Parallel Data Processing with MapReduce

293S Tao Yang, 2017

Overview

- **What is MapReduce?**
  - Example with word counting
- **Parallel data processing with MapReduce**
  - Hadoop file system
- **More application example**

Motivations

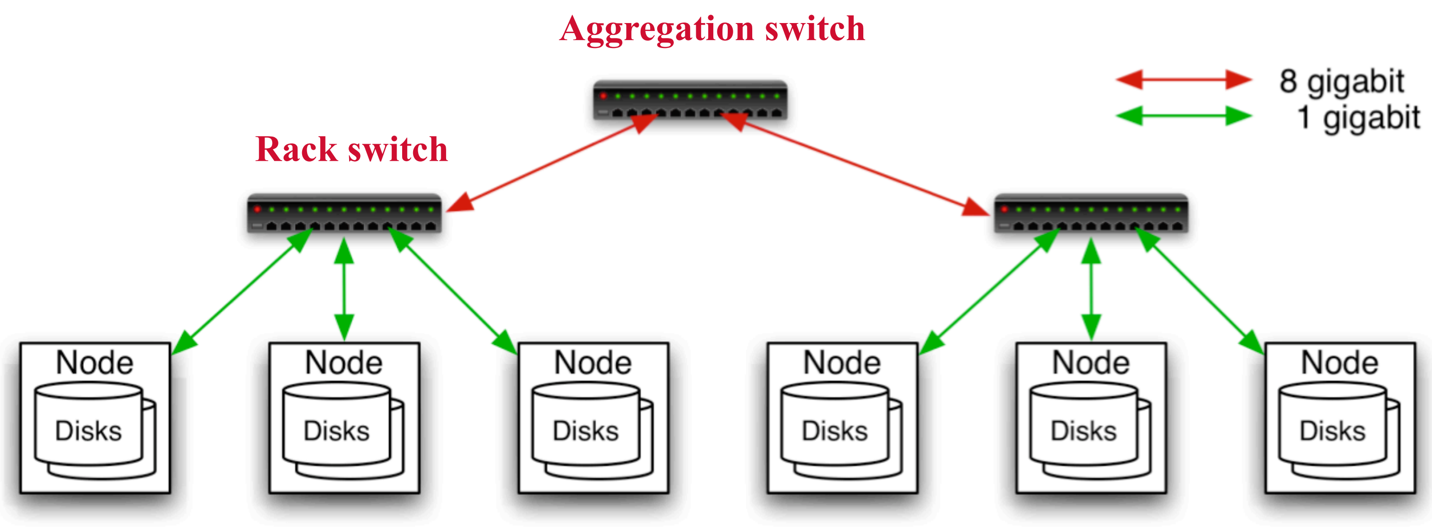
- **Motivations**
  - Large-scale data processing on clusters
  - Massively parallel (hundreds or thousands of CPUs)
  - Reliable execution with easy data access
- **Functions**
  - Automatic parallelization & distribution
  - Fault-tolerance
  - Status and monitoring tools
  - A clean abstraction for programmers
    - Functional programming meets distributed computing
    - A batch data processing system

Parallel Data Processing in a Cluster

- **Scalability to large data volumes:**
  - Scan 1000 TB on 1 node @ 100 MB/s = 24 days
  - Scan on 1000-node cluster = 35 minutes

- **Cost-efficiency:**
  - Commodity nodes /network
    - » Cheap, but not high bandwidth, sometime unreliable
  - Automatic fault-tolerance (fewer admins)
  - Easy to use (fewer programmers)

Typical Hadoop Cluster



- 40 nodes/rack, 1000-4000 nodes in cluster
- 1 Gbps bandwidth in rack, 8 Gbps out of rack
- Node specs :
  8-16 cores, 32 GB RAM, 8 × 1.5 TB disks

MapReduce Programming Model

- Inspired from map and reduce operations commonly used in functional programming languages like Lisp.
- Have multiple map tasks and reduce tasks
- Users implement interface of two primary methods:
  - Map: (key1, val1) → (key2, val2)
  - Reduce: (key2, [val2 list]) → [val3]

Inspired by LISP Function Programming

- **Two Lisp functions**
- **Lisp *map* function**
  - Input parameters: a function and a set of values
  - This function is applied to each of the values.

  Example:
  - (map ‘length ‘(() (a) (ab) (abc)))

  →(length(()) length(a) length(ab) length(abc))

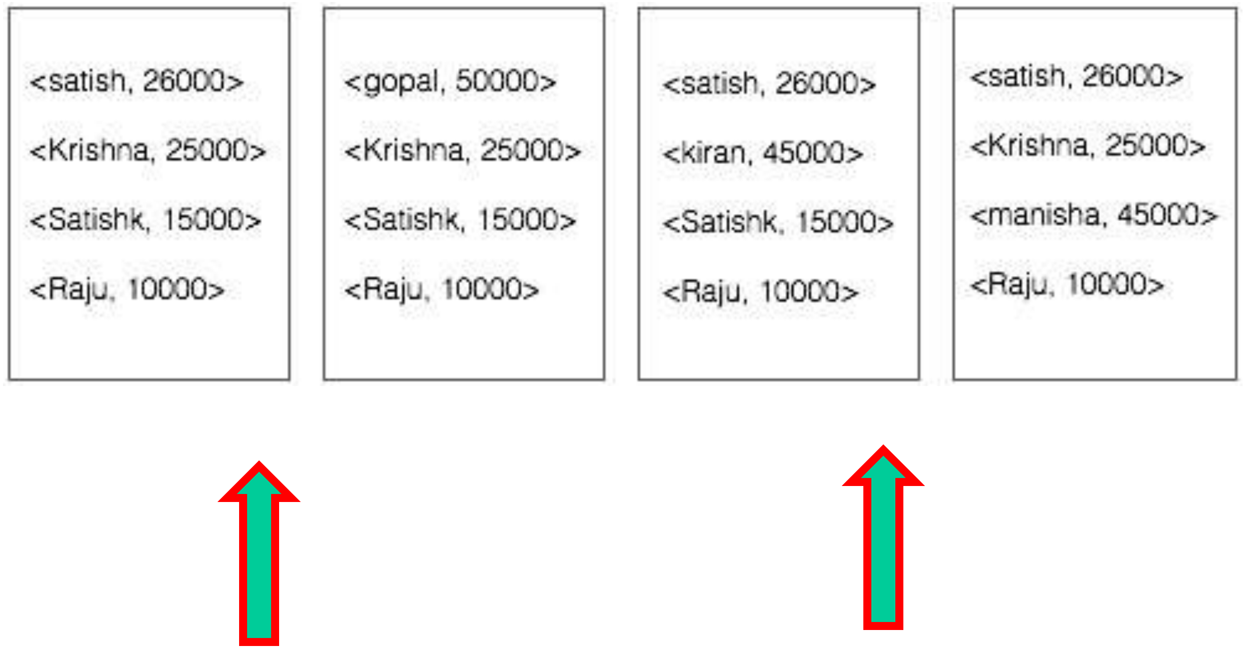
  → (0 1 2 3)

Lisp Reduce Function

- Lisp *reduce* function
  - given a binary function and a set of values.
  - It combines all the values together using the binary function.
- Example:
  - use the + (add) function to reduce the list (0 1 2 3)
  - (reduce #'+ '(0 1 2 3))

  → 6

Map/Reduce Tasks for World of Key-Value Pairs



<satish, 26000>
<Krishna, 25000>
<Satishk, 15000>
<Raju, 10000>

<gopal, 50000>
<Krishna, 25000>
<Satishk, 15000>
<Raju, 10000>

<satish, 26000>
<kiran, 45000>
<Satishk, 15000>
<Raju, 10000>

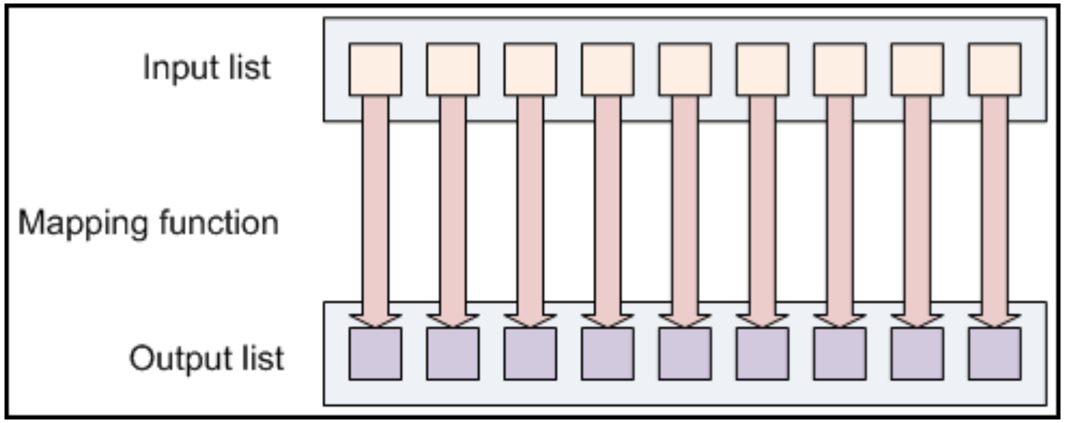
<satish, 26000>
<Krishna, 25000>
<manisha, 45000>
<Raju, 10000>

Map Tasks

Reduce Tasks

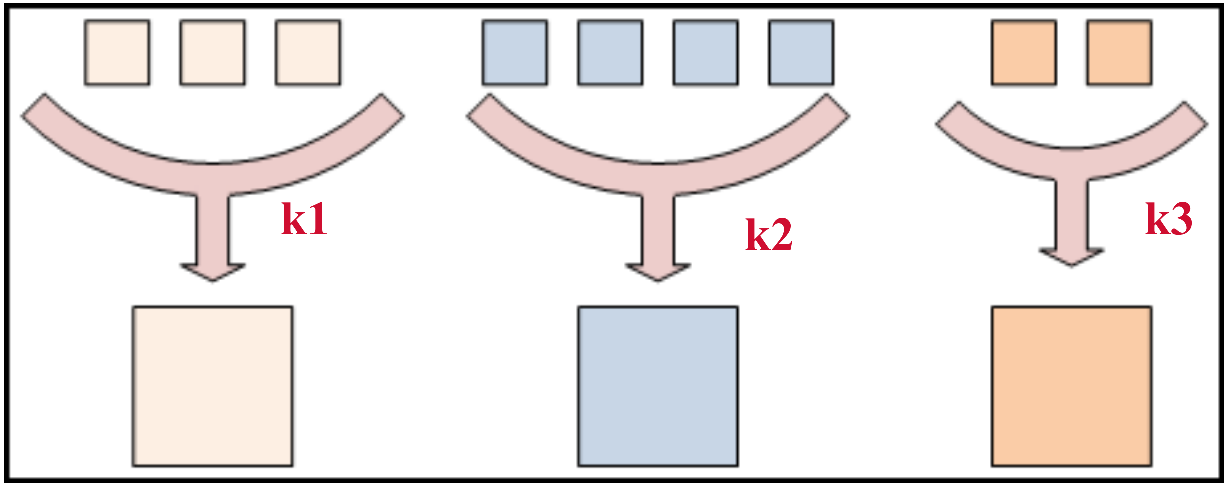
Example: Map Processing in Hadoop

- **Given a file**
  - A file may be divided by the system into multiple parts (called splits or shards).
- **Each record in a split is processed by a user Map function,**
  - takes each record as an input
  - produces key/value pairs

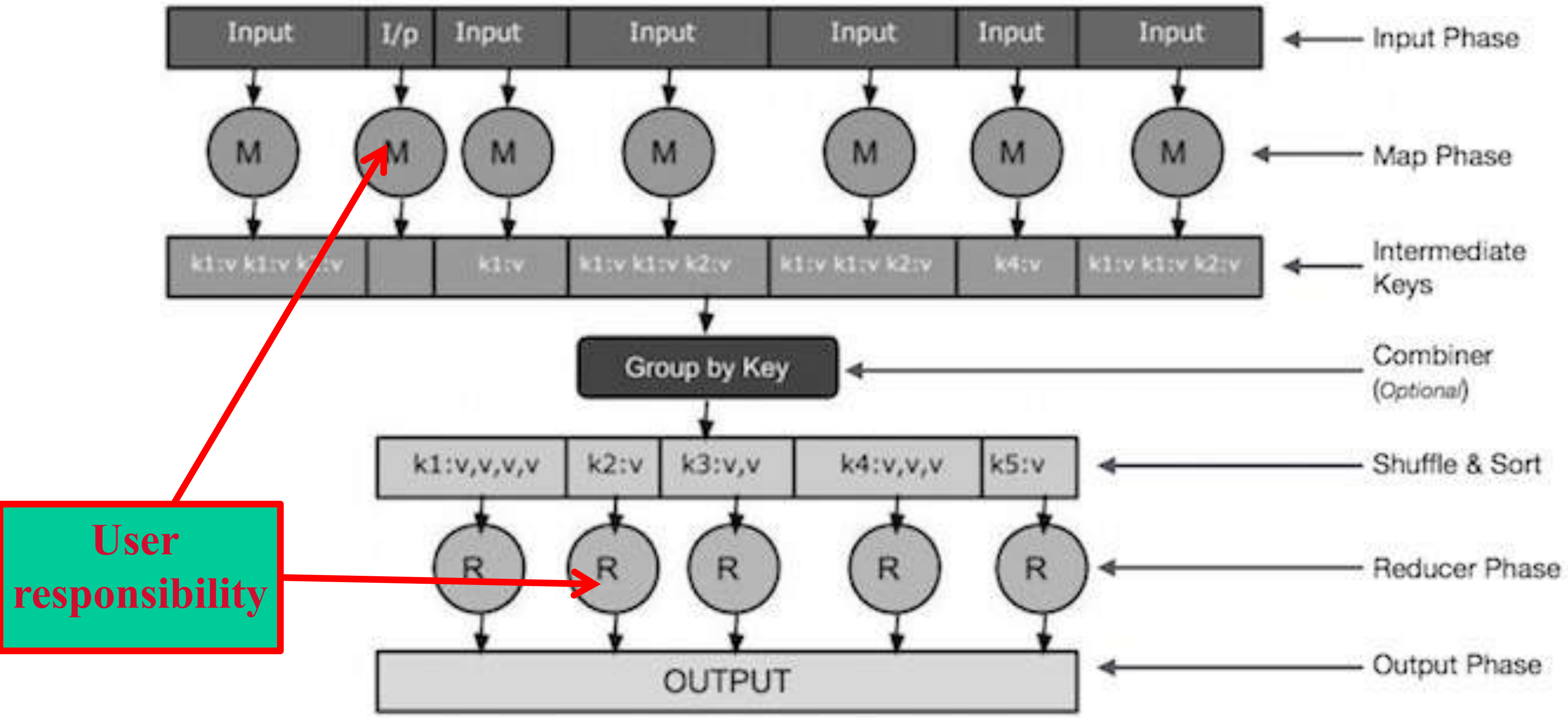


Processing of Reducer Tasks

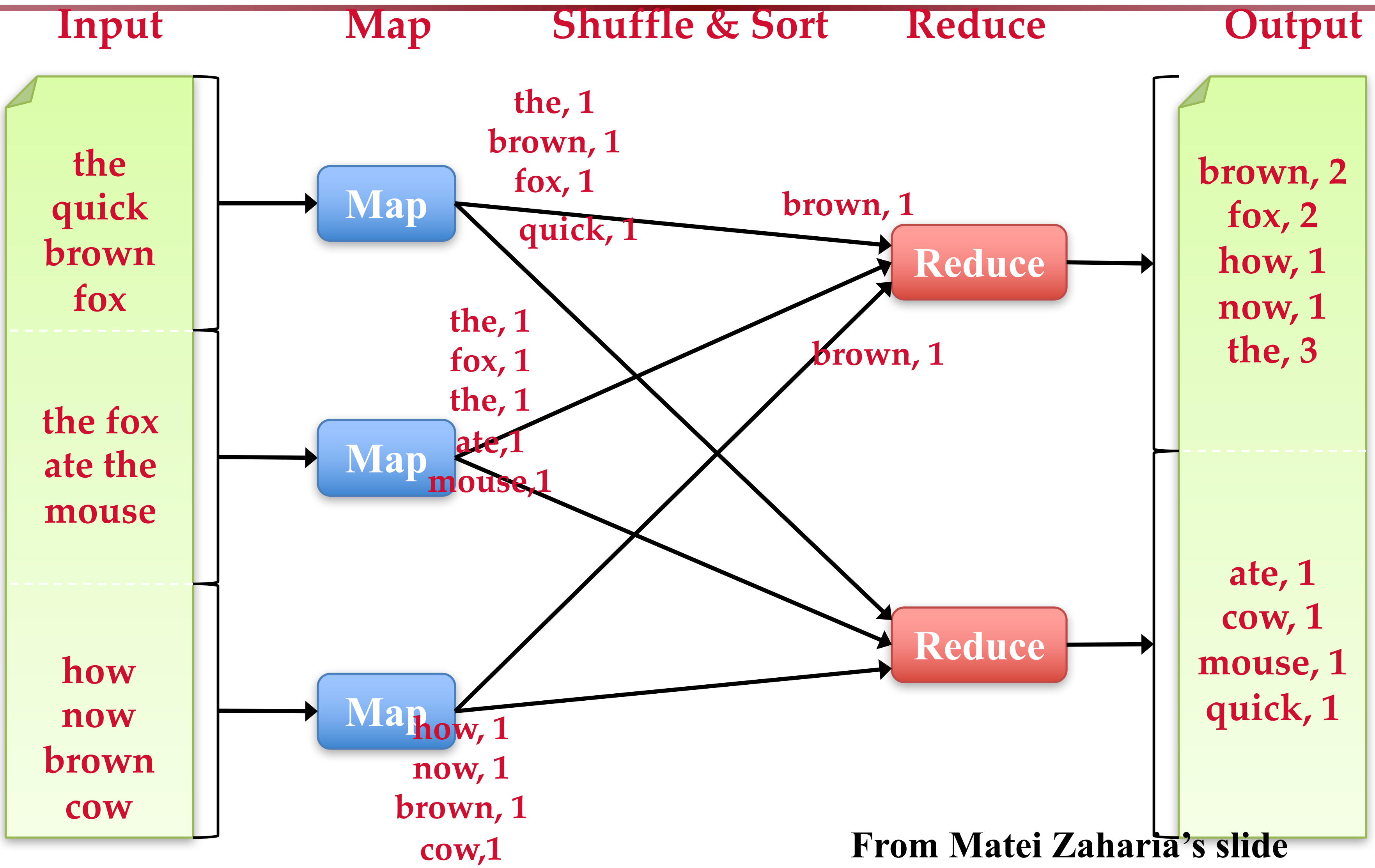
- **Given a set of (key, value) records produced by map tasks.**
  - all the intermediate values for a key are combined together into a list and given to a reducer. Call it [val2]
  - A user-defined function is applied to each list [val2]   and produces another value



Put Map and Reduce Tasks Together



Example of Word Count Job (WC)



From Matei Zaharia's slide

Input/output specification of the WC mapreduce job

Input : a set of (key values) stored in files

key: document ID

value: a list of words as content of each document

Output: a set of (key values) stored in files

key: wordID

value: word frequency appeared in all documents

MapReduce function specification:

```
map(String input_key, String input_value):
 reduce(String output_key, Iterator intermediate_values):
```

Pseudo-code

```
map(String input_key, String input_value):
// input_key: document name
// input_value: document contents
    for each word w in input_value:
      EmitIntermediate(w, "1");

reduce(String output_key, Iterator intermediate_values):
// output_key: a word
// output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
      result  = result + ParseInt(v);
    Emit(output_key, AsString(result));
```

MapReduce WordCount.java

Hadoop distribution: **src/examples/org/apache/hadoop/examples/WordCount.java**

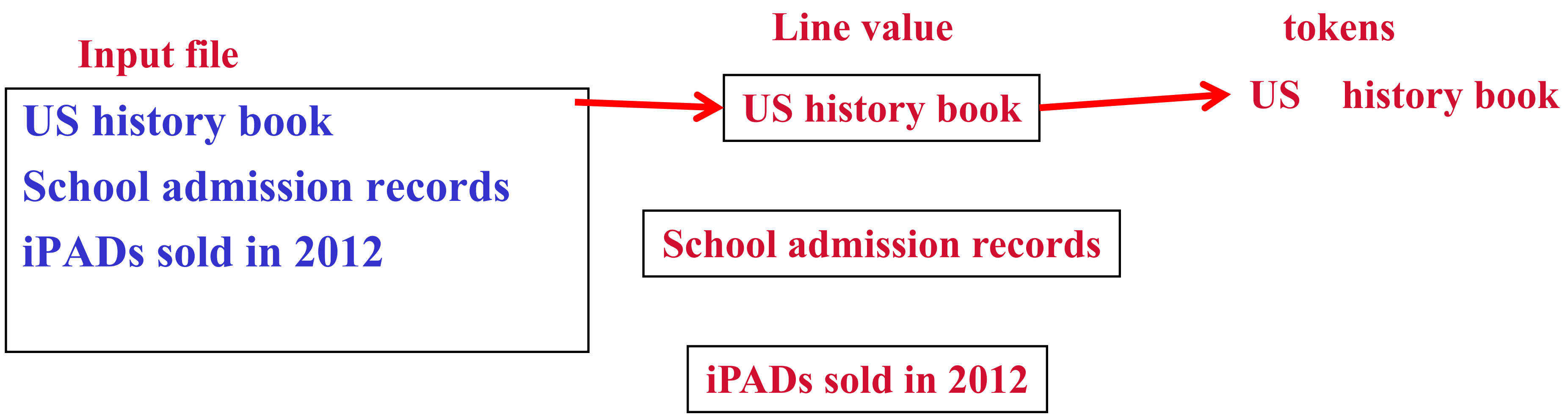
```
public static class TokenizerMapper
     extends Mapper<Object, Text, Text, IntWritable>{

  private final static IntWritable one = new IntWritable(1); // a mapreduce int class
  private Text word = new Text(); //a mapreduce String  class

  public void map(Object key, Text value, Context context
                  ) throws IOException, InterruptedException { // key is the offset of
 current record in a file
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) { // loop for each token
          word.set(itr.nextToken());  //convert from string to token
         context.write(word, one);  // emit (key,value) pairs for reducer
    }
```

MapReduce WordCount.java

map() gets a key, value, and context

- key - "bytes from the beginning of the line?“
- value - the current line;

in the while loop, each token is a "word" from the current line



Reduce code in WordCount.java

```
public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);  //convert  "int" to IntWritable
      context.write(key, result); //emit the final key-value result
    }
```

The driver to set things up and start

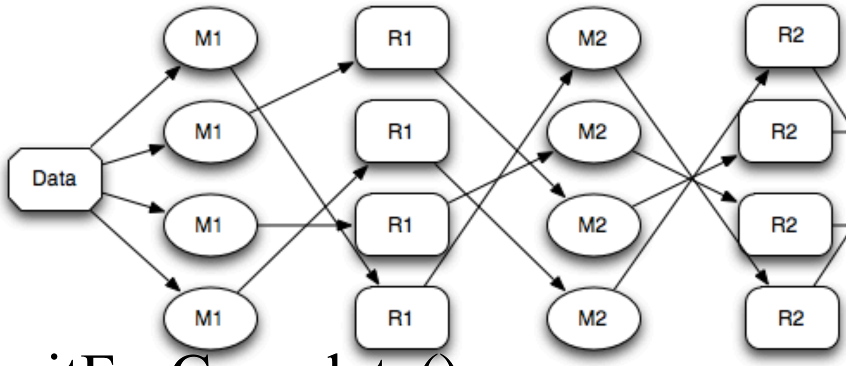
```
//   Usage: wordcount <in> <out>
 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    Job job = new Job(conf, "word count"); //mapreduce job
    job.setJarByClass(WordCount.class); //set jar file
    job.setMapperClass(TokenizerMapper.class); // set mapper class
    job.setCombinerClass(IntSumReducer.class); //set combiner class
    job.setReducerClass(IntSumReducer.class);   //set reducer class
    job.setOutputKeyClass(Text.class);                // output key class
    job.setOutputValueClass(IntWritable.class);  //output value class
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));   //job input path
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1])); //job output path
    System.exit(job.waitForCompletion(true) ? 0 : 1);  //exit status
```

Types of MapReduce Applications

- **Map only parallel processing**
  - Count word usage for each document
- **Map-reduce two-stage processing**
  - Count word usage for the entire document collection
- **Multiple map-reduce stages**
  1. Count word usage in a document set
  2. Identify most frequent words in each document, but exclude those most popular words in the entire document set

MapReduce Job Chaining

- Run a sequence of map-reduce jobs



- Use job.waitForComplete()
  - Define the first job including input/output directories, and map/combiner/reduce classes.
    - » Run the first job with job.waitForComplete()
  - Define the second job
    - » Run the second job with job.waitForComplete()
- Use JobClient.runJob(job)

Example

```
Job job = new Job(conf, "word count"); //mapreduce job
   job.setJarByClass(WordCount.class); //set jar file
   job.setMapperClass(TokenizerMapper.class); // set mapper class
   ...
   FileInputFormat.addInputPath(job, new Path(otherArgs[0]));   // input path
   FileOutputFormat.setOutputPath(job, new Path(otherArgs[1])); // output path
   job.waitForCompletion(true) ;
Job job1 = new Job(conf, "word count"); //mapreduce job
   job1.setJarByClass(WordCount.class); //set jar file
   job1. setMapperClass(TokenizerMapper.class); // set mapper class
   ...
   FileInputFormat.addInputPath(job1, new Path(otherArgs[1]));   // input path
   FileOutputFormat.setOutputPath(job1, new Path(otherArgs[2])); // output path
    System.exit(job1.waitForCompletion(true) ? 0 : 1);  //exit status
}
```

MapReduce Use Case: Inverted Indexing Preliminaries

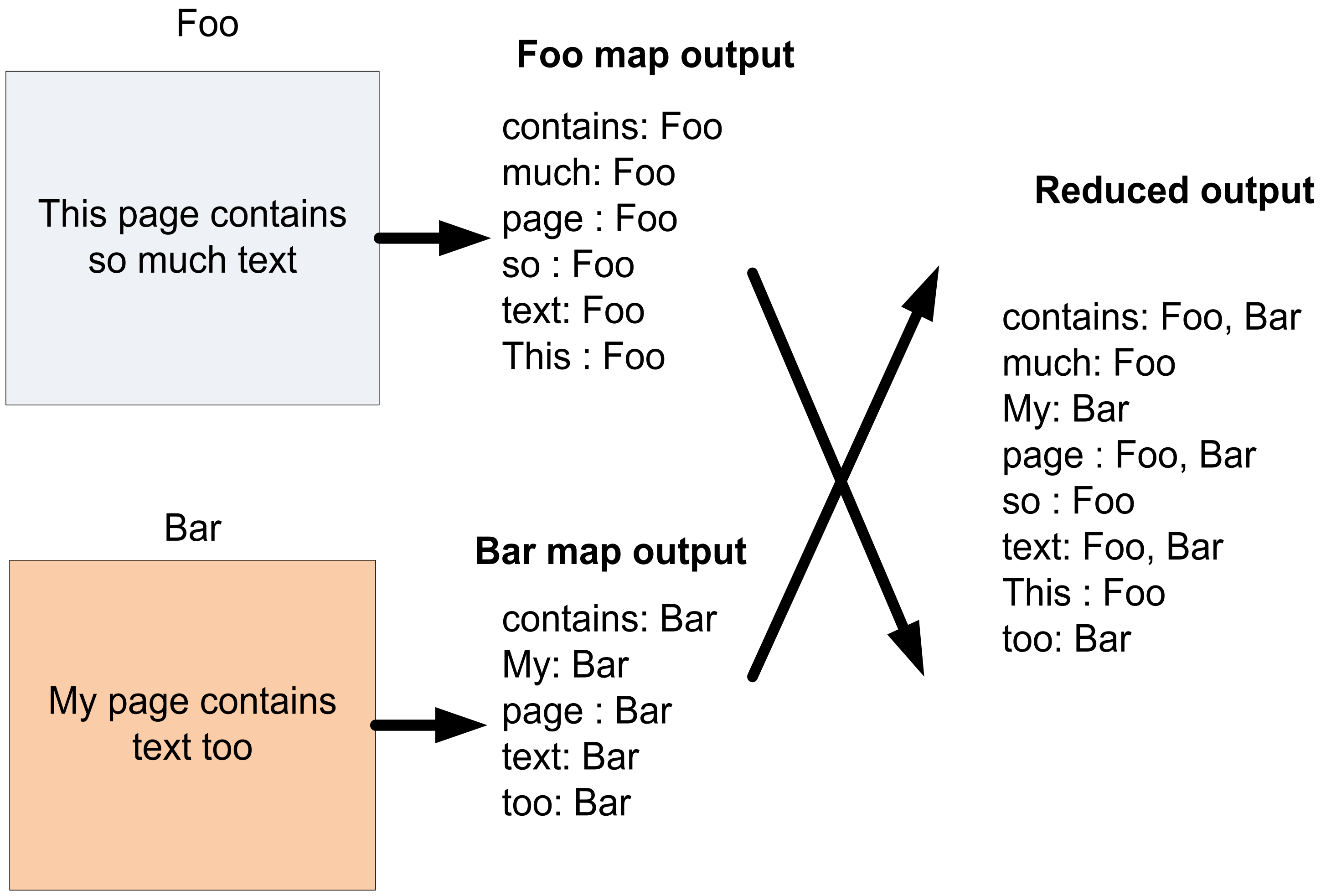
Construction of inverted lists for document search

- Input: documents: (docid, [term, term..]), (docid, [term, ..]), ..
- Output: (term, [docid, docid, …])
  - E.g., (apple, [1, 23, 49, 127, …])

A document id is an internal document id, e.g., a unique integer

- Not an external document id such as a url

Inverted Indexing: Data flow



Foo

This page contains so much text

Foo map output

contains: Foo
much: Foo
page : Foo
so : Foo
text: Foo
This : Foo

Bar

My page contains text too

Bar map output

contains: Bar
My: Bar
page : Bar
text: Bar
too: Bar

Reduced output

contains: Foo, Bar
much: Foo
My: Bar
page : Foo, Bar
so : Foo
text: Foo, Bar
This : Foo
too: Bar

Using MapReduce to Construct Inverted Indexes

- **Each Map task is a document parser**
  - Input: A stream of documents
  - Output: A stream of (term, docid) tuples
    - » (long, 1) (ago, 1) (and, 1) … (once, 2) (upon, 2) …
    - » We may create internal IDs for words.
- **Shuffle sorts tuples by key and routes tuples to Reducers**
- **Reducers convert streams of keys into streams of inverted lists**
  - Input: (long, 1) (long, 127) (long, 49) (long, 23) …
  - The reducer sorts the values for a key and builds an inverted list
  - Output: (long, [frequency:492, docids:1, 23, 49, 127, …])

Using Combiner () to Reduce Communication

- **Map:** (docid$_1$, content$_1$) → (t$_1$, ilist$_{1,1}$) (t$_2$, ilist$_{2,1}$) (t$_3$, ilist$_{3,1}$) …
  - Each output inverted list covers just one document
- **Combine locally**

  Sort by t

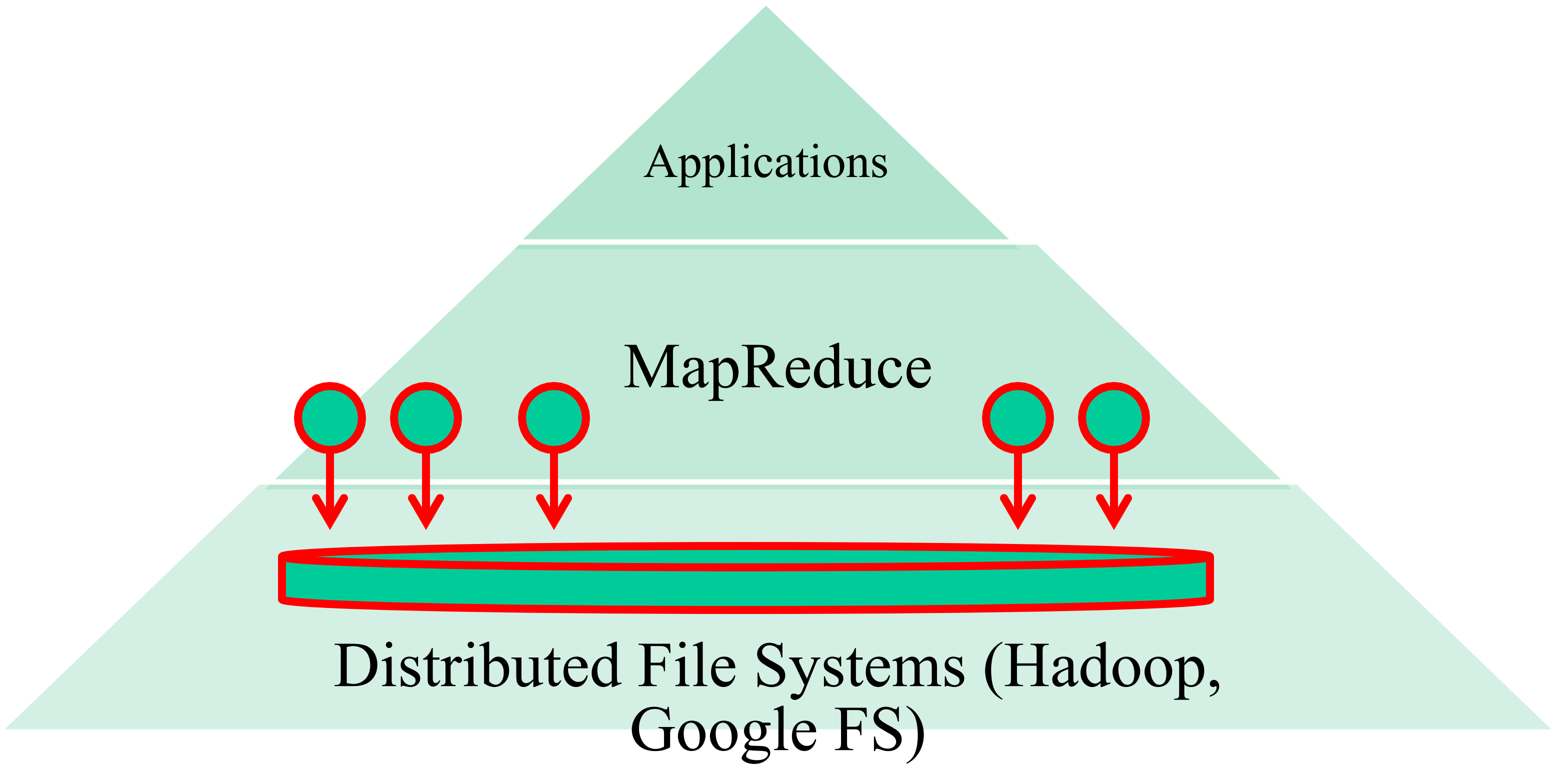
  Combiner: (t$_1$ [ilist$_{1,2}$ ilist$_{1,3}$ ilist$_{1,1}$ …]) → (t$_1$, ilist$_{1,27}$)
  - Each output inverted list covers a sequence of documents
- **Shuffle and sort** by t

  (t$_4$, ilist$_{4,1}$) (t$_5$, ilist$_{5,3}$) … → (t$_4$, ilist$_{4,2}$) (t$_4$, ilist$_{4,4}$) (t$_4$, ilist$_{4,1}$) …

- **Reduce:** (t$_7$, [ilist$_{7,2}$, ilist$_{3,1}$, ilist$_{7,4}$, …]) → (t$_7$, ilist$_{final}$)

ilist$_{i,j}$: the j'th inverted list fragment for term i

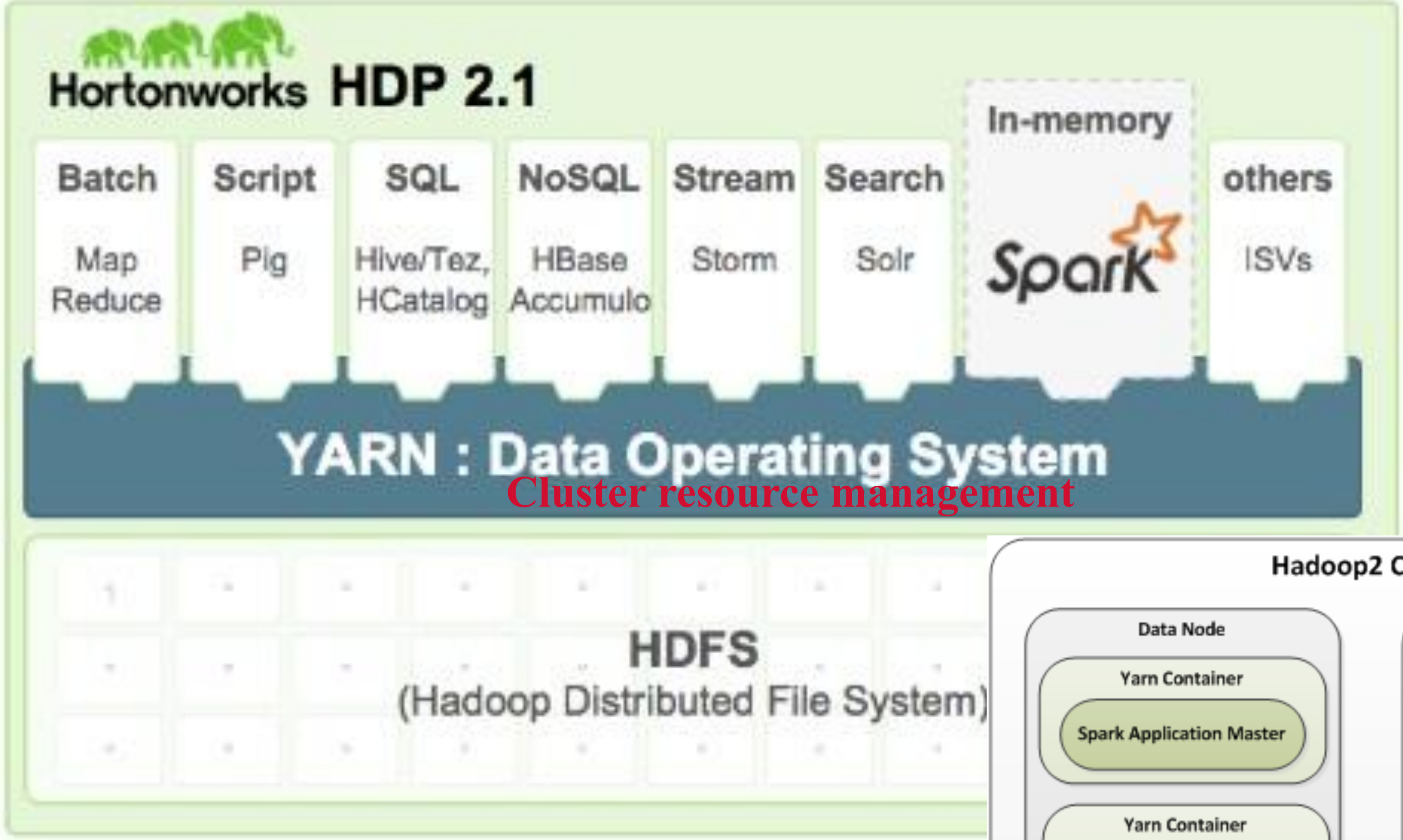
Systems Support for MapReduce



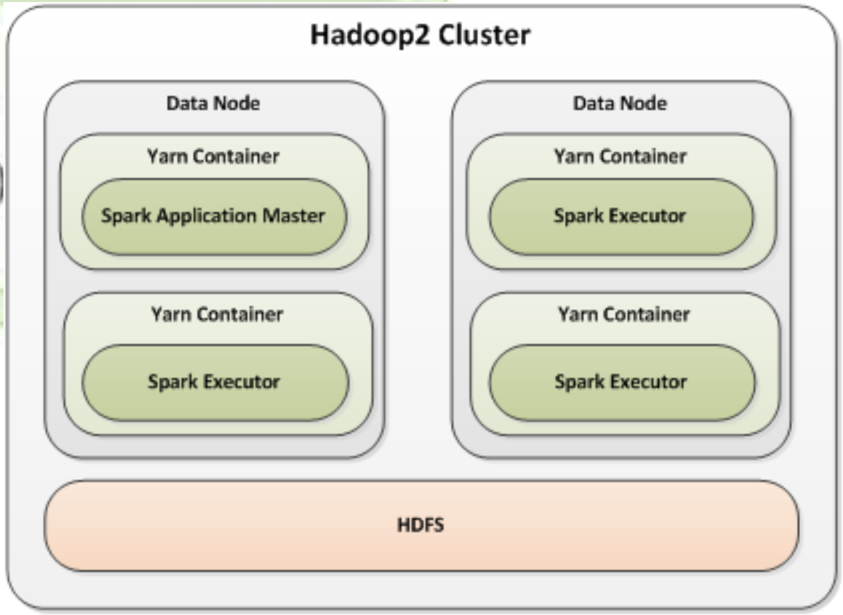
Hadoop and Tools

- **Various Linux Hadoop clusters**
  - Cluster +Hadoop: http://hadoop.apache.org
  - Amazon EC2
- **Windows and other platforms**
  - The NetBeans plugin simulates Hadoop
  - The workflow view works on Windows
- **Hadoop-based tools**
  - For Developing in Java, NetBeans plugin
- **Pig Latin,** a SQL-like high level data processing script language
- **Hive,** Data warehouse, SQL
- **HBase,** Distributed data store as a large table

New Hadoop Develpment



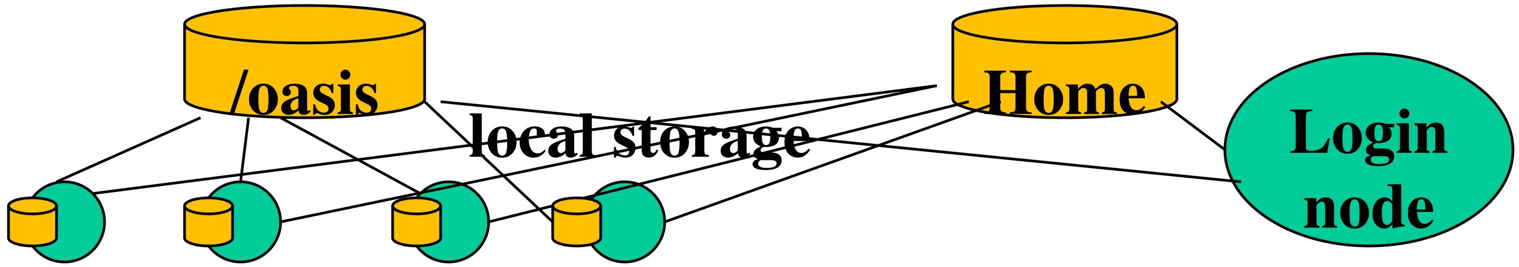
Cluster resource management



Comet Cluster

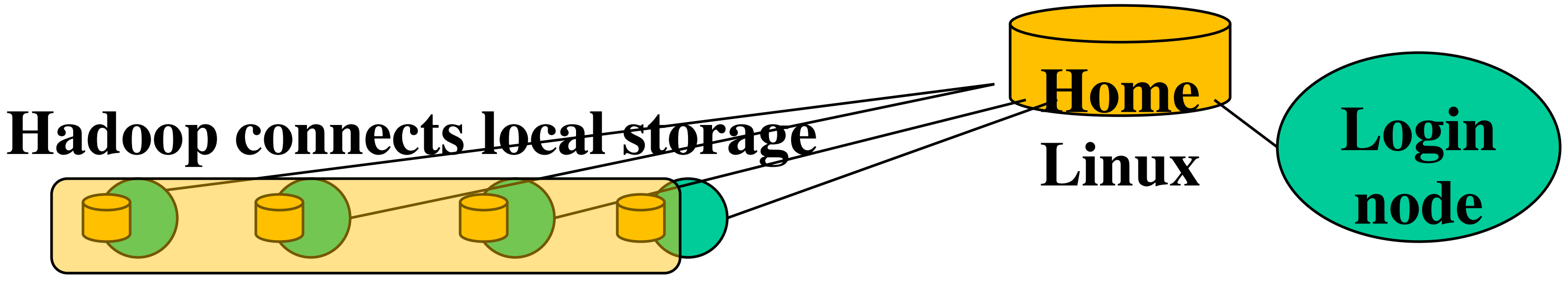
- Comet cluster has 1944 nodes and each node has 24 cores, built on two 12-core Intel Xeon E5-2680v3 2.5 GHz processors
- 128 GB memory and 320GB SSD for local scratch space.
- Attached storage: Shared 7 petabytes of 200 GB/second performance storage and 6 petabytes of 100 GB/second durable storage
  - Lustre Storage Area is a Parallel File System (PFS) called Data Oasis.
    - Users can access from

      /oasis/scratch/comet/$USER/temp_project



Hadoop installation at Comet

- *Installed in /opt/hadoop/1.2.1*
- **Configure Hadoop on-demand with myHadoop:**
  - /opt/hadoop/contrib/myHadoop/bin/myhadoop-configure.sh



Hadoop file system is built dynamically on the nodes allocated. Deleted when the allocation is terminated.

Shell Commands for Hadoop File System

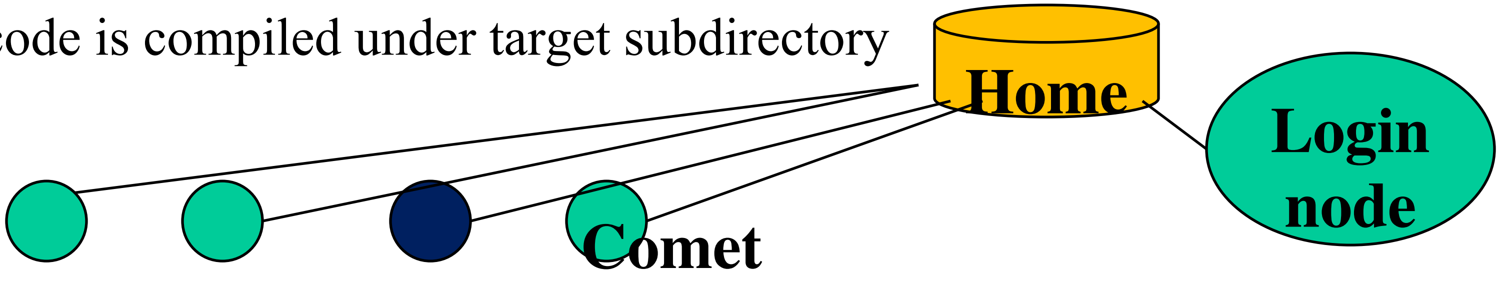
- **Mkdir, ls, cat, cp**
  - hadoop dfs -mkdir /user/deepak/dir1
  - hadoop dfs -ls /user/deepak
  - hadoop dfs -cat /usr/deepak/file.txt
  - hadoop dfs -cp /user/deepak/dir1/abc.txt /user/deepak/dir2
- **Copy data from the local file system to HDF**
  - hadoop dfs -copyFromLocal <src:localFileSystem> <dest:Hdfs>
  - Ex: hadoop dfs –copyFromLocal /home/hduser/def.txt /user/deepak/dir1
- **Copy data from HDF to local**
  - hadoop dfs -copyToLocal <src:Hdfs> <dest:localFileSystem>

http://www.bigdataplanet.info/2013/10/All-Hadoop-Shell-Commands-you-need-Hadoop-Tutorial-Part-5.html

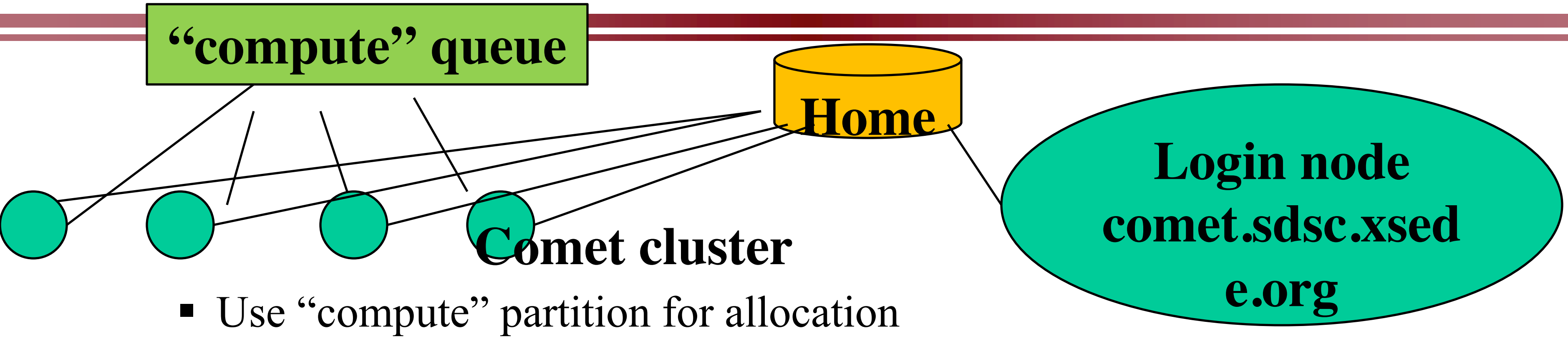
Compile the sample Java code at Comet

Java word count example is available at Comet under
/home/tyang/cs240sample/mapreduce/.

- **cp –r /home/tyang/cs240sample/mapreduce .**
- **Allocate a dedicated machine for compiling**
  - /share/apps/compute/interactive/qsubi.bash -p compute --nodes=1 --ntasks-per-node=1 -t 00:
- **Change work directory to mapreduce and type make**
  - Java code is compiled under target subdirectory



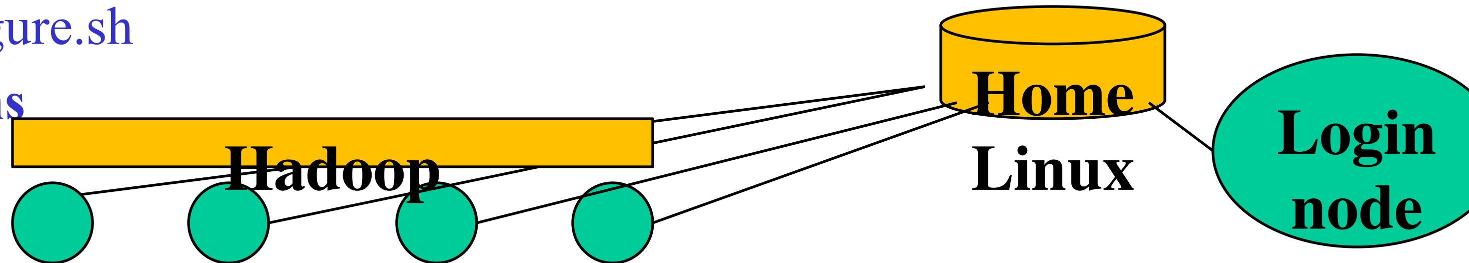
How to Run a WordCount Mapreduce Job



- Use "compute" partition for allocation
- Use Java word count example at Comet under /home/tyang/cs240sample/mapreduce/.
- sbatch submit-hadoop-comet.sh
  - » Data input is in test.txt
  - » Data output is in WC-output
- Job trace is wordcount.1569018.comet-17-14.out

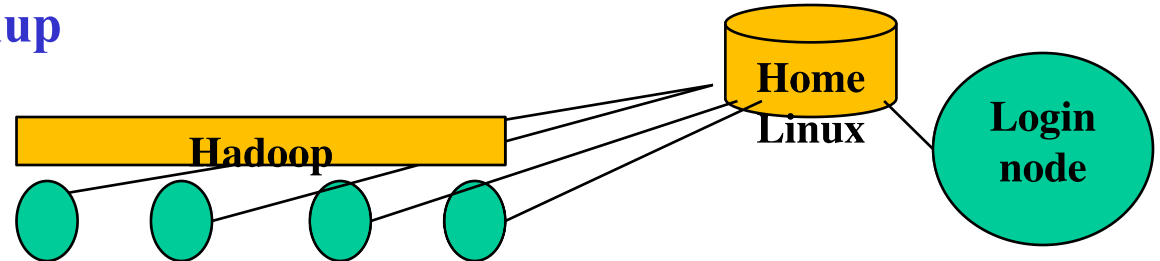
Sample script (submit-hadoop-comet.sh)

```
#!/bin/bash
#SBATCH --job-name="wordcount"
#SBATCH --output="wordcount.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
#SBATCH -t 00:15:00
Export HADOOP_CONF_DIR=/home/$USER/cometcluster
export WORKDIR=`pwd`
module load hadoop/1.2.1

#Use myheadoop to build a Hadoop file system on allocated nodes
myhadoop-configure.sh
#Start all demons
start-all.sh
```



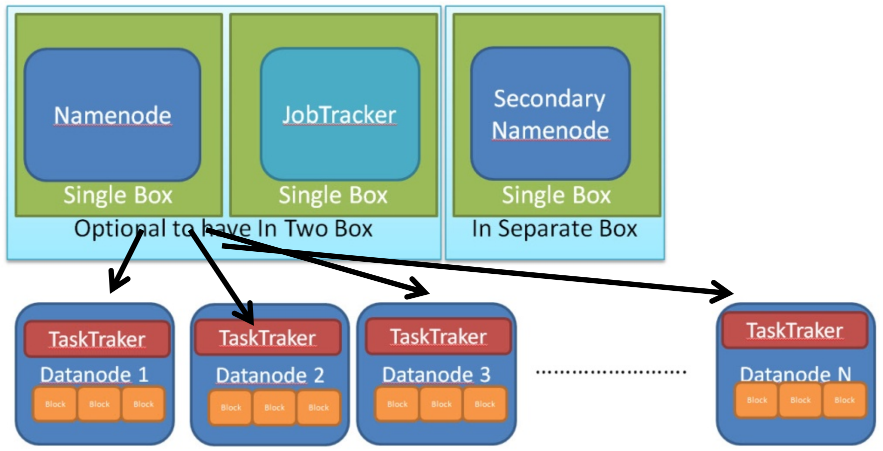
Sample script

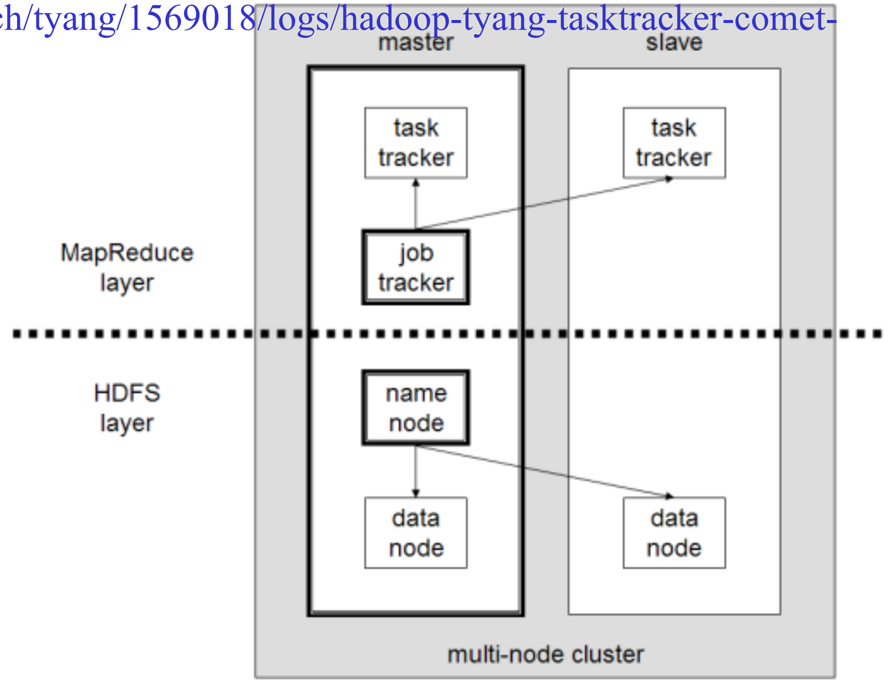
```
#make an input directory in the hadoop file system
hadoop dfs -mkdir input
#copy data from local Linux file system to the Hadoop file system
hadoop dfs -copyFromLocal $WORKDIR/test.txt input/
#Run Hadoop wordcount job
hadoop jar $WORKDIR/wordcount.jar  wordcount input/ output/
# Create a local directory WC-output to host the output data
# It does not report error even the file does not exist
rm -rf WC-out >/dev/null || true
mkdir -p WC-out
# Copy out the output data
hadoop dfs -copyToLocal output/part* WC-out
#Stop all demons and cleanup
stop-all.sh
myhadoop-cleanup.sh
```



Sample output trace wordcount.1569018.comet-17-14.out

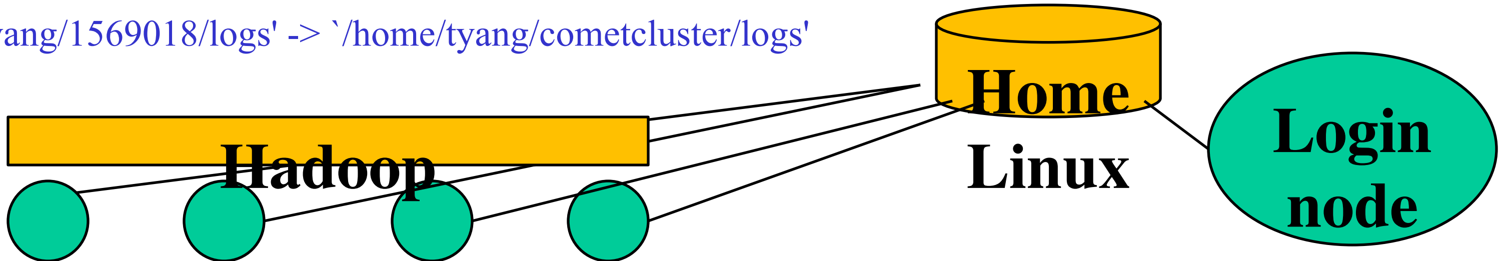
```
starting namenode, logging to /scratch/tyang/1569018/logs/hadoop-tyang-namenode-comet-17-14.out
comet-17-14.ibnet: starting datanode, logging to /scratch/tyang/1569018/logs/hadoop-tyang-datanode-comet-17-
    14.sdsc.edu.out
comet-17-15.ibnet: starting datanode, logging to /scratch/tyang/1569018/logs/hadoop-tyang-datanode-comet-17-
    15.sdsc.edu.out
comet-17-14.ibnet: starting secondarynamenode, logging to /scratch/tyang/1569018/logs/hadoop-tyang-
    secondarynamenode-comet-17-14.sdsc.edu.out
starting jobtracker, logging to /scratch/tyang/1569018/logs/hadoop-tyang-jobtracker-comet-17-14.out
comet-17-14.ibnet: starting tasktracker, logging to /scratch/tyang/1569018/logs/hadoop-tyang-tasktracker-comet-
    17-14.sdsc.edu.out
comet-17-15.ibnet: starting tasktracker, logging to /scratch/tyang/1569018/logs/hadoop-tyang-tasktracker-comet-
    17-15.sdsc.edu.out
```





Sample output trace wordcount.1569018.comet-17-14.out

```
16/01/31 17:43:44 INFO input.FileInputFormat: Total input paths to process : 1
16/01/31 17:43:44 INFO util.NativeCodeLoader: Loaded the native-hadoop library
16/01/31 17:43:44 WARN snappy.LoadSnappy: Snappy native library not loaded
16/01/31 17:43:44 INFO mapred.JobClient: Running job: job_201601311743_0001
16/01/31 17:43:45 INFO mapred.JobClient:  map 0% reduce 0%
16/01/31 17:43:49 INFO mapred.JobClient:  map 100% reduce 0%
16/01/31 17:43:56 INFO mapred.JobClient:  map 100% reduce 33%
16/01/31 17:43:57 INFO mapred.JobClient:  map 100% reduce 100%
16/01/31 17:43:57 INFO mapred.JobClient: Job complete: job_201601311743_0001
comet-17-14.ibnet: stopping tasktracker
comet-17-15.ibnet: stopping tasktracker
stopping namenode
comet-17-14.ibnet: stopping datanode
comet-17-15.ibnet: stopping datanode
comet-17-14.ibnet: stopping secondarynamenode
Copying Hadoop logs back to /home/tyang/cometcluster/logs...
`/scratch/tyang/1569018/logs' -> `/home/tyang/cometcluster/logs'
```



Sample input and output

```
$ cat test.txt
how are you today 3 4 mapreduce program
1 2 3 test send
how are you  mapreduce
1    send test USA california new

$ cat WC-out/part-r-00000
1       2
2       1
3       2
4       1
USA     1
are     2
california      1
how     2
mapreduce       2
new     1
program 1
send    2
test    2
today   1
you     2
```

Notes

- **Java process listing "jps", shows the following demons**

  NameNode (master), SecondaryNameNode, Datanode (hadoop),JobTracker, TaskTracker
- **To check the status of your job**

  squeue -u username
- **To cancel a submitted job**

  scancel job-id
- You have to request *all* 24 cores on the nodes. Hadoop is java based and any memory limits start causing problems. Also, in the compute partition you are charged for the whole node anyway.

Notes

- Your script should delete the outout directory if you want to rerun and copy out data   to that directory. Otherwise the Hadoop copy back fails because the file already exists.

  The current script forces to remove "WC-output".

- If you are running several Mapreduce jobs simultaneously, please make sure you choose different locations for for the configuration files. Basically change the line:

```
export HADOOP_CONF_DIR=/home/$USER/cometcluster
```

to point to different directories for each run. Otherwise the configuration from different jobs will overwrite in the same directory and cause problems.