MAPR Academy

Introduction to Developing Hadoop Applications

Learning Goals

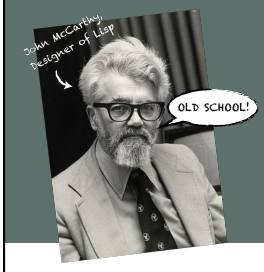
1. Illustrate the MapReduce model conceptually
2. Describe a brief history of MapReduce
3. Discuss how MapReduce works at a high-level
4. Define how data flows in MapReduce

- Cards Game Animation Placeholder

Learning Goals

1. Illustrate the MapReduce model conceptually
2. Describe a brief history of MapReduce
3. Discuss how MapReduce works at a high-level
4. Define how data flows in MapReduce

Lisp Map-Reduce



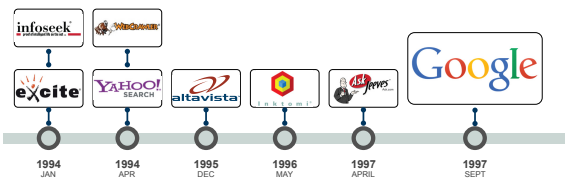
(map square '(1 2 3 4)) = (1 4 9 16)

- Applies same logic to each value, ***one value at a time***
- Emits result for each value

(reduce + '(1 4 9 16)) = 30

- Applies same logic to ***all the values*** taken together
- Emits single result for all values

Web Search Engines



Simplified Web Search Engine Approach

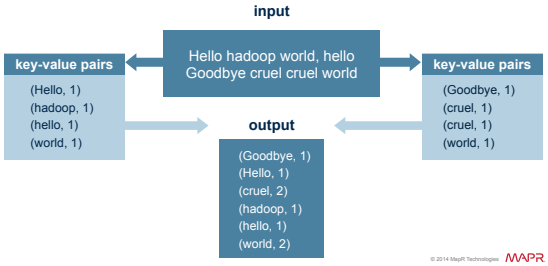
http://www.google.com/intl/en/insidesearch/howsearchworks/thestory/

Word Count Algorithm from Google White Paper

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

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

MapReduce: Simplified Data Processing on Large Clusters Jeffrey Dean (jeff@google.com) & Sanjay Ghemawat (sanjay@google.com) , Google,Inc.

Word Count Example

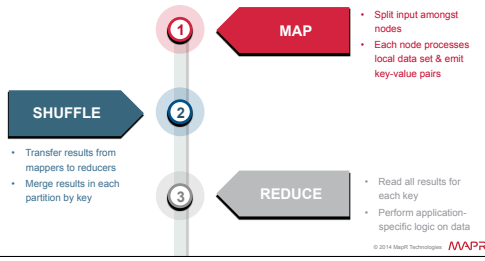


Learning Goals

1. Illustrate the MapReduce model conceptually
2. Describe a brief history of MapReduce
3. Discuss how MapReduce works at a high-level
4. Define how data flows in MapReduce

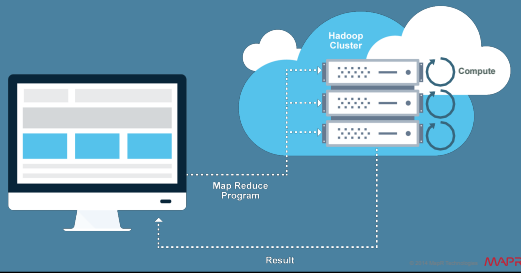
 10

Describe a Summary of Hadoop MapReduce

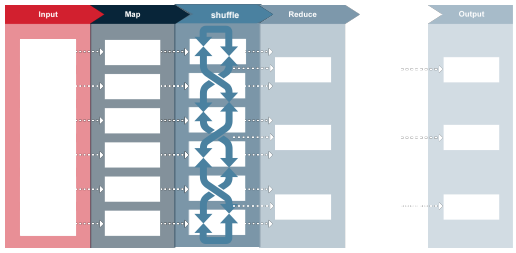


 11

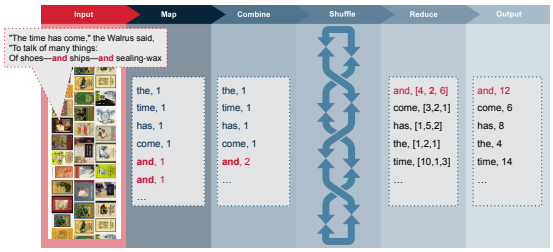
Describe the Hadoop Runtime Model



MapReduce Flow



MapReduce Example: Word Count



Knowledge Check

The following are true of a MapReduce job:

1. The data in the map phase is split amongst the task tracker nodes where the data is located
2. The shuffle phase is handled by the Hadoop framework.
3. Output from the mappers is sent to the reducers as partitions.
4. Reducers emit zero or more key-value pairs based on the application logic
5. All of the above

Learning Goals

1. Illustrate the MapReduce model conceptually
2. Describe a brief history of MapReduce
3. Discuss how MapReduce works at a high-level
4. Define how data flows in MapReduce

 16

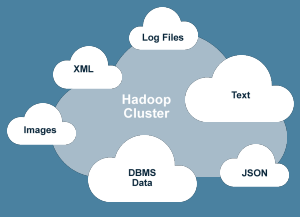
Define a Simplified MapReduce Workflow

1. **Load**
2. **Analyze**
3. **Store**
4. **Read**

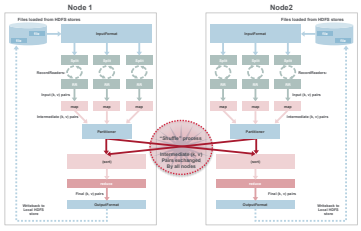
Describe How Data Gets Loaded Into the Cluster

MapR-FS

- POSIX + NFS Access
- Pre-Load or Permanent Store

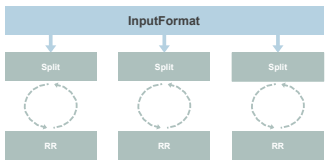


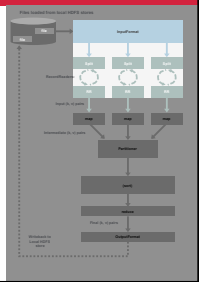
Summary of Execution and Data Flow



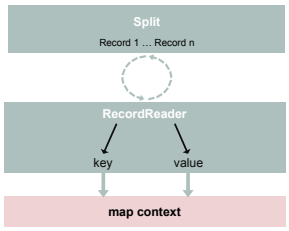
 19

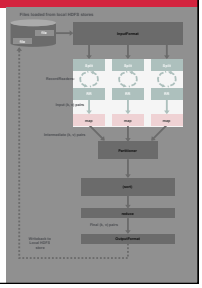
InputFormat Class



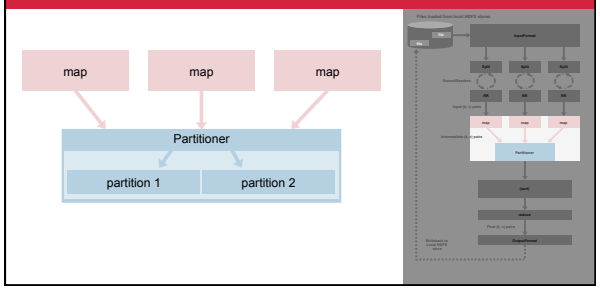


Mapper Class

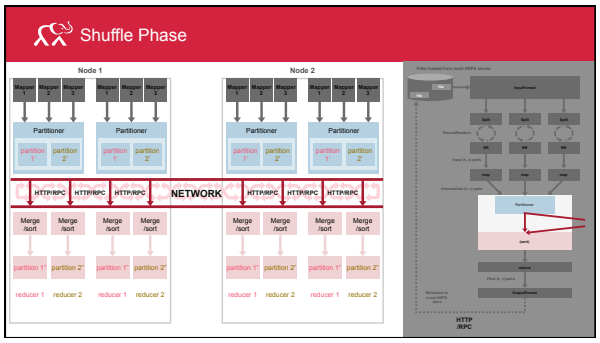




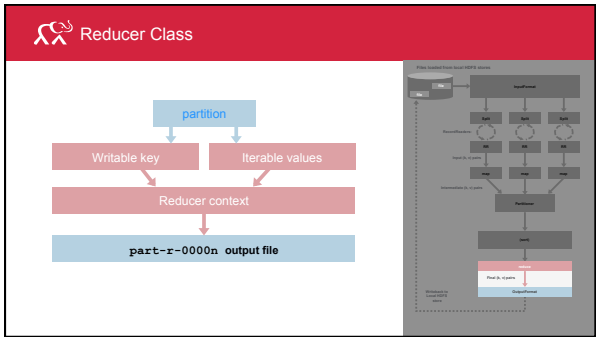
Partitioner Class



Shuffle Phase



Reducer Class



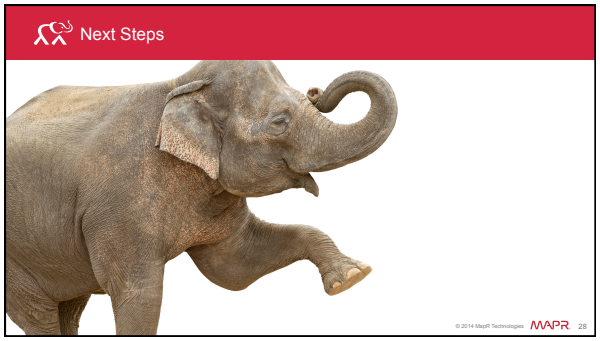
Results From MapReduce Job

- `_SUCCESS`
- `_logs/history*`
- `part-r-00000, part-r-00001, . . .`
- `part-m-00000, part-m-00001, . . .`

Refer to Lab Guide for Exercise Instructions

Summary

- MapReduce model
- History of MapReduce
- Data flow in MapReduce


Next Steps
© 2014 MapR Technologies
28

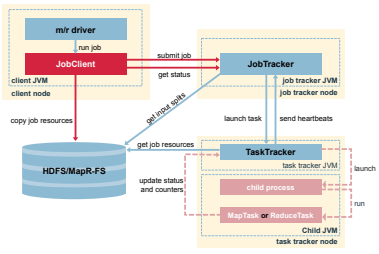
MAPR Academy

Job Execution Framework
MapReduce v1 & v2

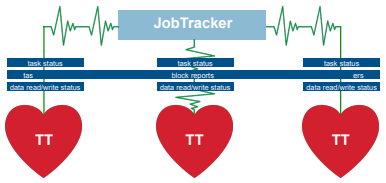
Learning Goals

1. Describe the MapReduce v1 job execution framework
2. Compare MapReduce v1 to MapReduce v2 (YARN)
3. Describe how jobs execute in YARN
4. Describe how to monitor jobs in YARN

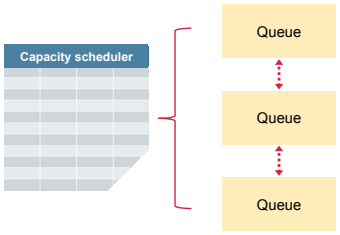
MapReduce Job Execution Framework



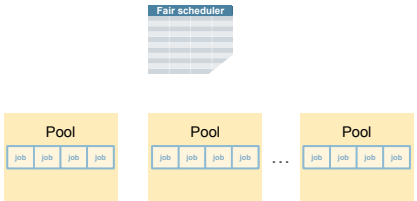
How the Heartbeat is Used



Hadoop Job Scheduling



Hadoop Fair Scheduler



Fair Scheduler Web UI

CentOS006 Fair Scheduler Administration

Pools

Pool	Running Jobs	Map Tasks				Reduce Tasks				Scheduling Mode
		Min Share	Max Share	Running	Fair Share	Min Share	Max Share	Running	Fair Share	
ExpressLane	0	0	-	0	0.0	0	-	0	0.0	FAIR
jcasaletto	1	0	-	1	1.0	0	-	0	0.0	FAIR
sridhar	0	0	-	0	0.0	0	-	0	0.0	FAIR
user01	0	0	-	0	0.0	0	-	0	0.0	FAIR
default	0	0	-	0	0.0	0	-	0	0.0	FAIR

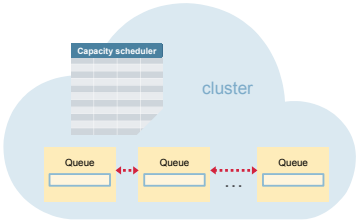
Running Jobs

Submitted	JobID	User	Name	Pool	Priority	Map Tasks			Reduce Tasks		
						Finished	Running	Fair Share	Finished	Running	Fair Share
Apr 24, 11:29	job_201403251527_0018	jcasaletto	universityjcasaletto	jcasaletto	NORMAL	0 / 1	1	1.0	0 / 1	0	0.0

Scheduler runs on JobTracker node

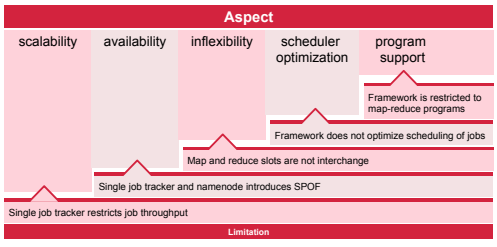
 MAPR 7

Hadoop Capacity Scheduler



 MAPR 8

Limitations in the Hadoop Execution Framework



 MAPR 9

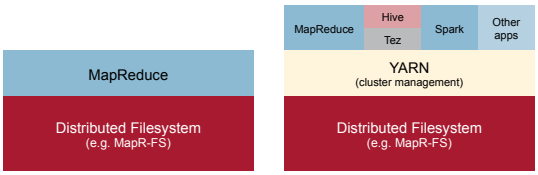
Learning Goals

1. Describe the MapReduce job execution framework
2. Compare MapReduce v1 to MapReduce v2 (YARN)
3. Describe how jobs execute in YARN
4. Describe how to monitor jobs in YARN

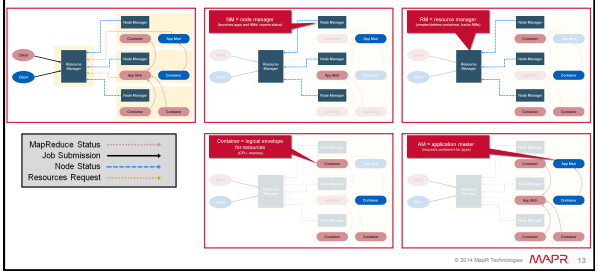
Motivation for YARN

1. **Map and reduce slot configuration is not dynamic**
   Inflexibility leads to underutilization
   No slot configuration in YARN
2. **Hadoop framework only supports MapReduce jobs**
   No support for non-MR apps
   YARN supports MR and non-MR apps
3. **Single job tracker has scalability limitations**
   ~4000 nodes per cluster max
   YARN equivalent of JT has multiple instances per cluster for scale
4. **YARN uses same API and CLI as MRv1 (with similar Web UIs)**

Differences MRv1 and MRv2



YARN Architecture



Knowledge Check

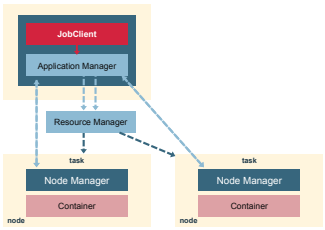
The following is true of YARN

1. There is no slot configuration in YARN
2. The YARN equivalent of Job Tracker supports multiple instances per cluster to scale.
3. YARN support MapReduce and non-MapReduce jobs
4. All of the above

Learning Goals

1. Describe the MapReduce job execution framework
2. Compare MapReduce v1 to MapReduce v2 (YARN)
3. Describe how jobs execute in YARN
4. Describe how to monitor jobs in YARN

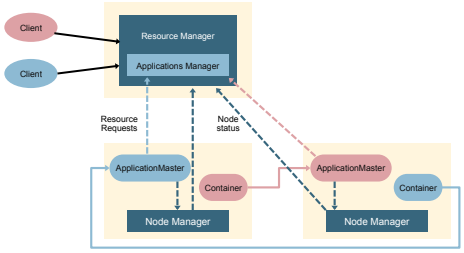
MapReduce Job Lifecycle in MRv2



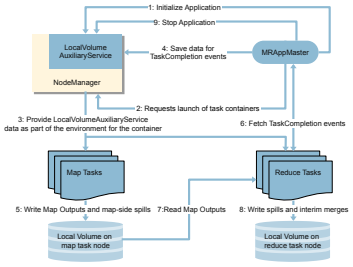
Non-MapReduce Job Lifecycle in MRv2

1. User submits app request by passing config for AM to RM
2. RM starts AM and allocates container
3. AM launches container and monitors it
4. When AM is done, it unregisters from RM

Describe How a Job is Launched in MRv2



MapR Direct Shuffle in YARN



Learning Goals

1. Describe the MapReduce job execution framework
2. Compare MapReduce v1 to MapReduce v2 (YARN)
3. Describe how jobs execute in YARN
4. Describe how to monitor jobs in YARN

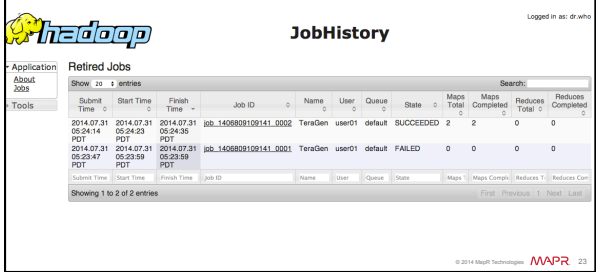
MRv1 and MRv2 on the Same MapR Cluster

`hadoop1, hadoop -classic`	Command Line	`hadoop2, mapred, yarn`
`MAPR_MAPREDUCE_MODE=classic`	Environment Variable	`MAPR_MAPREDUCE_MODE=yarn`
`default_mode=classic`	Client	`default_mode=yarn`
`maprcli cluster \ mapreduce set -mode classic`	MapR Cluster-Wide Default	`maprcli cluster \ mapreduce set -mode yarn`

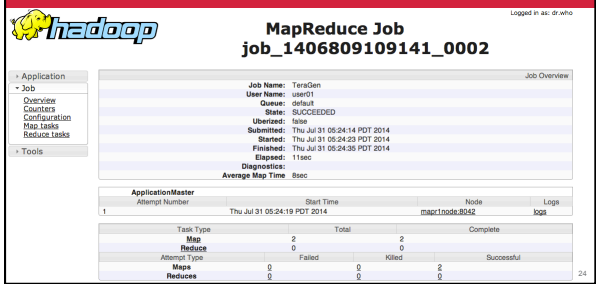
Differences in MRv2 Job Management

MRv1	MRv2 (YARN)
JT or TT Web Uis	UIs for RM/NM/History Server
MapR metrics database	MapR metrics database (future release of MapR)
`hadoop job`	`mapred` CLI

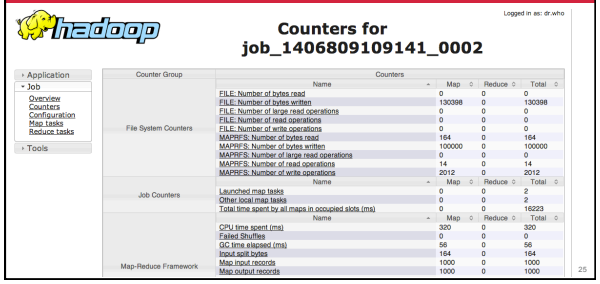
Job History Server



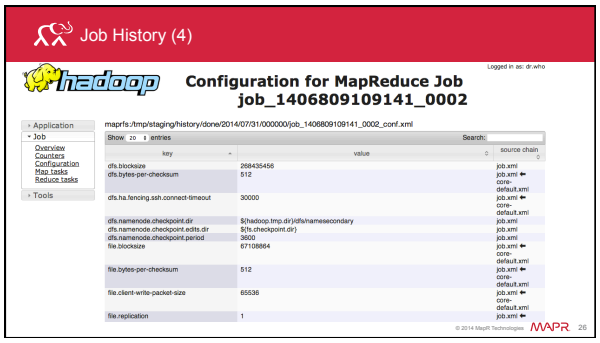
Job History (2)



Job History (3)



Job History (4)



Refer to your lab guide for exercise instructions

Summary

- Execution of MapReduce
- Motivation of YARN development
- Differences between the 2 versions
- YARN job execution and management

 28

Next Steps

 29

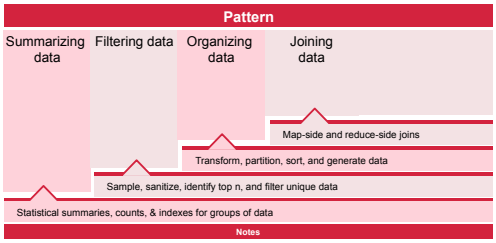
MAPR Academy

Write a MapReduce Program

Learning Goals

1. Summarize programming problem
2. Design and implement
   - Mapper class
   - Reducer class
   - Driver class
3. Build and execute code

MapReduce Design Patterns



Some MapReduce Programming Tips

- Start with a template for the driver, mapper, and reducer classes
- Modify the template to suit the needs of your application
- **Understand the flow and transformation of data:**

- Identify appropriate types for keys and values

Example Data Set

2000	2025191	1788950	236241	1544607	1458185	86422	480584	330765	149819
2001	1991082	1862846	128236	1483563	1516008	-32445	507519	346838	160681
2002	1853136	2010894	-157758	1337815	1655232	-317417	515321	355662	159659
2003	1782314	2159899	-377585	1258472	1796890	-538418	523842	363009	160833
2004	1880114	2292841	-412727	1345369	1913330	-567961	534745	379511	155234
2005	2153611	2471957	-318346	1576135	2069746	-493611	577476	402211	175265
2006	2406869	2655050	-248181	1798487	2232981	-434494	608382	422069	186313
2007	2567985	2728686	-160701	1932896	2275049	-342153	635089	453637	181452
2008	2523991	2982544	-458553	1865945	2507793	-641848	658046	474751	183295
2009	2104989	3517677	-1412688	1450980	3000661	-1549681	654009	517016	136993
2010	2162706	3457079	-1294373	1531019	2902397	-1371378	631687	554682	77005
2011	2303466	3603059	-1299593	1737678	3104453	-1366775	565788	498606	67182
2012	2450164	3537127	-1086963	1880663	3029539	-1148876	569501	507588	61913

year delta

Learning Goals

1. Summarize programming problem
2. Design and implement
   - Mapper class
   - Reducer class
   - Driver class
3. Build and execute code

Input to the Mapper Class

Input format = `TextInputFormat`
Key = `LongWritable`
Value = `Text`

More detail later

First record:

input key: `0`

input value: `1901 588 525 63 588 525 63 ..... ..... .....`

```
StringTokenizer itr = new StringTokenizer(value.toString(),"\\s+");
```


Output of the Mapper Class

output key / output value

```
context.write(new Text("summary"),new Text(year + "_" + delta));
```

First few lines of output

```
summary   1901_63
summary   1902_77
summary   1903_45
summary   1904_-43
summary   1905_-23
summary   1906_25
```


Design and Implement the Mapper Class

```
public class ReceiptsMapper  extends Mapper
<LongWritable,Text,Text,Text> {
   public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
      StringTokenizer iterator = new
StringTokenizer(value.toString(),"\\s+");
      String year = iterator.nextToken();
      iterator.nextToken();
      iterator.nextToken();
      String delta= iterator.nextToken();
      context.write(new Text("summary"), new Text(year + "_" +
delta);
   }
}
```

Knowledge Check

Which statements are true of Mapper class?

1. The four arguments to the Mapper class represent input key type, input value type, output key type, and output value type.
2. The mapper class calls the map() method
3. The first two arguments to the map() method are the key and value which must match the types defined in the Mapper class definition
4. All of the above

 10

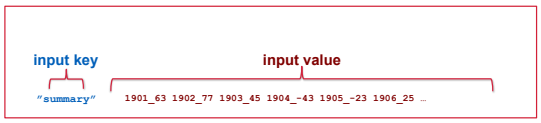
Learning Goals

1. Summarize programming problem
2. Design and implement
   - Mapper class
   - Reducer class
   - Driver class
3. Build and execute code

 11

Input to the Reducer Class

- Mapper output key = `Text` → Reducer input key = `Text`
- Mapper output value = `Text` → Reducer input values = `Text`



 12

Output of the Reducer Class

Output key = `Text`
- min(year):

Output value = `FloatWritable`
min (from `long` to `FloatWritable`)

```
min(2009):      -1412688.0
```


Design and Implement the Reducer Class (1)

```
public class ReceiptsReducer  extends Reducer
<Text,Text,Text,FloatWritable> {

   public void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException {
     long tempValue = 0L, min=Long.MAX_VALUE;
     Text tempYear=null, tempValue=null, minYear=null, maxYear=null;
     String compositeString;
     String[] compositeStringArray;
```


Design and Implement Reducer Class (2)

```
for (Text value: values) {
     compositeString = value.toString();
     compositeStringArray = compositeString.split("_");
     tempYear = new Text(compositeStringArray[0]);
     tempValue = new Long(compositeStringArray[1]).longValue();
     if(tempValue < min) {
        min=tempValue;
        minYear=tempYear;
     }
  }
  Text keyText = new Text("min" + "(" + minYear.toString() + "): ");
  context.write(keyText, new FloatWritable(min));
 }
}
```


Knowledge Check

Which statements are true of the Reducer class?.

1. The four arguments to the Reducer class represent the type for input key, input value, output key, and output value
2. If the output value of the Mapper is Text, the input value to the Reducer can be LongWritable
3. If the output key of the Mapper is Text, the input key to the Reducer class must also be Text.
4. All of the above
5. 1 & 3 only

 16

Learning Goals

1. Summarize programming problem
2. Design and implement
   - Mapper class
   - Reducer class
   - Driver class
3. Build and execute code

 17

Implement the Driver (1)

```
public class ReceiptsDriver extends Configured implements Tool {
   public int run(String[] args) throws Exception {
      if (args.length != 2) {
         System.err.printf("usage: %s [generic options] <inputfile>
<outputdir>\n", getClass().getSimpleName());
         System.exit(1);
      }
      Job job = new Job(getConf(), "my receipts");
      job.setJarByClass(ReceiptsDriver.class);
      job.setMapperClass(ReceiptsMapper.class);
      job.setReducerClass(ReceiptsReducer.class);
```

 18

Implement the Driver (2)

```
  job.setInputFormatClass(TextInputFormat.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(FloatWritable.class);
  job.setMapOutputValueClass(Text.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  return job.waitForCompletion(true) ? 0 : 1;
  }
  public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();
     System.exit(ToolRunner.run(conf, new ReceiptsDriver(), args));
  }
}
```


Knowledge Check

Which statements are true of the Driver class?

1. The Driver class first checks the invocation of the command (checks the count of the command-line arguments provided)
2. It sets values for the job, including the driver, mapper, and reducer classes used.
3. In the Driver class, we can specify how we want to launch the job – synchronously or asychronously
4. All of the above

Learning Goals

1. Summarize programming problem
2. Design and implement
   - Mapper class
   - Reducer class
   - Driver class
3. Build and execute code

Configure the Environment

```
$ cat ~/.profile
export HADOOP_HOME=/opt/mapr/hadoop/hadoop-0.20.2
export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native/Linux-amd64-64
export PATH=$HADOOP_HOME/bin:$PATH
export CLASSPATH=$HADOOP_HOME/*:$HADOOP_HOME/lib/*
export HADOOP_CLASSPATH=$CLASSPATH
```


Build the Jar

```
$ mkdir classes
$ javac -d classes ReceiptsMapper.java
$ javac -d classes ReceiptsReducer.java
$ jar -cvf Receipts.jar -C classes/ .
$ javac -classpath $CLASSPATH:Receipts.jar -d classes ReceiptsDriver.java
$ jar -uvf Receipts.jar -C classes/ .
```


Launch the Hadoop Job

```
$ hadoop jar Receipts.jar Receipts.ReceiptsDriver \
/user/user01/RECEIPTS/DATA/receipts.txt \
/user/user01/RECEIPTS/OUT
```

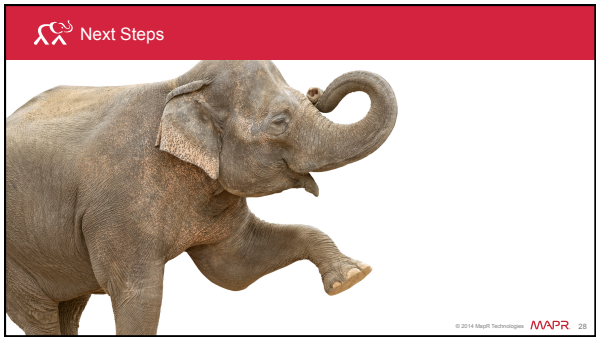

Examine the Output

```
$ hadoop fs -cat /user/user01/RECEIPTS/OUT/part-r-00000
min(2009):   -1412688.0
```


Refer to Lab Guide for Exercise Instructions

Summary

- Design and implement a Mapper, Reducer and Driver classes
- Build and execute code


Next Steps
© 2014 MapR Technologies
28

MAPR Academy

Use the MapReduce API

Learning Goals

1. API overview
2. Examine
   - Mapper input processing data flow
   - Reducer output processing data flow
3. Explore the Mapper, Reducer and Job class API

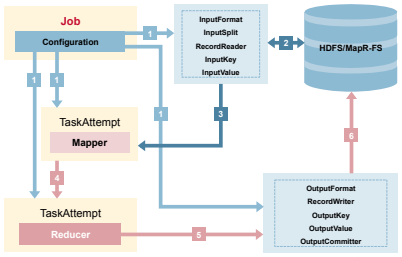
Hadoop Version Support on MapR

Apache hadoop

mapred and mapreduce

Aspect	mapred	mapreduce
Supported on MapR	yes	yes
Deprecated	no	no
YARN-compatible	yes	yes
Types	interfaces	abstract classes
Objects	OutputCollector Reporter JobConf	Context
Methods	map() reduce()	map() reduce() cleanup() setup() run()
Output files	part-nnnnn	part-r-nnnnn part-m-nnnnn
Reducer input values	java.lang.iterable	java.lang.iterator

Interactions Between Objects



WritableComparable Types

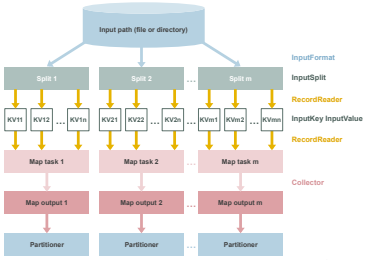
```
public interface WritableComparable extends Writable, Comparable
{
    void readFields(DataInput in);
    void write(DataOutput out);
    int compareTo(WritableComparable o)
}
```


Learning Goals

1. API overview
2. Examine
   - Mapper input processing data flow
   - Reducer output processing data flow
3. Explore the Mapper, Reducer and Job class API

 7

Mapper Input Flow



 8

`InputFormat` Class

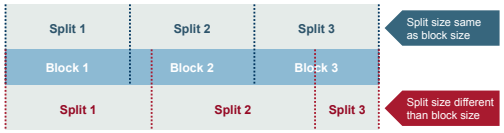
```
public abstract class InputFormat<K, V> {

 // returns array of InputSplits for job
 public abstract List<InputSplit> getSplits(JobContext) throws
IOException, InterruptedException;

 // create new record reader
 public abstract RecordReader<K, V> createRecordReader(InputSplit
split, TaskAttemptContext context) throws IOException,
InterruptedException;

}
```

Implementations include:

- `TextInputFormat` (for single-line records in text files)
- `SequenceFileInputFormat` (for binary files)

 9

Input Split Boundaries



InputSplit Class

```
public abstract class InputSplit {

   // returns number of bytes in the split
   public abstract long getLength() throws IOException;

   // returns array of nodes
   public abstract String[] getLocations() throws IOException;

}
```

Implementations include:

- `FileSplit` (breaks each file into splits)
- `CombineFileSplit` (breaks multiple files into single split)

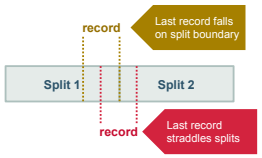
RecordReader Interface

```
public interface RecordReader<K, V> {
    //Reads the next key/value pair from the input for processing.
    boolean next(K key, V value) throws IOException;
    //Creates an object of the appropriate type to be used as a key and value.
    K createKey();
    V createValue();
    //returns the current position in the input
    long getPos() throws IOException;
    //Close this InputSplit
    public void close() throws IOException;
    float getProgress() throws IOException;
}
```

Implementations include:

- `LineRecordReader`
- `SequenceFileRecordReader`

Identifying Record Boundaries



 13

Knowledge Check

In the Mapper Input flow (pick all that apply)

1. Input files are split and key-value pairs generated for each split and sent to the same Mapper.
2. The map() method is called for each key-value pair once and output sent to the partitioner.
3. Results from the mapper are stored in the file system of the Control node
4. 1, 2 & 3
5. 1 & 2 only

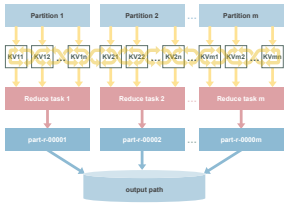
 14

Learning Goals

1. API overview
2. Examine
   - Mapper input processing data flow
   - Reducer output processing data flow
3. Explore the Mapper, Reducer and Job class API

 15

Reducer Output Flow



OutputFormat Class

```
public abstract class OutputFormat<K, V> {
   public abstract RecordWriter<K, V> getRecordWriter(TaskAttemptContext
context) throws IOException, InterruptedException;

   public abstract void checkOutputSpecs(JobContext context) throws
IOException, InterruptedException;

   public abstract OutputCommitter getOutputCommitter(TaskAttemptContext
context) throws IOException, InterruptedException;
}
```


RecordWriter Class

```
public abstract class RecordWriter<K, V> {

   public abstract void write(K key, V value) throws IOException,
InterruptedException;

   public abstract void close(TaskAttemptContext context) throws
IOException, InterruptedException;

}
```


Knowledge Check

In the Reduce output flow (pick all that apply)

1. Each reducer takes its partition as input, processes one iterable list of key-value pairs at a time.
2. Produces an output file called 'part-r-0000x
3. The output directory for this file is specified in the Job configuration and must reside on the HDFS/Mapr-FS file system
4. 1, 2 & 3
5. 1 & 3 only

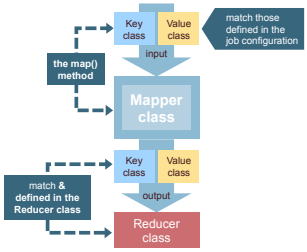
 19

Learning Goals

1. API overview
2. Examine
   - Mapper input processing data flow
   - Reducer output processing data flow
3. Explore the Mapper, Reducer and Job class API

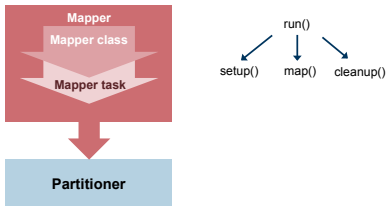
 20

Input and Output Keys and Values



 21

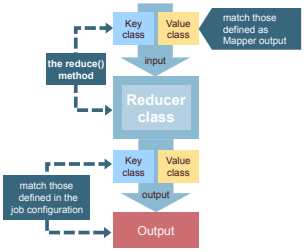
Mapper Class



Mapper Example

```
public  class MyWordcountMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {
   private Text word = new Text();
   private final static IntWritable one = new IntWritable(1);
   public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
      String line = value.toString();
      StringTokenizer iterator = new StringTokenizer(line);
      while (iterator.hasMoreTokens()) {
         word.set(iterator.nextToken());
         context.write(word, one);
      }
   }
}
```

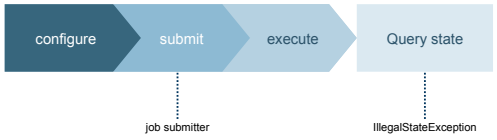

Reducer Class



Reducer Example

```
public class MyWordcountReducer extends
Reducer<Text,IntWritable,Text,IntWritable> {
   public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable value : values) {
         sum += value.get();
      }
      context.write(key, new IntWritable(sum));
   }
}
```


Job Class



Job Object

```
Configuration conf = new Configuration();
Job job = new Job(conf, "mywordcount");
```


Job Methods

void	**setGroupingComparatorClass**(Class<? extends RawComparator> cls) Define the comparator that controls which keys are grouped together for a single call to Reducer.reduce(Object, Iterable, org.apache.hadoop.mapreduce.Reducer.Context)
void	**setInputFormatClass**(Class<? extends InputFormat> cls) Set the InputFormat for the job.
void	**setJarByClass**(Class<?> cls) Set the Jar by finding where a given class came from.
void	**setJobName**(String name) Set the user-specified job name.
void	**setMapOutputKeyClass**(Class<?> theClass) Set the key class for the map output data.
void	**setMapOutputValueClass**(Class<?> theClass) Set the value class for the map output data.
void	**setMapperClass**(Class<? extends Mapper> cls) Set the Mapper for the job.
void	**setNumReduceTasks**(int tasks) Set the number of reduce tasks for the job.
void	**setOutputFormatClass**(Class<? extends OutputFormat> cls) Set the OutputFormat for the job.
void	**setOutputKeyClass**(Class<?> theClass) Set the key class for the job output data.
void	**setOutputValueClass**(Class<?> theClass) Set the value class for job outputs.

Job Methods

void	**setPartitionerClass**(Class<? extends Partitioner> cls) Set the Partitioner for the job.
void	**setReducerClass**(Class<? extends Reducer> cls) Set the Reducer for the job.
void	**setSortComparatorClass**(Class<? extends RawComparator> cls) Define the comparator that controls how the keys are sorted before they are passed to the Reducer.
void	**setWorkingDirectory**(Path dir) Set the current working directory for the default file system.
void	**submit**() Submit the job to the cluster and return immediately.
boolean	**waitForCompletion**(boolean verbose) Submit the job to the cluster and wait for it to finish.

Simplistic Driver Implementation

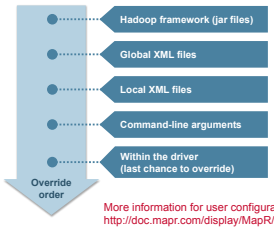
```
public class MyWordcountDriver {
   public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      Job job = new Job(conf, "mywordcount");
      job.setJarByClass(MyWordcountDriver.class);
      job.setMapperClass(MyWordcountMapper.class);
      job.setReducerClass(MyWordcountReducer.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);
      job.setInputFormatClass(TextInputFormat.class);
      job.setOutputFormatClass(TextOutputFormat.class);
      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputPath(job, new Path(args[1]));
      System.exit(job.waitForCompletion(true) ? 0 : 1);
   }
}
```

Best Way to Implement the Driver

```
public class MyWordcountDriver extends Configured implements Tool {
  public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      System.exit(ToolRunner.run(conf, new MyWordcountDriver (), args));
   }

  public int run(String[] args) throws Exception {
      Job job = new Job(conf, "mywordcount");
      . . .
      job.waitForCompletion(true) ? 0 : 1;
}
```

 MAPR 31

How to Set Configuration Parameters



More information for user configurable parameters at
http://doc.mapr.com/display/MapR/MapR+Parameters

 MAPR 32

Refer to your lab guide for exercise instructipns

Summary

- MapReduce API
- Writable types
- Mapper - Split
- Map method – single record
- Reducer - one or more partitions
- Reduce method – single key (and all associated values)
- ToolRunner for driver code

Next Steps

MAPR Academy

Managing, Monitoring, and Testing MapReduce Jobs

Learning Goals

1. Work with counters
2. Use the MCS to monitor jobs
3. Manage and display jobs, history, and logs
4. Write unit tests for MapReduce programs

Summary of Counters

Counter	Description
File system	Total number of bytes read and written during a Hadoop job
Job	Summary of task cardinality and CPU time
Framework	Granular summaries of CPU and memory consumption, records read & written, & bytes read & written in each phase of MapReduce
Custom	Completely specific to the application

File System Counters

Counter	Description
FILE_BYTES_WRITTEN	Total number of bytes written to local file system
MAPRFS_BYTES_READ	Total number of bytes read from MapR-FS
MAPRFS_BYTES_WRITTEN	Total number of bytes written to MapR-FS

Job Counters

Counter	Description
DATA_LOCAL_MAPS	Total number of map tasks executed on local data
FALLOW_SLOTS_MILLIS_MAPS	Total time map tasks spend waiting after slots are reserved (pre-emption)
FALLOW_SLOTS_MILLIS_REDUCES	Total time reduce tasks spend waiting after slots are reserved (pre-emption)
SLOTS_MILLIS_MAPS	Total time map tasks spend executing
SLOTS_MILLIS_REDUCES	Total time reduce tasks spend executing
TOTAL_LAUNCHED_MAPS	Total number of map tasks launched, including failed tasks
TOTAL_LAUNCHED_REDUCES	Total number of reduce tasks launched, including failed tasks

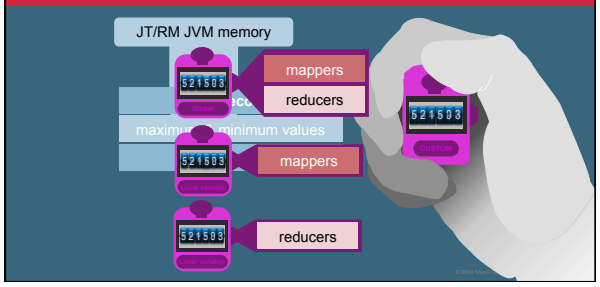
Framework Counters (1)

Counter	Description
COMBINE_INPUT_RECORDS	Incremented for every record read during combine phase, if used.
COMBINE_OUTPUT_RECORDS	Incremented for every record written during combine phase, if used.
CPU_MILLISECONDS	Total time spent by all tasks on CPU
GC_TIME_MILLIS	Total time spent doing garbage collection
MAP_INPUT_RECORDS	Incremented for every record successfully read in map phase
MAP_OUTPUT_RECORDS	Incremented for every record successfully written in map phase
PHYSICAL_MEMORY_BYTES	Total physical memory consumed by all tasks

Framework Counters (2)

Counter	Description
REDUCE_INPUT_GROUPS	Total number of unique keys
REDUCE_INPUT_RECORDS	Total number of records read by all reduce tasks
REDUCE_OUTPUT_RECORDS	Total number of records written by all reduce tasks
REDUCE_SHUFFLE_BYTES	Total number of bytes of output from map tasks copied during shuffle to reducers
SPILLED_RECORDS	Total number of records spilled to disk by all map and reduce tasks
SPLIT_RAW_BYTES	Total number of bytes consumed for metadata (offset + length) during splits
VIRTUAL_MEMORY_BYTES	Total number of virtual memory bytes consumed by tasks (RAM + swap)

Custom Counters



Knowledge Check

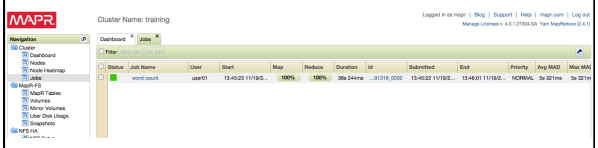
You can use the built-in counters to validate that:

1. The correct number of bytes were read and written
2. The correct number of tasks were launched and successfully ran
3. The amount of CPU and memory consumed is appropriate for your job and cluster nodes
4. The correct number of records were read and written
5. All of the above

Learning Goals

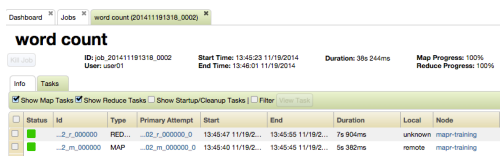
1. Work with counters
2. Use the MCS to monitor jobs
3. Manage and display jobs, history, and logs
4. Write unit tests for MapReduce programs

Displaying Jobs in MCS

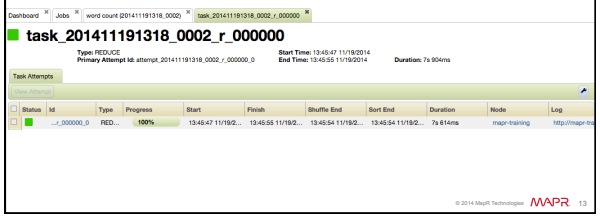


https://webserver:8443

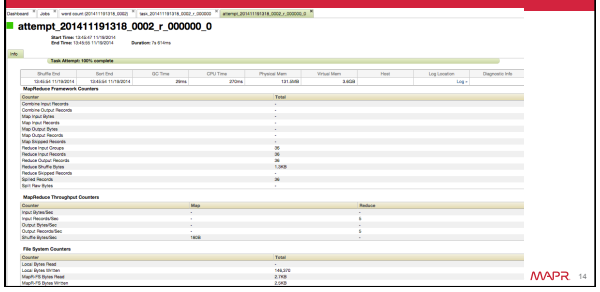
Getting Job Summaries in MCS



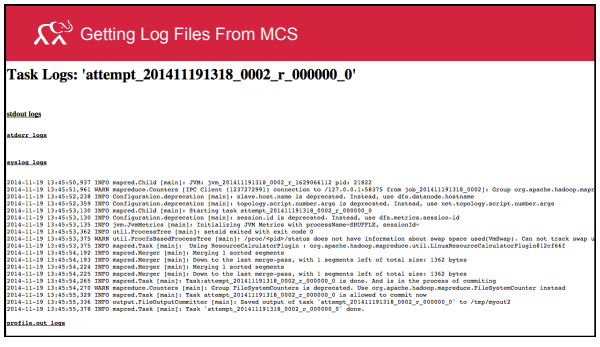
Getting Task Summaries in MCS



Getting Task Details in MCS



Getting Log Files From MCS



Learning Goals

1. Work with counters
2. Use the MCS to monitor jobs
3. Manage and display jobs, history, and logs
4. Write unit tests for MapReduce programs

 16

Tracking Status of Launched Jobs (1)

```
$ hadoop job -list
1 jobs currently running
JobId     State     StartTime UserName  Priority  SchedulingInfo
job_201404051135_0322    1    1396917391960  user20    NORMAL    NA

$ hadoop job -status job_201404051135_0322
Job: job_201404051135_0322
file: maprfs:/var/mapr/cluster/mapred/jobTracker/staging/user20/.staging/
job_201404051135_0322/job.xml
tracking URL: http://ip-10-170-165-141:50030/jobdetails.jsp?jobid=job_201404051135_0322
map() completion: 0.024717983
reduce() completion: 0.0
Counters: 19
<output omitted>
```

 17

Tracking Status of a Launched Job (2)

 18

Stopping a Launched Job

```
$ hadoop job -list
1 jobs currently running
JobId    State    StartTime    UserName Priority SchedulingInfo
job_201404051135_03221    1396917391960    user20   NORMAL   NA

$ hadoop job -kill job_201404051135_03221
```


Modifying a Job Priority

When job is launched using API

```
. . .
   Configuration conf = new Configuration();
   conf.set("mapred.job.priority", "VERY_HIGH");
   Job job = new Job(conf, "receipts" + System.getProperty("user.name"));
. . .
```

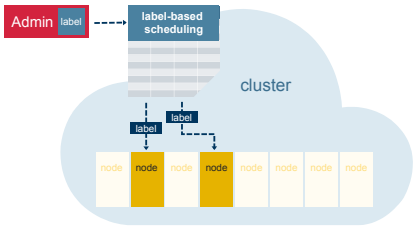
When job is launched using a configuration file (or passing –D mapred.job.priority=X at job submission

```
<configuration>
<property>
<name>mapred.job.priority</name><value>HIGH</value>
</property>
</configuration>
```

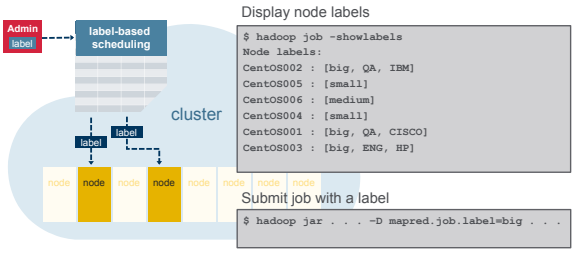
After job is submitted (but before it is run) using hadoop CLI

```
$ hadoop job -set-priority job_201311221648_0039 VERY_LOW
13/11/26 14:59:12 INFO fs.JobTrackerWatcher: Current running JobTracker is:
ip-10-198-41-188/10.198.41.188:9001
Changed job priority.
```


MapR Feature: Label-Based Scheduling



Using Label-Based Scheduling



Display node labels

```
$ hadoop job -showlabels
Node labels:
CentOS002 : [big, QA, IBM]
CentOS005 : [small]
CentOS006 : [medium]
CentOS004 : [small]
CentOS001 : [big, QA, CISCO]
CentOS003 : [big, ENG, HP]
```

Submit job with a label

```
$ hadoop jar . . . -D mapred.job.label=big . . .
```


Apache Commons Logging (JCL)

Apache Commons™
http://commons.apache.org/

LOG4J™

The Apache Avalon Project
http://avalon.apache.org/

Logging Information

```
. . .
private static Log log = LogFactory.getLog(MyClass.class);
. . .
public void map( . . .) {
   if(debugcondition) {
      log.debug("debug log message");
      // OR
      System.out.println("debug output");
   else if (errorcondition) {
      log.error("error log message");
      // OR
      System.err.println("error output");
   }
. . .
}
```


Viewing the History of a Job

```
$ hadoop job -history TERASORT.OUT/
Hadoop job: job_201311211623_0015
. . .
Status: SUCCESS
. . .
|Job Counters                  |Aggregate execution time of mappers(ms)|0
|0         |28,978
. . .
Task Summary
Setup    1    1        0    0    23-Nov-2013 15:08:25  23-Nov-2013 15:08:26
(1sec)
. . .
Analysis
Time taken by best performing map task task_201311211623_0015_m_000001: 12sec
. . .
```


Learning Goals

1. Work with counters
2. Use the MCS to monitor jobs
3. Manage and display jobs, history, and logs
4. Write unit tests for MapReduce programs

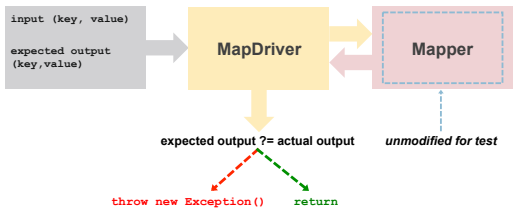
Summary of MRUnit

- Testing harness for MapReduce based on `JUnit`
- Developed at Cloudera (http://mrunit.apache.org)

Apache MRUnit™

- Uses `LocalJobRunner` to execute code

Conceptual View of `MRUnit` Map Test



Example MRUnit Code: Map-only (1)

```
public class ReceiptsTest {
   private static MapDriver<LongWritable, Text, Text, Text> mapDriver;

   @Before
   private static void setUp() {
      Receipts.ReceiptsMapper mapper = new Receipts.ReceiptsMapper();
      mapDriver = MapDriver.newMapDriver(mapper);
   }
```

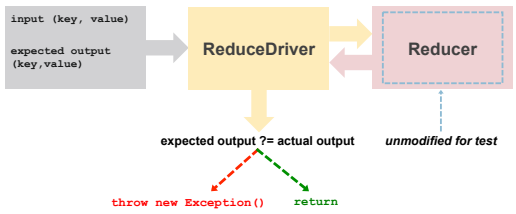

Example MRUnit Code:Map-only (2)

```
   @Test
   private static void testMapper() throws IOException,
InterruptedException {
      final LongWritable key = new LongWritable(0);
      Text value = new Text("1901 588 525 63 588 525
63 .......... .......... ..........");
      mapDriver
         .withInput(key, value)
         .withOutput(new Text("summary"), new Text ("1901_63"))
          .runTest();
      }
```


Example MRUnit Code:map-only (3)

```
   public static void main(String[] args) {
      setUp();
      try { testMapper();}
      catch (Exception e) {System.err.println("exception: " +
e.toString());}
      return;
   }
}
```


Conceptual View of **MRUnit** Reducer Test



Refer to Lab Guide for Exercise Instructions

Summary

- Counters
- MCS
- Launch, manage, and monitor
- Test

 34

Next Steps

 35

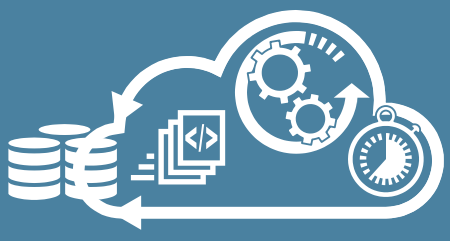
MAPR Academy

Managing Performance

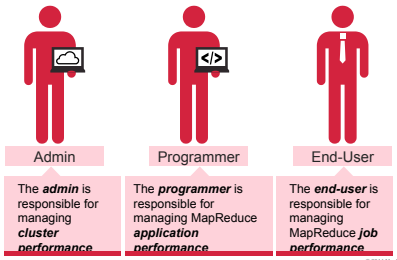
Learning Goals

1. Review components of MapReduce performance
2. Enhance performance in MapReduce jobs
3. Describe MapR performance enhancements

Performance Tuning Tips

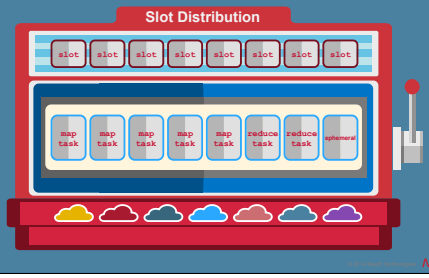


Breakdown of Runtime Performance



 4

Map and Reduce Slots



 5

Knowledge Check

Slot configuration is calculated by:

1. CPU
2. Memory
3. Disk
4. All of the above

 6

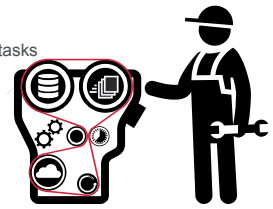
Learning Goals

1. Review components of MapReduce performance
2. Enhance performance in MapReduce jobs
3. Describe MapR performance enhancements

 7

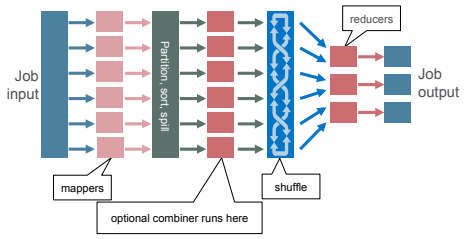
Performance Enhancements

- Using a custom combiner
- Compressing map results
- Modifying number of reduce tasks
- Using speculative execution
- Reusing a JVM
- Configuring sort properties
- Configuring Java properties



 8

Custom Combiners



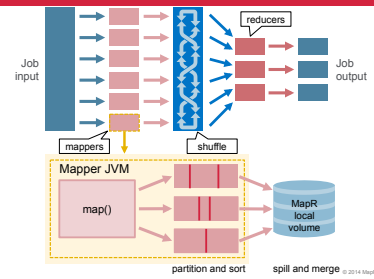
 9

Example: Using a Combiner

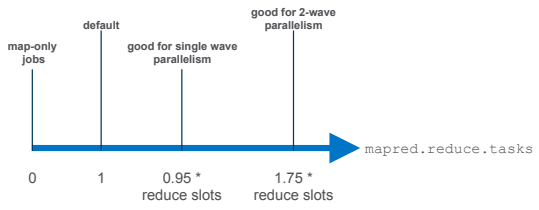
```
public class UniversityDriver extends Configured implements Tool {

   public int run(String[] args) throws Exception {
     job.setCombinerClass(MyCombiner.class);
. . .
```

```
public class MyReducer  extends Reducer
<Text,IntWritable,Text,FloatWritable> {

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

   }
}
```

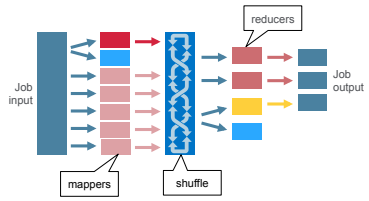

Map Output Compression



Tuning Number of Reduce Tasks

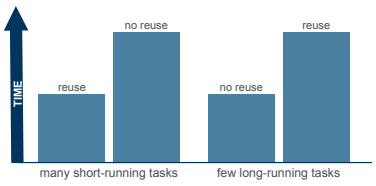


Speculative Execution



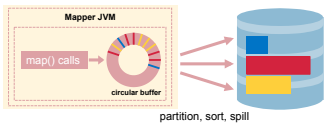
 13

Reusing JVMs



 14

Map-side Sort Properties



 15

Limiting Memory Consumption

property	description
`mapred.map.child.java.opts`	JVM properties that apply to map tasks
`mapred.reduce.child.java.opts`	JVM properties that apply to reduce tasks
`mapred.child.java.opts`	JVM properties that apply to both map and reduce tasks
`mapred.child.ulimit`	Maximum size of all virtual memory consumed by a task and its children

 16

Knowledge Check

Some performance enhancements that can be used for MapReduce jobs are:

1. Using a custom Combiner (best if the number of keys is significantly less than the number of records)
2. Compressing Mapper results
3. Modifying number of reduce tasks
4. All of the above

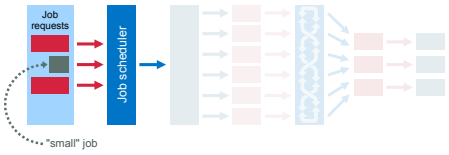
 17

Learning Goals

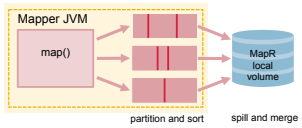
1. Review components of MapReduce performance
2. Enhance performance in MapReduce jobs
3. Describe MapR performance enhancements

 18

MapR ExpressLane



Direct Shuffle



Label-Based Scheduling (redux)

- **Display node labels**

```
$ hadoop job -showlabels
Node labels:
CentOS002 : [big, QA, IBM]
CentOS005 : [small]
CentOS006 : [medium]
CentOS004 : [small]
CentOS001 : [big, QA, CISCO]
CentOS003 : [big, ENG, HP]
```

- **Submit job with a label**

Q: what about data locality for map tasks?

```
$ hadoop jar my.jar -D mapred.job.label=big myin myout
```


Node Labels and Topology

```
$ hadoop job -showlabels
Node labels:
CentOS001 : [small]
CentOS002 : [small]
CentOS003 : [medium]
CentOS004 : [medium]
CentOS005 : [large]
CentOS006 : [large]
```

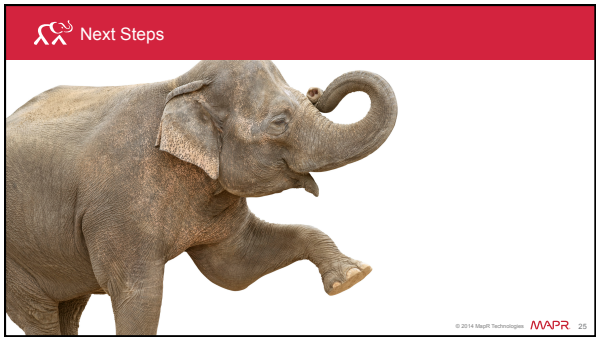
Hostname	Physical IP(s)	File...	Physical Topology
CentOS001	10.10.82.51 10.10.82.52	0 ago	/row1/rack1/CentOS001
CentOS002	10.10.82.53 10.10.82.54	0 ago	/row1/rack1/CentOS002
CentOS003	10.10.82.55 10.10.82.56	0 ago	/row1/rack1/CentOS003
CentOS004	10.10.82.57 10.10.82.58	0 ago	/row1/rack1/CentOS004
CentOS005	10.10.82.59 10.10.82.60	0 ago	/row1/rack1/CentOS005
CentOS006	10.10.82.61 10.10.82.62	0 ago	/row1/rack1/CentOS006

same label ↔ same topology

Refer to Lab Guide for Exercise Instructions

Summary

- Components of MapReduce performance
- Enhancing performance of MapReduce jobs


Next Steps
© 2014 MapR Technologies
25

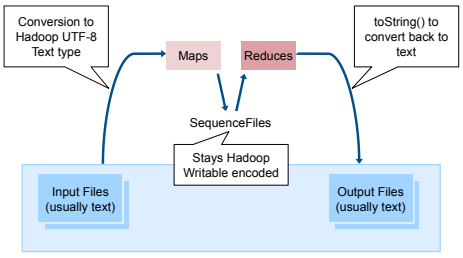
MAPR Academy

Working with Data

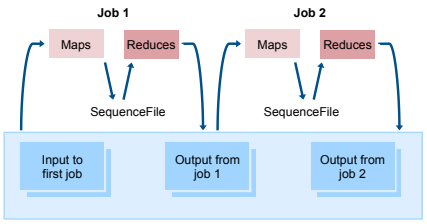
Learning Goals

1. Working with sequence files
2. Working with the distributed cache
3. Working with HBase

Generic MapReduce Data Flow

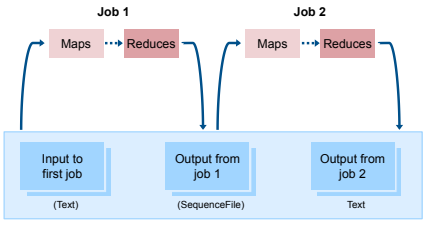


Typical MapReduce Workflows



 4

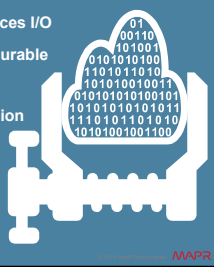
Sequence Files with MapReduce



 5

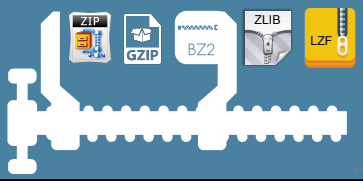
File Compression

- **Use compression because it reduces I/O**
- **MapR file system supports configurable compression**
- **Sequence files support compression**

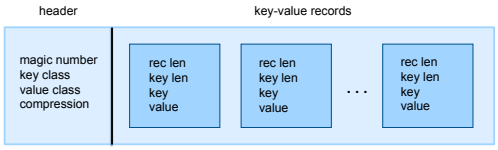


Compressing Sequence Files

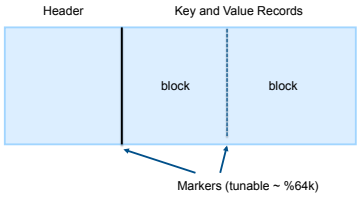
- NONE ➔ do not compress
- RECORD ➔ compress values only (not keys)
- BLOCK ➔ compress sequences of records in blocks



Internal Layout of a Sequence File



Sync Markers



MapReduce SequenceFile Syntax

```
job.setOutputFormatClass(SequenceFileOutputformat.class);

SequenceFileOutputFormat.setCompressOutput(job,true);

SequenceFileOutputFormat.setOutputCompressorClass(job,GzipCodec.class)
;

SequenceFileOutputFormat.setOutputCompressionType(job,
CompressionType.BLOCK);

job.setInputFormatClass(SequenceFileInputFormat.class);
```


Knowledge Check

Which of the following are true of sequence files?

1. Intermediate data from the Mapper is stored as sequence files
2. Sequence files can be compressed
3. Using sequence files can save space and time in complex workflows comprising multiple MapReduce jobs
4. All of the above
5. 1 & 3

Learning Goals

1. Working with sequence files
2. Working with the distributed cache
3. Working with HBase

Populate the Distributed Cache Using the **hadoop** Command

- **Pass plain text files to Mapper and Reducer classes**

```
hadoop jar . . . --files file1,file2 . . .
```

- **Pass Java jar files to Mapper and/or Reducer classes**

```
hadoop jar . . . --libjars jar1,jar2 . . .
```

- **Pass JAR/TAR or ZIP/GZIP archives to Mapper/Reducer classes**

```
hadoop jar . . . --archives archive1,archive2 . . .
```

 13

Populate the Distributed Cache using the API

```
// driver class

DistributedCache.addCacheFile(new Path("/user/user01/myfile").toUri(),
getConf());

DistributedCache.addCacheArchive(new Path("/user/user01/
myarchive.zip").toUri(), getConf());
```

 14

Describe How to Consume the Distributed Cache

```
// Mapper or Reducer class
. . .
Path[] localFiles, localArchives;

public void setup(Context context) throws IOException {
      Configuration conf = context.getConfiguration();
      localFiles = DistributedCache.getLocalCacheFiles(conf);
      localArchives = DistributedCache.getLocalArchives(conf);
   }

// consume cache in map or reduce method
```

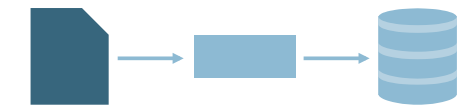
 15

Learning Goals

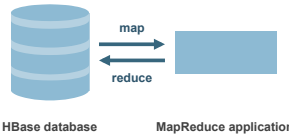
1. Working with sequence files
2. Working with the distributed cache
3. Working with HBase

Using HBase as a Source

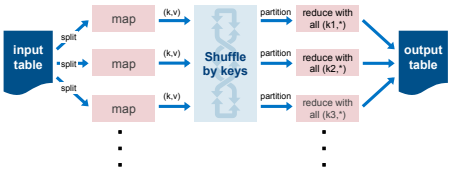
Using HBase as a Sink



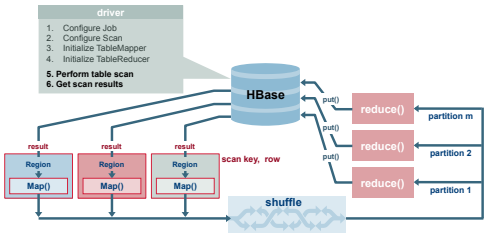
Using HBase as a Source and Sink



Data Flow in MapReduce with HBase



TableMapReduceUtil Model



Use a Flat-Wide Schema

```
create '/mapr/my.cluster.com/user/user01/trades_flat',
{NAME=>'price',VERSIONS=>1000000},
{NAME=>'vol',VERSIONS=>1000000},
{NAME=>'stats'}
```

rowkey	price:00	price:10	…	price:23	vol	stats:min	stats:max	stats:mean
AMZN_20131010								
GOOG_20131010								
CSCO_20131010								
…								

 22

Driver Example

```
public int run(String[] args) throws Exception {
    // set up the job
    Job job = Job.getInstance(getConf(), getClass().getSimpleName());
    job.setJarByClass(getClass());
   //Create and configure a Scan instance.
    Scan scan = new Scan();
    scan.setMaxVersions();
    scan.addFamily(Bytes.toBytes("price");
    // initialize the mapper and reducer
    TableMapReduceUtil.initTableMapperJob(INPUT_TABLE, scan,
            StockMapper.class, Text.class, LongWritable.class, job);
    TableMapReduceUtil.initTableReducerJob(
       OUTPUT_TABLE, StockReducer.class, job);
    return job.waitForCompletion(true) ? 0 : 1;
  }
```

 23

TableMapper Example

```
public class StockMapper extends TableMapper<Text, LongWritable> {

 @Override
 protected void map(ImmutableBytesWritable key, Result row,
     Context context) {
        String keyString = Bytes.toString(key.get());
        final Text outKeyText = new Text(keyString);
        // for all columns in input row
        for (KeyValue col : row.list()) {
           byte[] cellValueBytes = col.getValue();
           long price = Bytes.toLong(cellValueBytes);
           context.write(outKeyText, new LongWritable(price));
        }
  }
}
```

 24

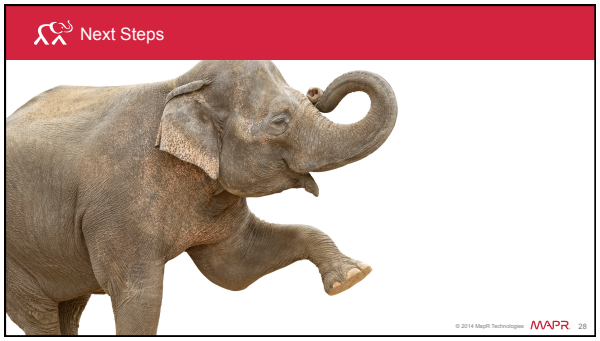
TableReducer Example

```
public class StockReducer extends
     TableReducer<Text, LongWritable, ImmutableBytesWritable> {

  @Override
  protected void reduce(Text key, Iterable<LongWritable> prices,
          Context context) {
      long min=Long.MAX_VALUE;
      long temp = 0L;
      for (LongWritable price : prices) {
         temp=price.get();
         if(temp < min) min=temp;
      }
      Put put = new Put(Bytes.toBytes(key.toString()));
      put.add(OUTPUT_FAMILY, OUTPUT_COLUMN),
          Bytes.toBytes(Float.toString((float)min/100)));
      context.write(key, put);
  }
}
```

Refer to Lab Guide for Exercise Instructions

Summary

1. Working with sequence files
2. Working with the distributed cache
3. Working with HBase


Next Steps
© 2014 MapR Technologies
28

MAPR Academy

Launching Jobs

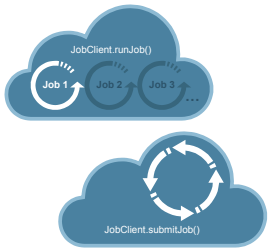
Learning Goals

- Implement programmatic job control in the driver

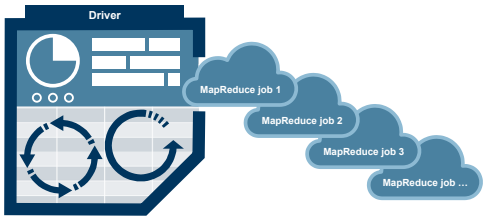
Use MapReduce chaining

Use Oozie to manage MapReduce workflows

Review of the Job Client



Controlling Multiple Jobs in the Driver



 4

Submitting Two Jobs

```
public class TwoJobs {
   public static class Map extends MapReduceBase implements Mapper<...> {
     public void map() { ... }
   }
   public static class Reduce extends MapReduceBase implements
Reducer<...> {
     public void reduce() { ... }
   }
   public static class Map2 extends MapReduceBase implements Mapper<...>
{
     public void map() { ... }
   }
   public static class Reduce2 extends MapReduceBase implements
Reducer<...> {
     public void reduce() { ... }
   }
```

 5

Submitting Two Jobs (2)

```
public static void main(String[] args) throws Exception {
    JobConf conf1 = new JobConf(TwoJobs.class);
    conf1.setMapperClass(Map.class);
    conf1.setReducerClass(Reduce.class);
    JobClient.runJob(conf1);
    JobConf conf2 = new JobConf(TwoJobs.class);
    conf2.setMapperClass(Map2.class);
    conf2.setReducerClass(Reduce2.class);
    JobClient.runJob(conf2);
}
```

 6

Learning Goals

Implement programmatic job control in the driver

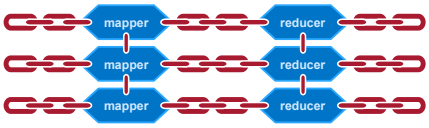
- Use MapReduce chaining

Use Oozie to manage MapReduce workflows

 7

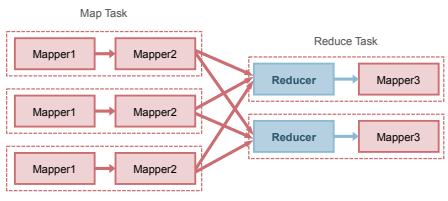
Job Chaining

- `ChainMapper` and `ChainReducer` "chain" successive mappers or successive reducers
- Data transfer is in memory, rather than disk



 8

MapReduce Chaining



Pattern: [MAP+ REDUCE MAP*]

 9

Job Chaining Pattern

```
...
 JobConf conf = new JobConf(true);
...
 JobConf mapAConf = new JobConf(false);
 ChainMapper.addMapper(conf, AMap.class, LongWritable.class, Text.class,
Text.class, Text.class, true, mapAConf);

 JobConf mapBConf = new JobConf(false);
ChainMapper.addMapper(conf, BMap.class, Text.class, Text.class,
   LongWritable.class, Text.class, false, mapBConf);

  JobConf reduceConf = new JobConf(false);
ChainReducer.setReducer(conf, Reduce.class, LongWritable.class, Text.class,
Text.class, IntWritable.class, true, reduceConf);
 …
 JobClient.runJob(conf);
```

 10

Knowledge Check

Which of the following are true of MapReduce chaining

1. Can be used to manage multiple MapReduce jobs
2. Data from mappers are passed onto next mappers or reducers in memory
3. You must use the Mapred package for chaining
4. All of the above
5. 2 & 3 only

 11

Learning Goals

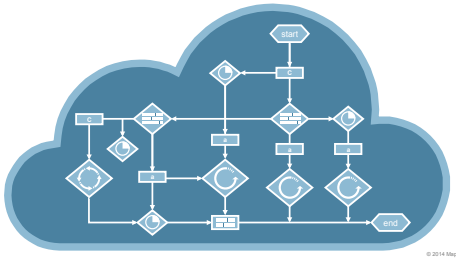
Implement programmatic job control in the driver

Use MapReduce chaining

Use Oozie to manage MapReduce workflows

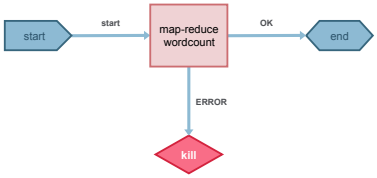
 12

Overview of Oozie



 13

WordCount Workflow DAG



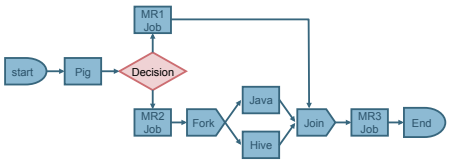
 14

Oozie Workflow XML

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
    <start to='wordcount'/>
    <action name='wordcount'>
        <map-reduce>
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <configuration>
                <property>
                    <name>mapred.mapper.class</name>
                    <value>org.myorg.WordCount.Map</value>
                </property>
                <property>
                    <name>mapred.reducer.class</name>
                    <value>org.myorg.WordCount.Reduce</value>
                </property>
                <property>
                    <name>mapred.input.dir</name>
                    <value>${inputDir}</value>
                </property>
                <property>
                    <name>mapred.output.dir</name>
                    <value>${outputDir}</value>
                </property>
            </configuration>
        </map-reduce>
        <ok to='end'/>
        <error to='end'/>
    </action>
    <kill name='kill'>
        <message>Something went wrong: ${wf:errorCode('wordcount')}</message>
    </kill/>
    <end name='end'/>
</workflow-app>
```

 15

More Complex Workflow Example



Knowledge Check

Apache Oozie is a client-server workflow engine for MapReduce and the ecosystem. Which of the following are true of Oozie?

1. Oozie workflows can be modified by editing the corresponding XML file
2. Workflows are represented as cyclic graphs
3. Oozie workflows handle linear flow
4. All of the above

Refer to Lab Guide for Exercise Instructions

Summary

- Launching multiple MapReduce jobs
- MapReduce Chaining
- Apache Oozie

 19

Next Steps

 20

MAPR Academy

Using non-Java programs (Streaming MapReduce)

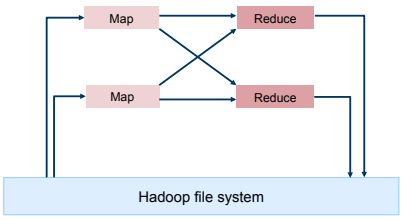
Learning Goals

1. Overview using non-Java programming (streaming)
2. Define the programming contract for mappers & reducers
3. Monitoring and debugging MapReduce streaming jobs

MapReduce Streaming

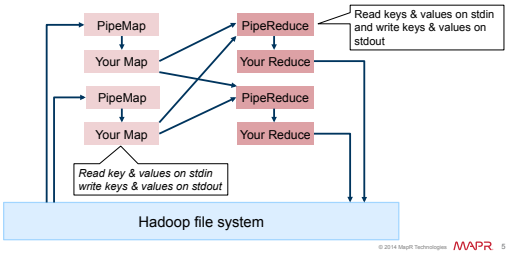
- MapReduce streaming enables using other languages
- Streaming ***!= pipes*** (C++ equivalent)
- Streaming ***may*** introduce some performance penalty
- Streaming ***may*** improve performance

Review the Typical MapReduce Job



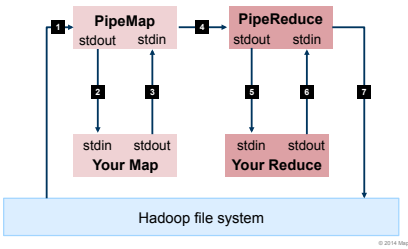
 4

MapReduce Streaming



 5

Data Flow in a Streaming Job



 6

Linux Commands in Streaming (1)

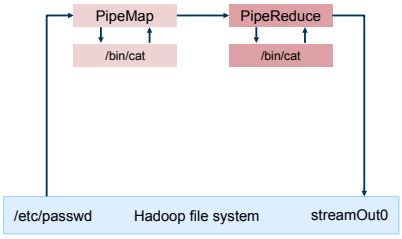
```
# Read input from local filesystem
# Write output to home dir on cluster
•  #  eg. /mapr/my.cluster.com/user/...

export CONTRIB=/opt/mapr/hadoop/hadoop*/contrib/streaming
export STREAMINGJAR=hadoop-*-streaming.jar
export THEJARFILE=$CONTRIB/$STREAMINGJAR

hadoop jar $THEJARFILE \
-input  file:///etc/passwd \
-output  streamOut0 \
-mapper '/bin/cat' \
-reducer '/bin/cat'
```

file or directory, may use –input multiple times

may also specify –combiner script

may specify –inputformat or –outputformat or –partitioner Java classes

Linux Commands in Streaming (2)



Linux Commands in Streaming (3)

more /etc/passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
...
```

more /mapr/my.cluster.com/user/user01/streamOut0/part-00000

```
avahi-autoipd:x:103:110:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false
avahi:x:104:111:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
backup:x:34:34:backup:/var/backups:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
couchdb:x:105:113:CouchDB Administrator,,,:/var/lib/couchdb:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
games:x:5:60:games:/usr/games:/bin/sh
...
```


Knowledge Check

What advantages does MapReduce streaming provide?

1. You can use streaming for rapid prototyping using sed/ awk.
2. It enables you to use languages other than Java (Perl, Python) to write MapReduce programs
3. It definitely enhances performance
4. All of the above
5. 1 & 2

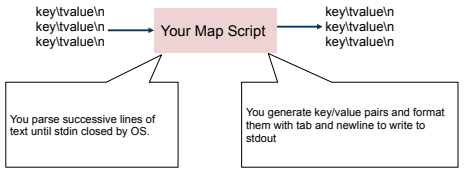
 10

Learning Goals

1. Overview using non-Java programming (streaming)
2. Define the programming contract for mappers & reducers
3. Monitoring and debugging MapReduce streaming jobs

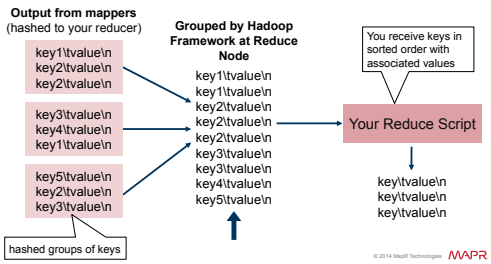
 11

Data Flow Through Mapper

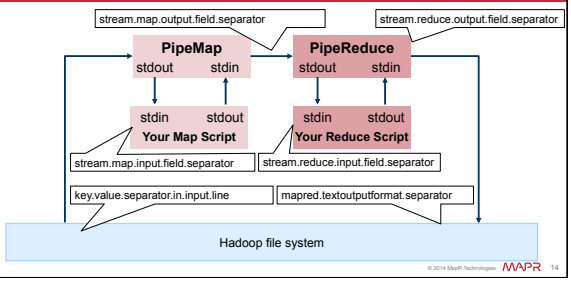


 12

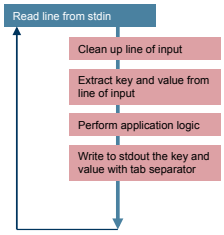
Data Flow Through Reducer



Configure Field Separators



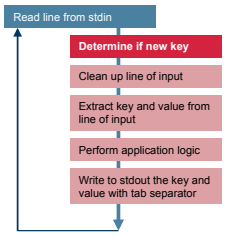
Mapper Data Processing



Perl Example: Mapper

```
#!/usr/bin/env perl
while (<>) {      # read stdin
  chomp;    # remove last char if newline from the implicit variable $_
  my ( $key,$value ) = split(/\t/,$_);     # extract key and value
  print "key:".$key."\t"."value:".$value."\n";     # write to stdout
}
```


Reducer Data Processing



Reducer Data Processing

```
#!/usr/bin/env python
import sys

currentKey = None
groupTotal = 0

for line in sys.stdin:
  line = line.strip()  # remove leading and trailing whitespace
  if not line:    # ingnore empty lines
      continue

  keyFromLine, valueFromLine = line.split("\t", 1)

  if currentKey == keyFromLine:  # key hasn't changed
      groupTotal += int(valueFromLine)
  else:   # we just got new key
      if currentKey:  # if not first line of processing
          print( "%s\t%d" % (currentKey, groupTotal) )
      groupTotal = int(valueFromLine)  # total is just our current valueFromLine
      currentKey = keyFromLine  # set the new key as our currentKey

# The last key must be processed
if currentKey == keyFromLine:
  print( "%s\t%d" % (currentKey, groupTotal) )
```


Python MapReduce Job

```
export CONTRIB=/opt/mapr/hadoop/hadoop*/contrib/streaming
export STREAMINGJAR=hadoop-*-streaming.jar
export THEJARFILE=$CONTRIB/$STREAMINGJAR
rm -rf /mapr/my.cluster.com/user/user01/pyNoOpMapRed 2>>/dev/null
hadoop jar $THEJARFILE \
   -D mapred.reduce.tasks=2 \
   -mapper "python noOpMap.py" \
   -file noOpMap.py  \
   -reducer "python noOpReduce.py" \
   -file noOpReduce.py  \
   -input file:///$PWD/goodInput.txt   \
   -output "pyNoOpMapRed"
```


Launching Perl Streaming Code

```
#!/usr/bin/env bash
export CONTRIB=/opt/mapr/hadoop/hadoop*/contrib/streaming
export STREAMINGJAR=hadoop-*-streaming.jar
export THEJARFILE=$CONTRIB/$STREAMINGJAR

hadoop jar $THEJARFILE \
  -mapper 'perl map.pl' \
  -reducer 'perl reduce.pl' \
  -file map.pl \
  -file reduce.pl \
  -input  file:///etc/passwd \
  -output  perlOut
```


Learning Goals

1. Overview using non-Java programming (streaming)
2. Define the programming contract for mappers & reducers
3. Monitoring and debugging MapReduce streaming jobs

General Debugging Tips

- Launch map or reduce script ***"standalone" with stdin***
- Test with ***bad data***
- Run a ***map-only*** job to test mapper
- Run ***reduce-only*** with an "identity mapper" that you create
- Use ***counters and status***

Perl Mapper Updating a Counter

Perl

```
. . .
print STDERR "reporter:counter: MyGroup,NumberOfRecords,100\n";
```

Python

```
. . .
sys.stderr.write("reporter:counter: Mygroup, NumberOfRecords,100\n"
```


Perl Reducer Updating Status

```
#!/usr/bin/env perl
my $numRecords = 0;
while (<>) {       # read from stdin
  my ($key,$value) = split(/\t/);      # split input on newline
  $numRecords += 1;                    # count num of records
}
print "$numRecords\n";              # after all stdin processed
                                    # write the number of records
print STDERR "reporter:status:A Reducer Exiting\n";
```


Refer to Lab Guide for Exercise Instructions

Summary

- Non-Java programming
- Configure parameters
- Monitor and debug

 26

Next Steps

 27