CMPE 282 Cloud Services ***MapReduce Lab***

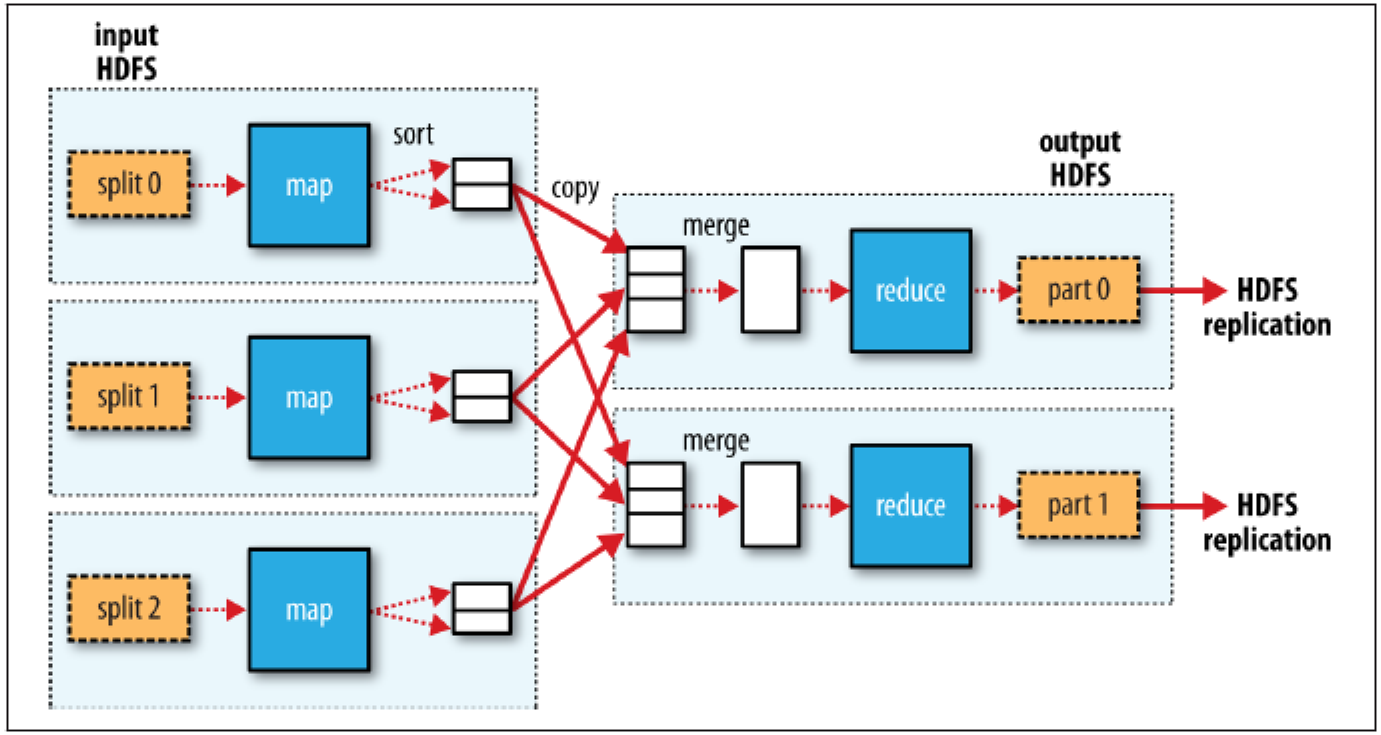
Instructor: Kong Li

Content

- HDFS Cmd Line Interface
- MapReduce Tutorial
- MapReduce Considerations
- Common Mistakes
- HW

MapReduce: Refresher

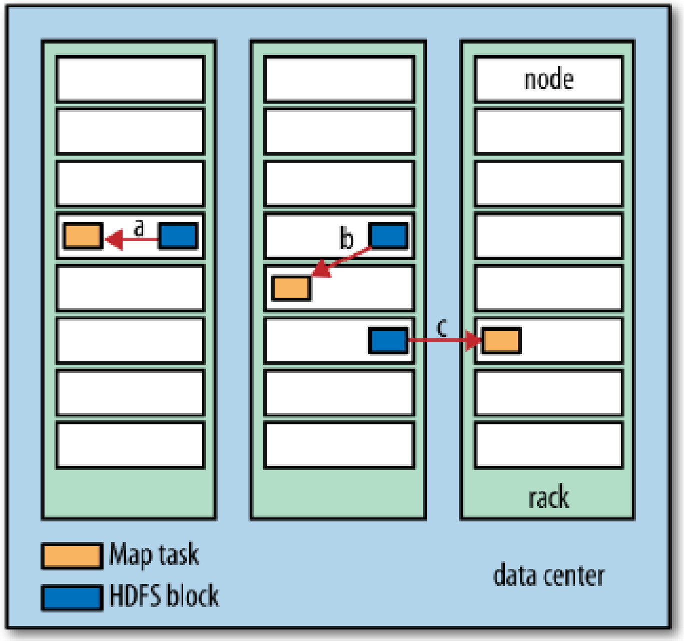
- Required
  - map: (K1, V1) → list(K2, V2)
  - reduce: (K2, list(V2)) → list(K3, V3)
- Optional
  - Combiner F: (K2, list(V2)) → list(K2, V2)
    - Part of map phase
    - Often the combiner and reduce functions are the same
  - Partition F: (K2, V2) → integer



- HDFS: job input & output
- Driver program
- JobTracker, Task Tracker
- Resource manager, node manager

MapReduce Job

- Input data: divided into multiple *splits*
  - By default, 128 MB (HDFS block size)
- Map task: (K1, V1) → list(K2, V2)
  - Input: HDFS
    - Data locality: data-local, rack-local, off-rack
  - Output: local disk (why?)
  - One map task per split
    - Run map function for each *record in the split*
- Reduce task: (K2, list(V2)) → list(K3, V3)
  - Input: shuffle-and-sorted map output's intermediate key-value pairs by key
    - Each reduce task can be fed by many map tasks
  - Output: HDFS
  - # of reduce tasks: specified independently
    - Nothing to do with input data size
    - In driver by default, job.setNumReduceTasks(1);
    - OK to have 0 reduce task



Trade-offs:
More splits → more map tasks, more parallelism, higher overhead
Less splits → less map tasks, more sequentiality, less overhead

HDFS Cmd Line

```
$ hadoop -help
```

- Copy file from local disk to HDFS

```
$ hadoop fs -copyFromLocal input/docs/quangle.txt \
hdfs://localhost/user/tom/quangle.txt
$ hadoop fs -copyFromLocal input/docs/quangle.txt /user/tom/quangle.txt
$ hadoop fs -copyFromLocal input/docs/quangle.txt quangle.txt
```

- Copy file from HDFS to local disk

```
$ hadoop fs -copyToLocal quangle.txt quangle.copy.txt
```

- Makedir on HDFS

```
$ hadoop fs -mkdir -p books
```

- List files on HDFS

```
$ hadoop fs -ls .
Found 2 items
drwxr-xr-x - tom supergroup 0 2014-10-04 13:22 books
-rw-r--r-- 1 tom supergroup 119 2014-10-04 13:21 quangle.txt
```

- Parallel copying

```
$ hadoop distcp file1 file2
```

WordCount

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

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

  public void map(Object key, Text value, Context context)
               throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      context.write(new Text(itr.nextToken()), new IntWritable(1));
    }
  }
 }

public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {

   public void reduce(Text key, Iterable<IntWritable> values,
                Context context)
                throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
     sum += val.get();
    }
    context.write(key, new IntWritable(sum));
   }
 }
 public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();
   Job job = Job.getInstance(conf, "word count");
   job.setJarByClass(WordCount.class);
   job.setMapperClass(TokenizerMapper.class);
   job.setCombinerClass(IntSumReducer.class);
   job.setReducerClass(IntSumReducer.class);
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);
   FileInputFormat.addInputPath(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));
   System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```

WordCount - map

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

    public void map(Object key, Text value, Context context)
                throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        context.write(new Text(itr.nextToken()), new IntWritable(1));
      }
    }
}
```

keyin, valuein, keyout, valueout

Classes must match keyin, valuein

Classes must match keyout, valueout

Input file01:
Hello World Bye World
Input file02:
Hello Hadoop Goodbye Hadoop

map 1 output:
<Hello, 1>
<World, 1>
<Bye, 1>
<World, 1>

map 2 output:
<Hello, 1>
<Hadoop, 1>
<Goodbye, 1>
<Hadoop, 1>

WordCount – map + combiner

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

  public void map(Object key, Text value, Context context)
              throws IOException, InterruptedException {
   StringTokenizer itr = new StringTokenizer(value.toString());
   while (itr.hasMoreTokens()) {
    context.write(new Text(itr.nextToken()), new IntWritable(1));
   }
  }
}
```

keyin, valuein, keyout, valueout

Classes must match keyin, valuein

Classes must match keyout/valueout

Input file01:
Hello World Bye World
Input file02:
Hello Hadoop Goodbye Hadoop

map+combiner 1 output:
<Bye, 1>
<Hello, 1>
<World, 2>

map+combiner 2 output:
<Goodbye, 1>
<Hadoop, 2>
<Hello, 1>

Combiner's keyin/valuein must match map's keyout/valueout
Combiner's keyout/valueout must match reduce's keyin/valuein

WordCount - reduce

keyin, valuein, keyout, valueout
keyin/valuein must match map's keyout/valueout

```
public static class IntSumReducer
     extends Reducer<Text, IntWritable, Text, IntWritable> {
```

Classes must match keyin, valuein

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

Classes must match keyout/valueout

reduce output:

```
<Bye, 1>
<Goodbye, 1>
<Hadoop, 2>
<Hello, 2>
<World, 2>
```

WordCount - driver

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

  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);

  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);

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

set the output class for <u>both</u> map and reduce

WordCount - run

- Assume env variables JAVA_HOME and PATH are set

```
$ export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
```

- Compile WordCount.java and create a jar

```
$ hadoop com.sun.tools.javac.Main WordCount.java
$ jar cf wc.jar WordCount*.class
```

- Assume /user/joe/wordcount/input as input, …/output as output

```
$ hadoop fs -ls /user/joe/wordcount/input/
/user/joe/wordcount/input/file01 /user/joe/wordcount/input/file02
$ hadoop fs -cat /user/joe/wordcount/input/file01
Hello World Bye World
$ hadoop fs -cat /user/joe/wordcount/input/file02
Hello Hadoop Goodbye Hadoop
```

- Run the app

```
$ hadoop jar wc.jar WordCount /user/joe/wordcount/input /user/joe/wordcount/output
```

- Output

```
$ hadoop fs -cat /user/joe/wordcount/output/part-r-00000
Bye 1
Goodbye 1
Hadoop 2
Hello 2
World 2
```

Left cmd lines can be replaced with

```
$ javac -classpath `hadoop classpath` -d . WordCount.java
$ jar cf wc.jar WordCount*.class
$ export HADOOP_CLASSPATH=wc.jar
…
$ hadoop WordCount /user/joe/wordcount/input \
/user/joe/wordcount/output
```

Map, reduce, and driver can be three separate classes – see WordCount2

Common Mistakes to Avoid

- Mapper and reducer should be stateless
  - No static variables - after `map` + `reduce` return, they should remember nothing about the processed data!
  - Why: No guarantees about which key-value pairs will be processed by which workers!
  - There are exceptions – see Top Ten pattern

```
HashMap h = new HashMap();
map(key, value) {
  if (h.contains(key)) {
    h.add(key,value);
    emit(key, "X");
  }
}
```

- Don't do your own I/O
  - Don't try to read from, or write to, files in the file system
  - The MapReduce framework does all the I/O for you:
    - All the incoming data will be fed as arguments to map and reduce
    - Any data your functions produce should be output via context.write

```
map(key, value) {
  File foo =
    new File("xyz.txt");
  while (true) {
    s = foo.readLine();
    ...
  }
}
```

Common Mistakes to Avoid (cont’d)

```
map(key, value) {
  emit("FOO", key + " " + value);
}
```

```
reduce(key, value[]) {
  /* do some computation on
  all the values */
}
```

- Mapper should not map too much data to the same key
  - Avoid mapping *everything* to the same key
    - Otherwise the reduce worker will be overwhelmed
  - OK if some reduce workers have more work than others
    - Ex: In WordCount, the reduce worker that works on the key 'and' has a lot more work than the reduce worker that works on 'syzygy'

MapReduce: Considerations

- Map: execution order not deterministic; processing time unpredictable
- Reduce tasks cannot start before all Maps have finished
- Not suitable for continuous input streams
- Spike in network util% after Map / before Reduce phase
- Number & size of key/value pairs: Obj creation & serialization overhead
- Aggregate partial results when possible ➔ Use combiner
- How many map tasks?
  - Smaller splits + many mappers vs larger splits + fewer mappers
    - Tradeoffs: Resource consumption, parallelism
  - Split (size) can be controlled by InputFormat.getSplits(), or job property mapreduce.input.fileinputformat.split.maxsize
- How many reduce tasks? usually determined by the algorithm
- Locality: Master tries to do work on nodes that have replicas of the data
- Fault tolerance: JobTracker re-executes the failed node's task(s)
- Speculative execution: master can deal with stragglers (slow machines) by re-executing their tasks somewhere else

Designing MapReduce Algorithms

- Key decision: What should be done by `map`, and what by `reduce`?
  - `map` can do something to each individual key-value pair, but it can't look at other key-value pairs
    - Ex: Filtering out key-value pairs we don't need
  - `map` can emit more than one intermediate key-value pair for each incoming key-value pair
    - Ex: Incoming data is text line, `map` produces (word,1) for each word
  - Output value from `map` is a class which can have several properties
    - Ex: `Map` output can be (key, {min, max})
  - `reduce` can aggregate data; it can look at multiple values, as long as `map` has mapped them to the same (intermediate) key
    - Ex: Count the number of words, add up the total cost, ...
- Need to get the intermediate format right
  - If `reduce` needs to look at several values together, `map` must emit them using the same key
- Multiple MapReduce jobs can be chained together

Minimal MapReduce driver, with defaults explicitly set

```
public class MinimalMapReduceWithDefaults extends Configured implements Tool {
  @Override
  public int run(String[] args) throws Exception {
    Job job = JobBuilder.parseInputAndOutput(this, getConf(), args);
    if (job == null) {
      return -1;
    }
    job.setInputFormatClass(TextInputFormat.class);
    job.setMapperClass(Mapper.class);
    job.setMapOutputKeyClass(LongWritable.class);
    job.setMapOutputValueClass(Text.class);
    job.setPartitionerClass(HashPartitioner.class);
    job.setNumReduceTasks(1);
    job.setReducerClass(Reducer.class);
    job.setOutputKeyClass(LongWritable.class);
    job.setOutputValueClass(Text.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    return job.waitForCompletion(true) ? 0 : 1;
  }
  public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new MinimalMapReduceWithDefaults(), args);
    System.exit(exitCode);
  }
}
```

each line of each input file as a separate record

setOutputKeyClass() and setOutputValueClass() set the output class for both map and reduce. If they are different, call setMapOutputKeyClass() and setMapOutputValueClass()

(key, value) pairs as "key \tvalue" on individual lines of a text file

CMPE vCenter Server Lab Rules

- You have permission only on
  - Resource Pool: **CMPE282 SEC1**
  - VM folder: **CMPE LABS/CMPE282 SEC1/workspace**
  - Datastore: **Classroom**

Your own area: a folder **YourName-L3SID** under workspace

- Naming convention for any newly-created VM/template/vApp
  - Naming convention: **<YourName>-<os><version>-<L3SID>** , e.g., john-ub1404-123
  - If necessary (avoid more naming collision), append **-1**, **-2**, etc at the end
- VM creation rules:
  - You can create VM only based on template, ISO file, or OVF/OVA **provided by the instructor**
    - Any created VM must connect to a specified network **without** internet access, unless allowed by instructor
  - You are **not** allowed to create VMs based on **your** uploaded ISO file or OVF/OVA
- Connection to vCenter Server and VM console
  - Web client: supported browser + several plug-ins (client support + remote console)
  - VM console (Web client or VI client): require SJSU VPN
  - ssh
- It is a shared environment
  - Be responsible - never disrupt other users
  - Clean up (**power off** or **delete** VM) as soon as you finish – penalty if you fail to do so
  - **Any malicious action will face discipline**

Login acct: vsphere.local\cmpe282_sec1_student

HW

- See Canvas for details

References

- http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
- https://github.com/tomwhite/hadoop-book/archive/master.zip
- https://github.com/adamjshook/mapreducepatterns/archive/master.zip
  - Get pom.xml from https://pragmaticintegrator.wordpress.com/2014/09/09/running-mapreduce-design-patterns-on-cloudera-cdh5/
- Cloudera Quickstart doc https://www.cloudera.com/documentation/enterprise/5-10-x/topics/quickstart.html