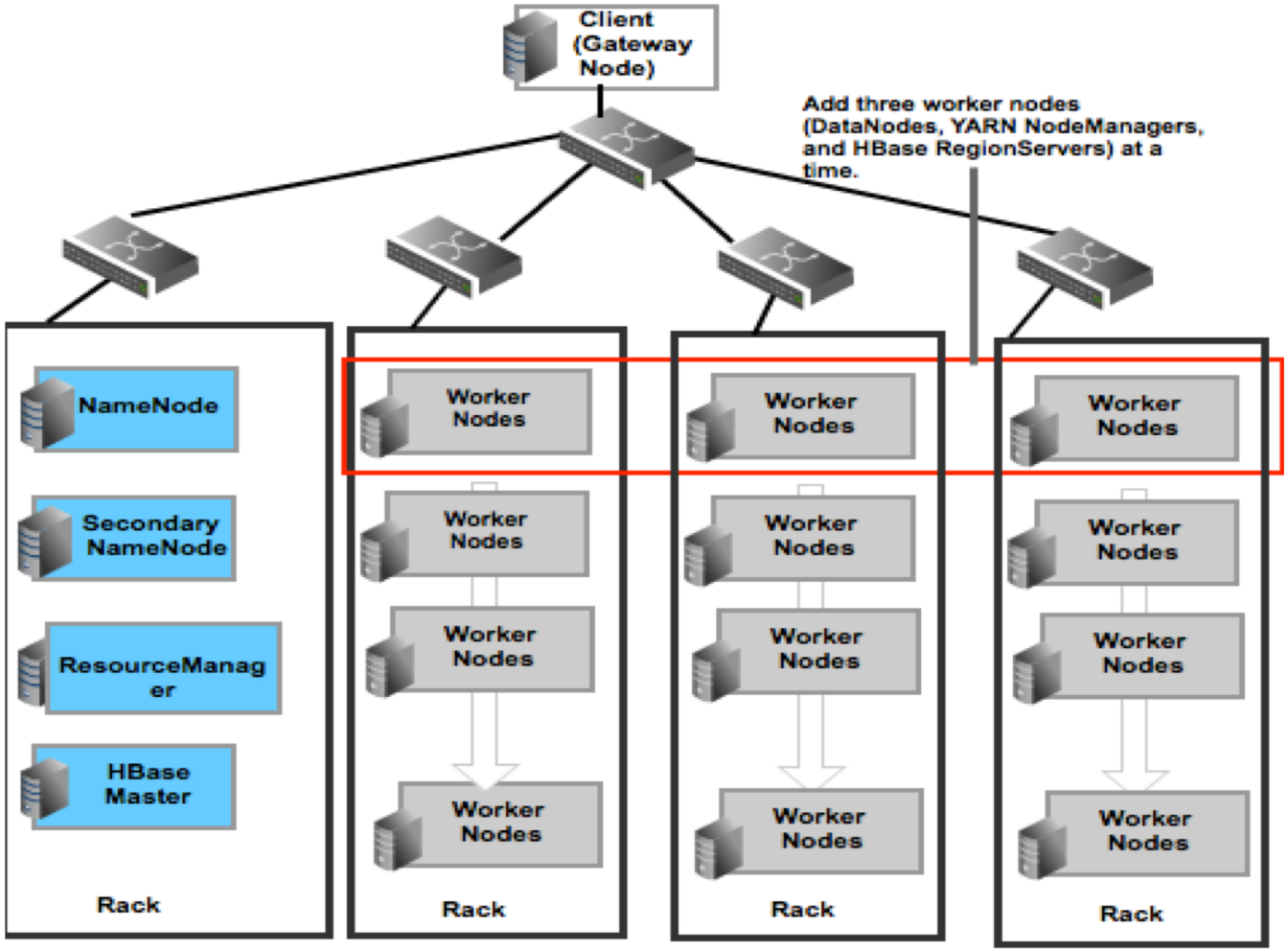


HPC Cluster

- Traditional HPC and HTC



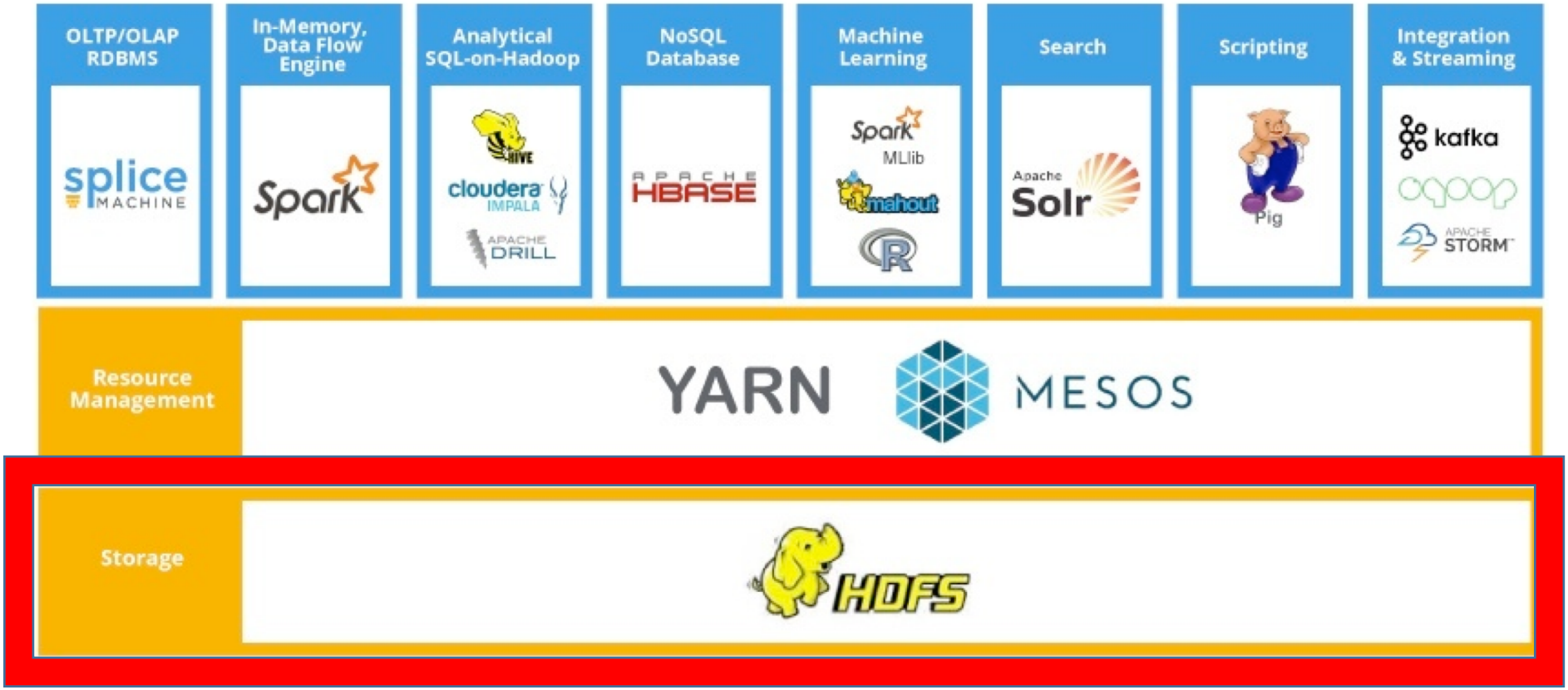
Hadoop Cluster



Type of Nodes in a Hadoop Cluster

- Name node
  - the “master”
  - Maintains a record of what and where data are stored across the cluster
  - Could be a single point of failure.
  - Multiple nodes may used for security and extensibility
- Data node
  - the “worker”
  - Each node usually has large local storage and large memory
  - Large data set are stored across data nodes with duplication
  - Resilience towards node failure/disk errors
- Service node
  - Metadata nodes for other application
  - Backup namenode
  - Usually not user accessible

Common Software Components with Hadoop Cluster



Key Components in Hadoop Cluster: HDFS

- Hadoop Distributed File System (HDFS)
  - The core software layer to handle efficient data storage.
  - A popular choice for many other big data applications
  - Optimized for large data sets
    - Smaller dataset may actually has negative impacts on performance
  - Strong fault tolerance potential
  - A file system built on top of the regular POSIX file system.
  - Regular system application cannot access HDFS directly,

Working with HDFS

HDFS has file system shell

hadoop fs [commands]

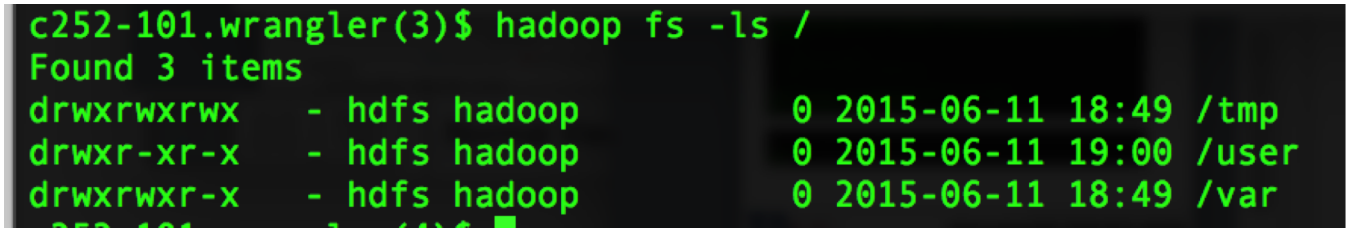
The file system shell includes a set of command to work with hdfs.

Command are similar to common linux commands e.g.

```
>hadoop fs -ls   #to list content of the default user directory.
>hadoop fs -mkdir abc #to make a directory in hdfs.
```

Hadoop Distributed File System (HDFS)

The hdfs will be set up with three top level directories



/tmp public writeable, used by many hadoop based application as temporary space.

/user all users home directory /user/$USERNAME

/var public readable, used by many hadoop based application to store log files etc.

Getting Data in and out HDFS

hadoop fs -put local_file [path_in_HDFS]

Put a file in your local system into the HDFS

Each file would be stored in one or more “blocks”

The default block size is 128MB.

The block size used can be override by users

Hadoop fs -get path_in_hdfs [path_in_local]

Get a file from the hadoop cluster to the local file system.

Other File Shell Commands

-stat returns stat information of a path

-cat/tail output to stdout

-setrep set replication factor

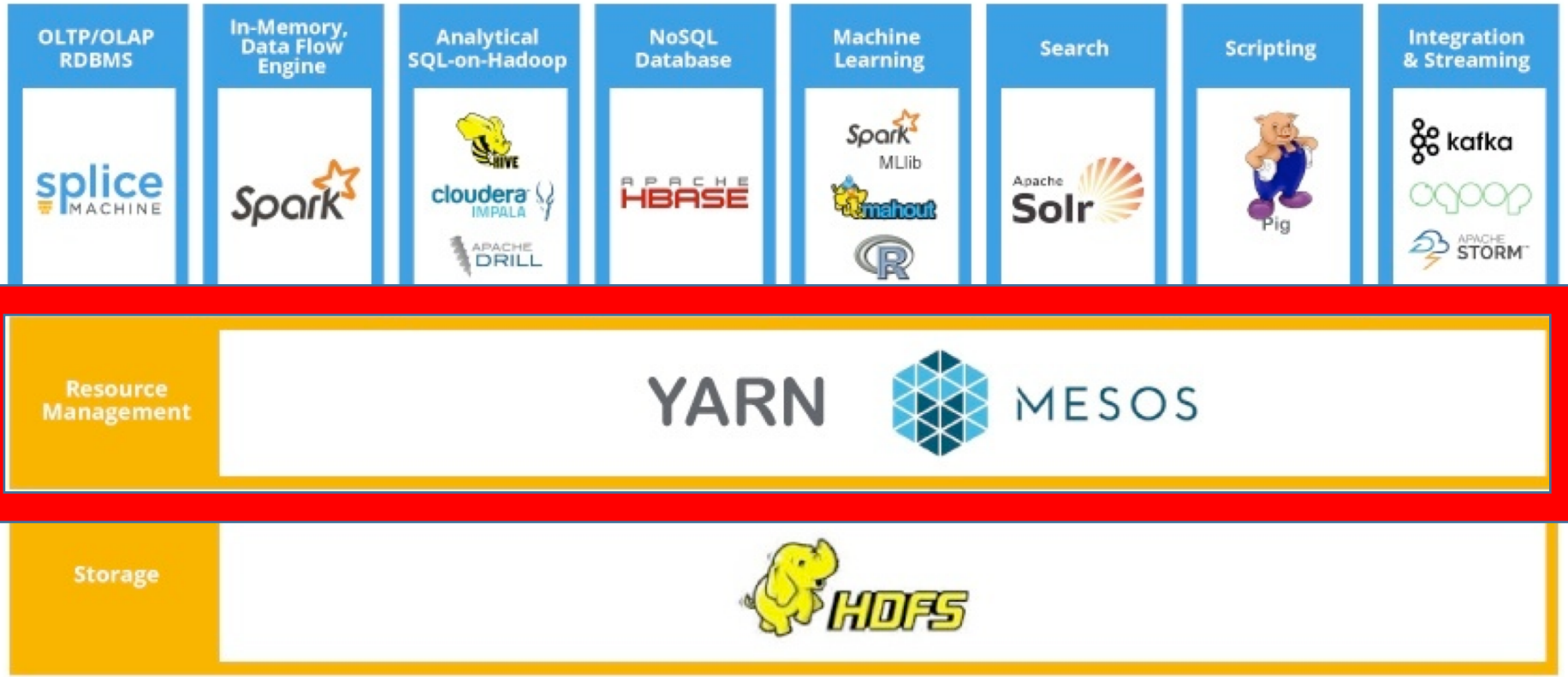
For a complete lists just do

hadoop fs

Or

visit http://hadoop.apache.org/docs/r2.5.2/hadoop-project-dist/hadoop-common/FileSystemShell.html

Common Software Components with Hadoop Cluster



Yet Another Resource Negotiator (YARN)

- A resource manager, determines number of processes available at a given moment towards a requests
- A job scheduler, schedule executions of multiple jobs.
- Can be used by other big data applications to request resources within the hadoop cluster
- There are also alternative options.

YARN Commands

yarn application

-list to list applications submitted to YARN

default will show active/queued application

```
Total number of applications (application-types: [] and st
                Application-Id      Application-Name
application_1431812610805_4832                GSO108
```

-kill to kill application specified by application-ID.

-appStates/appTypes filter options

YARN Commands

yarn node

-list list of status of data nodes

Let us know if there is less than expected live data nodes.

yarn logs

dump logs of a finished application

-applicationID specific log from which application

-containerID specify log from which container

Running Hadoop Application

All Hadoop application can be run as a console command.

The basic format is like following:

hadoop jar java_jar_name java_class_name [parameters]

The user can use –D to specify more Hadoop options. e.g.

-D mapred.map.tasks #number of map instances to be generated.

-D mapred.reduce.tasks # number of reduce instances to be used.

Running Wordcount example code

- The Hadoop distribution comes with an example jar which includes a set of exemplar map reduce code:

- To run wordcount example from that jar file, you can run command like following:

```
hadoop jar hadoop-mapreduce-examples.jar \
  wordcount   \  #java class name to run
  -D mapred.map.tasks=500     \  #number of mapper instance
  -D mapred.reduce.tasks=256 \ # number of reducer instance
/tmp/data/enwiki-20120104-pages-articles.xml  \    #input file on hdfs
wiki_wc     #folder to store the output
```

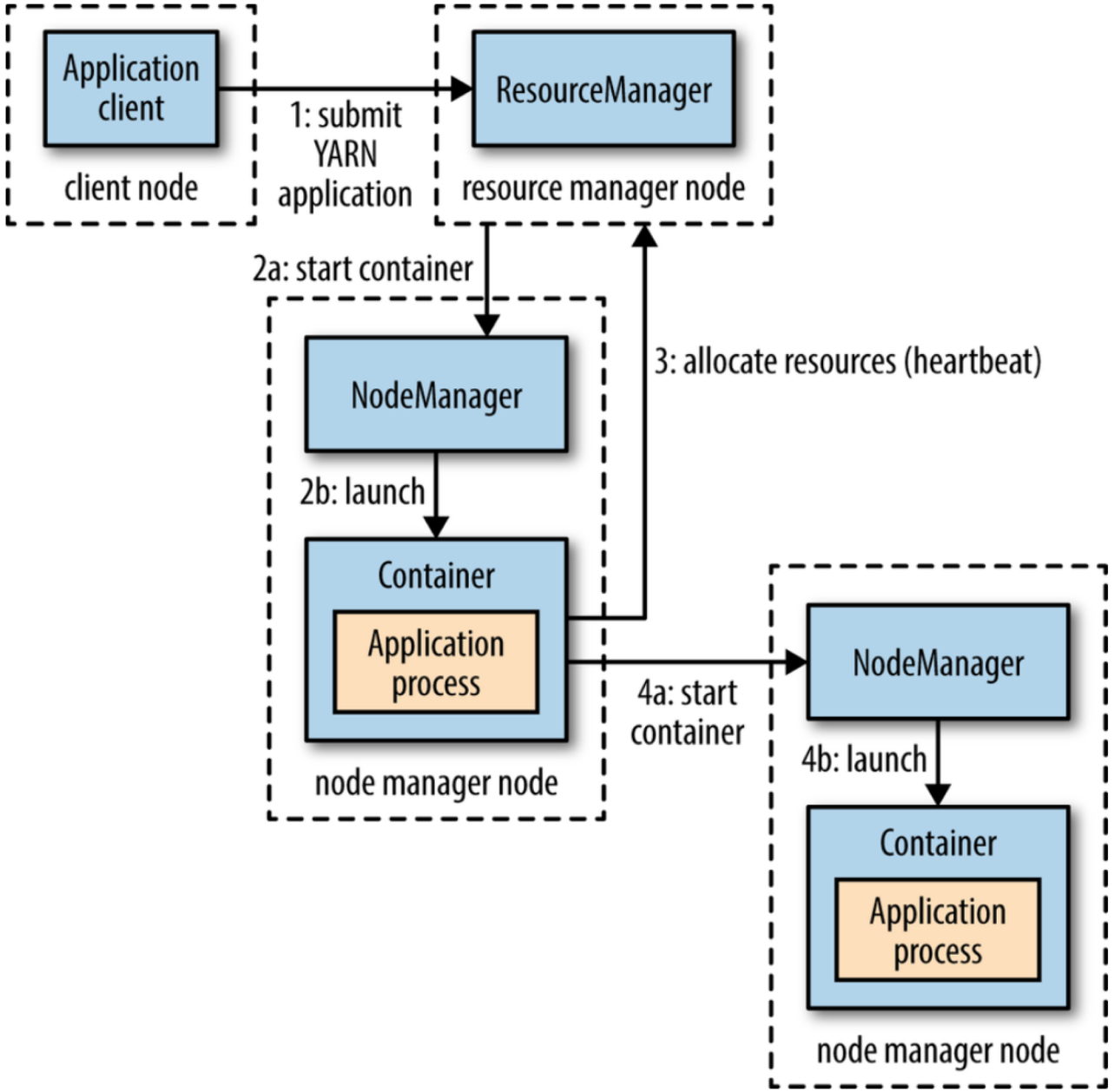
Running Wordcount Example code

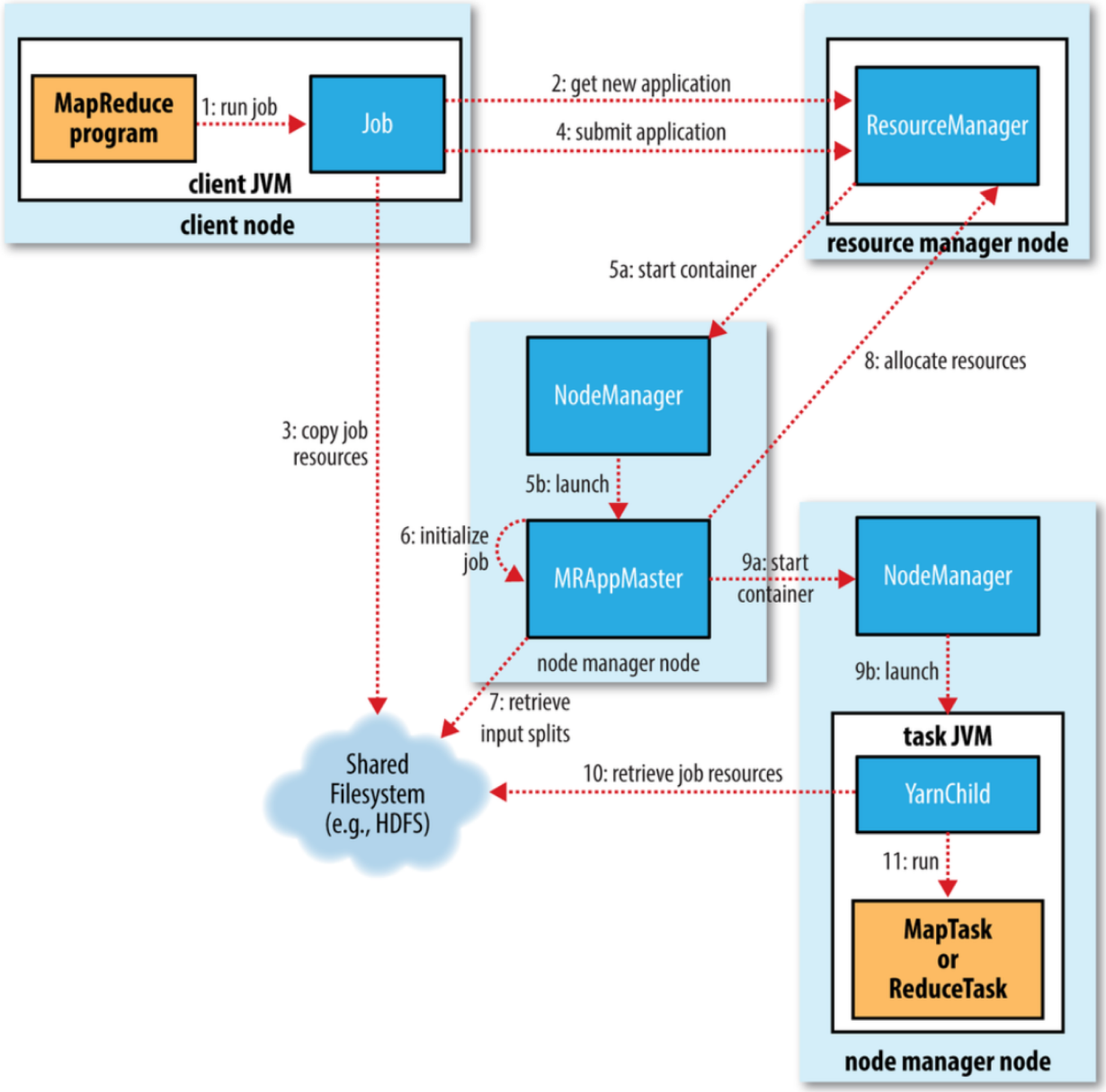
The on-screen output will show job running status

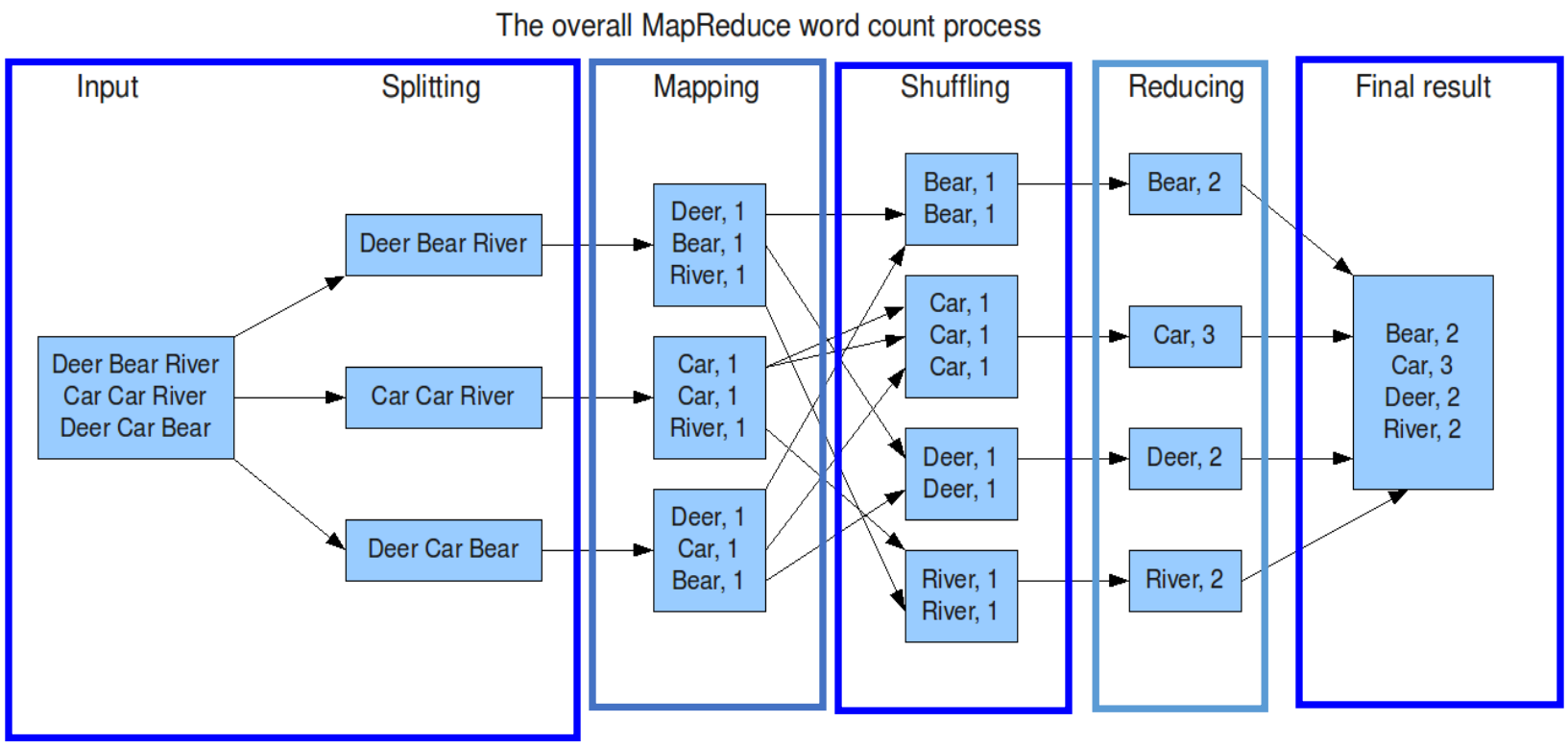
```
15/05/29 02:50:59 INFO client.RMProxy: Connecting to ResourceManager at name.rustler.tacc.utexas.edu/129.114.57.132:8032
15/05/29 02:51:01 INFO input.FileInputFormat: Total input paths to process : 1
15/05/29 02:51:02 INFO mapreduce.JobSubmitter: number of splits:266
15/05/29 02:51:02 INFO Configuration.deprecation: mapred.reduce.tasks is deprecated. Instead, use mapreduce.job.reduces
15/05/29 02:51:02 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
15/05/29 02:51:02 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1431812610805_3390
15/05/29 02:51:03 INFO impl.YarnClientImpl: Submitted application application_1431812610805_3390
15/05/29 02:51:03 INFO mapreduce.Job: The url to track the job: http://name.rustler.tacc.utexas.edu:8088/proxy/application_1431812610805_3390/
15/05/29 02:51:03 INFO mapreduce.Job: Running job: job_1431812610805_3390
15/05/29 02:51:09 INFO mapreduce.Job: Job job_1431812610805_3390 running in uber mode : false
15/05/29 02:51:09 INFO mapreduce.Job:  map 0% reduce 0%
15/05/29 02:51:20 INFO mapreduce.Job:  map 14% reduce 0%
15/05/29 02:51:21 INFO mapreduce.Job:  map 24% reduce 0%
15/05/29 02:51:22 INFO mapreduce.Job:  map 32% reduce 0%
15/05/29 02:51:23 INFO mapreduce.Job:  map 45% reduce 0%
15/05/29 02:51:24 INFO mapreduce.Job:  map 52% reduce 0%
15/05/29 02:51:25 INFO mapreduce.Job:  map 57% reduce 0%
15/05/29 02:51:26 INFO mapreduce.Job:  map 62% reduce 0%
15/05/29 02:51:27 INFO mapreduce.Job:  map 65% reduce 0%
15/05/29 02:51:28 INFO mapreduce.Job:  map 66% reduce 0%
15/05/29 02:51:30 INFO mapreduce.Job:  map 67% reduce 0%
15/05/29 02:51:38 INFO mapreduce.Job:  map 68% reduce 0%
15/05/29 02:51:39 INFO mapreduce.Job:  map 72% reduce 0%
15/05/29 02:51:40 INFO mapreduce.Job:  map 76% reduce 0%
15/05/29 02:51:41 INFO mapreduce.Job:  map 81% reduce 0%
15/05/29 02:51:42 INFO mapreduce.Job:  map 87% reduce 0%
15/05/29 02:51:43 INFO mapreduce.Job:  map 93% reduce 0%
15/05/29 02:51:44 INFO mapreduce.Job:  map 97% reduce 0%
15/05/29 02:51:45 INFO mapreduce.Job:  map 98% reduce 0%
15/05/29 02:51:46 INFO mapreduce.Job:  map 99% reduce 0%
15/05/29 02:51:47 INFO mapreduce.Job:  map 100% reduce 0%
15/05/29 02:51:49 INFO mapreduce.Job:  map 100% reduce 10%
15/05/29 02:51:50 INFO mapreduce.Job:  map 100% reduce 24%
15/05/29 02:51:51 INFO mapreduce.Job:  map 100% reduce 42%
15/05/29 02:51:52 INFO mapreduce.Job:  map 100% reduce 76%
15/05/29 02:51:53 INFO mapreduce.Job:  map 100% reduce 98%
15/05/29 02:51:54 INFO mapreduce.Job:  map 100% reduce 100%
15/05/29 02:51:57 INFO mapreduce.Job: Job job_1431812610805_3390 completed successfully
15/05/29 02:51:58 INFO mapreduce.Job: Counters: 50
        File System Counters
                FILE: Number of bytes read=12501618427
                FILE: Number of bytes written=25053078456
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=35618393055
                HDFS: Number of bytes written=6489193532
                HDFS: Number of read operations=1566
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=512
        Job Counters
                Launched map tasks=266
                Launched reduce tasks=256
                Data-local map tasks=213
                Rack-local map tasks=53
                Total time spent by all maps in occupied slots (ms)=15587438
                Total time spent by all reduces in occupied slots (ms)=5572760
                Total time spent by all map tasks (ms)=7793719
                Total time spent by all reduce tasks (ms)=2786380
```

How Hadoop cluster run a job?

- User submit job request
- YARN accepts the job and assign appropriate resources in the form of containers.
- The “application master” starts execution
- The application release container and resources back to the cluster when its done


Application client
client node
1: submit YARN application
ResourceManager
resource manager node
2a: start container
NodeManager
2b: launch
Container
Application process
node manager node
3: allocate resources (heartbeat)
4a: start container
NodeManager
4b: launch
Container
Application process
node manager node


MapReduce program
1: run job
Job
client JVM
client node
2: get new application
4: submit application
ResourceManager
resource manager node
5a: start container
NodeManager
8: allocate resources
3: copy job resources
5b: launch
6: initialize job
MRAppMaster
node manager node
9a: start container
NodeManager
9b: launch
task JVM
YarnChild
11: run
MapTask or ReduceTask
node manager node
7: retrieve input splits
Shared Filesystem (e.g., HDFS)
10: retrieve job resources


The overall MapReduce word count process
Input
Splitting
Mapping
Shuffling
Reducing
Final result
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, 1
Bear, 1
Car, 1
Car, 1
Car, 1
Deer, 1
Deer, 1
River, 1
River, 1
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

Hadoop Streaming API

```
hadoop jar hadoop-streaming.jar
    -input /path/to/input/in/hdfs                    #input file location
    -output /path/to/output/in/hdfs                  #output file location
    -mapper map                                      # mapper implementation
     -reducer reduce                                 #reduce implementation
     -file map          #location of the map code on local file system
     -file reduce       # location of the reduce code on local file system.
```

The map and reduce could be implemented in any programming language, even with bash script.

WordCount using Bash with Hadoop streaming

The map code:

```
-bash-4.1$ more mapwc.sh
#!/bin/bash

exclude="\.\,?!\-_:;\]\[\#\|\$()\""
while read split; do
 for word in $split; do
   term=`echo "${word//[$exclude]/}" | tr [:upper:] [:lower:]`
   if [ -n "$term" ]; then
     printf "%s\t%s\n" "$term" "1"
   fi
 done
done
```

WordCount using Bash with Hadoop streaming

The reduce code

```
-bash-4.1$ more reducewc.sh
#!/bin/bash

read currterm currnum
while read term num; do
  if [[ $term = "$currterm" ]] ; then
    currnum=$(( currnum + num ))
  else
    printf "%s\t%s\n" "$currterm" "$currnum"
    currterm="$term"
    currnum="$num"
  fi
done
printf "%s\t%s\n" "$currterm" "$currnum"
```

WordCount using Bash with Hadoop streaming

Putting together:

```
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar  \
   -D mapred.map.tasks=512  \
   -D mapred.reduce.tasks=256  \
   -D stream.num.map.output.key.fields=1 \
    -input /tmp/data/20news-all/alt.atheism \
    -output wiki_wc_bash \
    -mapper ./mapwc.sh -reducer ./reducewc.sh \
    -file ./mapwc.sh -file ./reducewc.sh
```

Note, with Hadoop streaming, you could use higher number of mapper.

Why?

Parallelism with HDFS / Hadoop

- How many mapper and reducer YARN will start?

- Based on user request
  - mapred.map.tasks #number of map instances to be generated.
  - mapred.reduce.tasks # number of reduce instances to be used.

- But also limited by
  - Data split
  - Computational resources

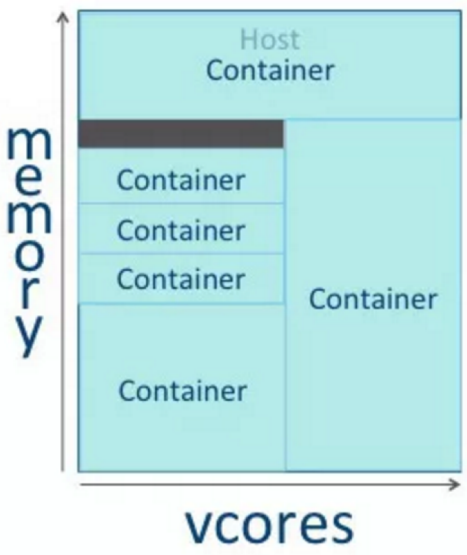
Parallelism with HDFS / Hadoop : Data Split

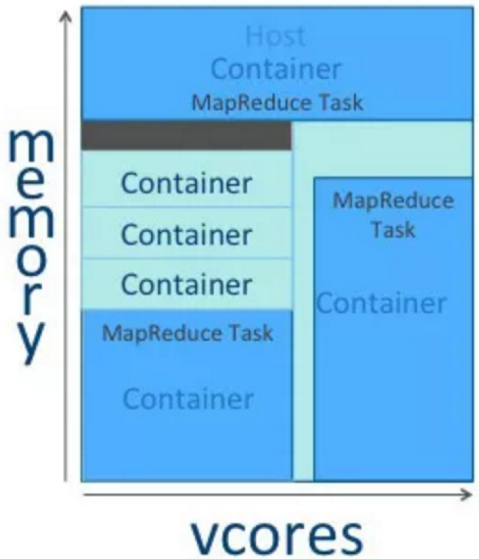
- The number of actual mapper created may be limited by the number of actual data blocks in the hdfs.
  - e.g. a file with 35GB with default block size as 128MB, the file will store in 266 blocks in hdfs.
  - All mappers may not be run at the same time, if there is not enough resources.
- The number of reducer maybe limited by the number of unique keys.
- Each reducer will generate an output file independent from each other.
  - So 256 reducer would result 256 files in the output folder.

Parallelism with HDFS / Hadoop : Resource Availability

- Data split may determine the maximum number of mapper and reducer a job may have.

- The actual number of concurrent mapper and reducer may determined by the available resources
  - Based on number of cores needed
  - Size of the memory requested.





Many Relevant Cluster/Job Settings Matters

# of mapper	mapred.map.tasks
# of reducers	mapred.reduce.tasks
# of core-per-container	yarn.nodemanager.resource.cpu-vcores
	yarn.scheduler.minimum-allocation-vcores
	yarn.scheduler.maximum-allocation-vcores
Memory per container	yarn.nodemanager.resource.memory-mb
	yarn.scheduler.minimum-allocation-mb
Memory per executors	yarn.scheduler.maximum-allocation-mb
	yarn.scheduler.increment-allocation-mb
Memory per mapper	
Memory per reducers	
…	

How to get started

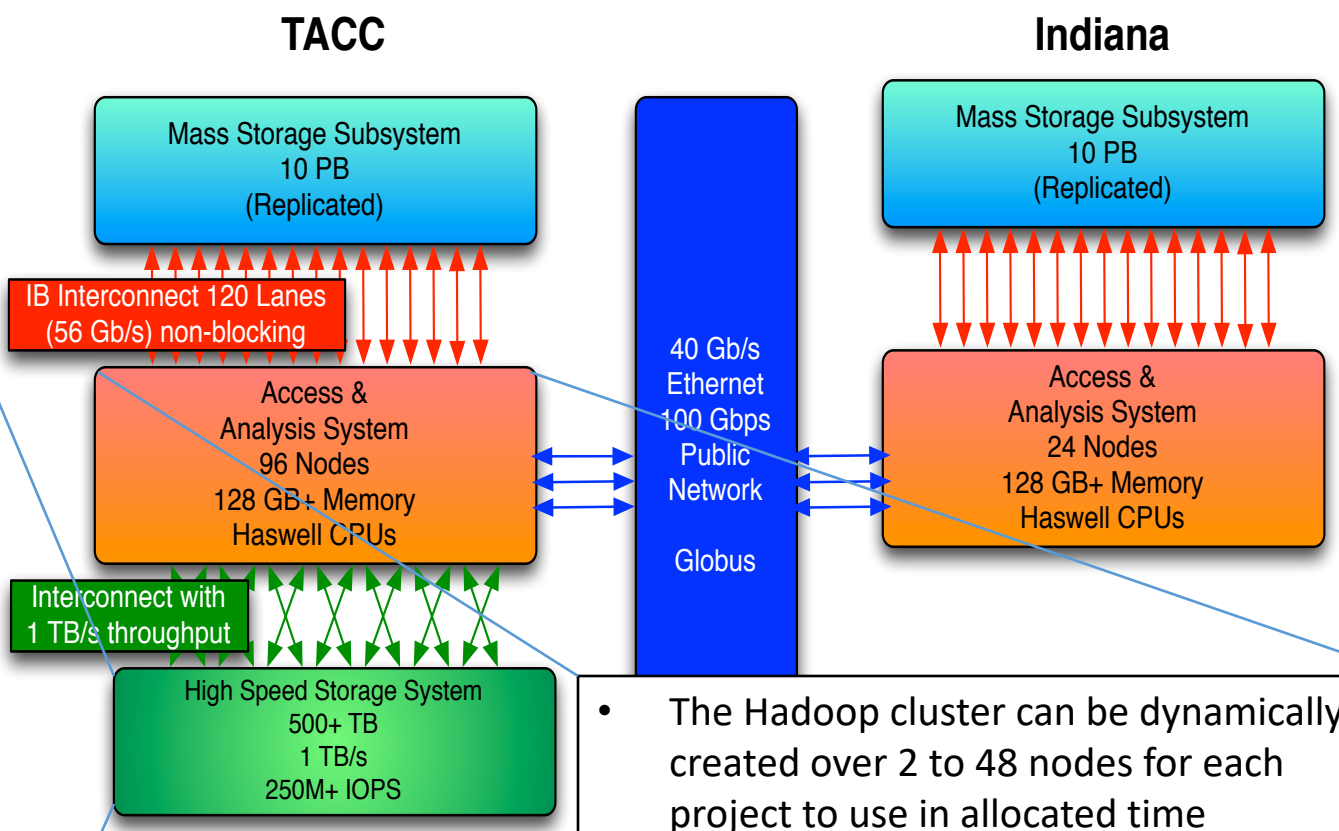
- Local Installation
  - Hadoop cluster can fully functional on your laptop
  - The code runs on your local version will run on real cluster
  - Just try to avoid large file…

- Cluster Installation
  - Install locally,
    - requires a minimum of two nodes.
    - Recommended minimum is four nodes.
  - Commercial cloud
    - Amazon, Google
  - Use TACC

Try Hadoop Locally

- Main referenence
  - https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html
- Prerequisite
  - Java 8
  - SSH
- Download
  - e.g from http://apache.claz.org/hadoop/common/stable/
- Unpack
- Try bin/hadoop

Hadoop Support at TACC: Wrangler



- A direct attached PCI interface allows access to the NAND flash.
- Not limited by networking connection
- Flash storage not tied to individual nodes

- The Hadoop cluster can be dynamically created over 2 to 48 nodes for each project to use in allocated time
- Each node have access to 4 TB flash storage across four channels
- Accessible via the Hadoop cluster via idev, batch job submission and VNC sessions.

Get Started with Hadoop on Wrangler

- Step 1: create a Hadoop reservation through Wrangler data portal,
  - portal.wrangler.tacc.utexas.edu
  - What do you need?
    - Any web browser

- Step 2: Access your Hadoop cluster and submit jobs
  - ssh username@wrangler.tacc.utexas.edu
  - What do you need?
    - Secure Shell Client
    - Any VNC client

Get Started with Hadoop on Wrangler

- Log on to the wrangler cluster using SSH
  - ssh trainXXX@wrangler.tacc.utexas.edu
- Submit a job to gain access to the Hadoop reservation
  - idev -N 1 -n 1 -r hadoop+TRAINING-OPEN+2552
- Some exercise example
  - https://goo.gl/Dg6YNr