CPS216: Advanced Database Systems
(Data-intensive Computing Systems)

How MapReduce Works (in Hadoop)

Shivnath Babu
Lifecycle of a MapReduce Job

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

        public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                output.collect(word, one);
            }
        }
    }

    public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
            int sum = 0;
            while (values.hasNext()) { sum += values.next().get(); }
            output.collect(key, new IntWritable(sum));
        }
    }

    public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(WordCount.class);
        conf.setJobName("wordcount");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        TextInputFormat.setInputPaths(conf, new Path(args[0]));
        TextOutputFormat.setOutputPath(conf, new Path(args[1]));
        JobClient.runJob(conf);
    }
}
```

Map function
Reduce function
Run this program as a MapReduce job
Lifecycle of a MapReduce Job

```java
public class WordCount {
    public static class Map extends MapReduceBase implements 
        Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        
        public void map(LongWritable key, Text value, 
            OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                output.collect(word, one);
            }
        }
    }

    public static class Reduce extends MapReduceBase implements 
        Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterator<IntWritable> values, 
            OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
            int sum = 0;
            while (values.hasNext()) { sum += values.next().get(); }
            output.collect(key, new IntWritable(sum));
        }
    }

    public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(WordCount.class);
        conf.setJobName("wordcount");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Map.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        TextInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));
        JobClient.runJob(conf);
    }
}
```
Lifecycle of a MapReduce Job

Input Splits  Map Wave 1  Map Wave 2  Reduce Wave 1  Reduce Wave 2
Components in a Hadoop MR Workflow

- **JobTracker (Master)**
  - Accepts MR jobs
  - Assigns tasks to workers
  - Monitors tasks
  - Handles failures

- **TaskTracker (Worker)**
  - Run Map and Reduce tasks
  - Manage intermediate output
  - A separate process
  - Run the Map/Reduce functions

- **Client**
  - UI for submitting jobs
  - Get various status information

Next few slides are from: http://www.slideshare.net/hadoop/practical-problem-solving-with-apache-hadoop-pig
Job Submission

1. Copy Input Files
2. Submit Job
3. Get Input Files’ info
4. Create Splits
5. Upload job information
6. Submit Job
Initialization
Scheduling
Execution

JobTracker

Task Tracker

DFS

Assign task for Execution

Read into Local Disk

Job.xml, Job.jar

Upto MAX_MAP_SLOTS Map Task JVMs Concurrently

Upto MAX_REDUCE_SLOTS Reduce Task JVMs Concurrently
Map Task

- User's Map Method
  - Deserialized Key/Value
  - User's Record Reader
  - Input Split

- Sort Buffer
  - User's Partitioner/Combiner
  - Sort/Combine & Spill
  - Local Disk
  - In-Mem Spill index

- Every map task does a small sort

- Record info (K,V offsets and Partition) maintained

- io.sort.mb
Sort Buffer

io.sort.mb

Index

Index and partition buffers (int arrays)

io.sort.record.percent * io.sort.mb

P KS VS

Sort buffer, bytes array

Wrap around buffer

io.sort.mb*io.sort.spill.percent
Reduce Tasks
Quick Overview of Other Topics (Will Revisit Them Later in the Course)

• Dealing with failures
• Hadoop Distributed FileSystem (HDFS)
• Optimizing a MapReduce job
Dealing with Failures and Slow Tasks

• What to do when a task fails?
  – Try again (retries possible because of idempotence)
  – Try again somewhere else
  – Report failure

• What about slow tasks: stragglers
  – Run another version of the same task in parallel. Take results from the one that finishes first
  – What are the pros and cons of this approach?

Fault tolerance is of high priority in the MapReduce framework
HDFS Architecture

Metadata (Name, replicas, ...): /home/foo/data, 3, ...

Client
- Read Datanodes
- Write

Rack 1
- Datanodes
- Replication

Rack 2
- Datanodes
- Blocks

Namenode
- Metadata.ops
- Block.ops
How are the number of splits, number of map and reduce tasks, memory allocation to tasks, etc., determined?
Job Configuration Parameters

- 190+ parameters in Hadoop
- Set manually or defaults are used
Hadoop Job Configuration Parameters

Client Node

```java
main() {
    JobConf jobConf = new JobConf();
    JobClient.runJob(jobConf);
}
```

Job Tracker Node

```java
public JobStatus submitJob() {
    read job.xml from hdfs
    read job.split from hdfs
    init map/reduce task
    add task queue
}
```

Task Tracker Node

```java
public void run() {
    while(true) {
        List<Task> tasks = jobTracker.heartbeat();
        runTask(tasks);
    }
}
```

User Task

```java
public void run() {
    Reader reader = inputFormat.getReader();
    while(reader.next()) {
        mapper.map(key, value...);
    }
}
```
Tuning Hadoop Job Conf. Parameters

• Do their settings impact performance?
• What are ways to set these parameters?
  – Defaults -- are they good enough?
  – Best practices -- the best setting can depend on data, job, and cluster properties
  – Automatic setting
Experimental Setting

• Hadoop cluster on 1 master + 16 workers

• Each node:
  – 2GHz AMD processor, 1.8GB RAM, 30GB local disk
  – Relatively ill-provisioned!
  – Xen VM running Debian Linux
  – Max 4 concurrent maps & 2 reduces
    • Maximum map wave size = 16x4 = 64
    • Maximum reduce wave size = 16x2 = 32

• Not all users can run large Hadoop clusters:
  – Can Hadoop be made competitive in the 10-25 node, multi GB to TB data size range?
### Parameters Varied in Experiments

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description and Use</th>
<th>Default Value</th>
<th>Values Considered</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapred.reduce.tasks</td>
<td>Number of reducer tasks</td>
<td>1</td>
<td>[5, 300]</td>
</tr>
<tr>
<td>io.sort.factor</td>
<td>Number of sorted streams to merge at once during sorting</td>
<td>10</td>
<td>[10, 500]</td>
</tr>
<tr>
<td>io.sort.mb</td>
<td>Size in MegaBytes of map-side buffer for sorting</td>
<td>100</td>
<td>[100, 200]</td>
</tr>
<tr>
<td>io.sort.record.percent</td>
<td>Fraction of io.sort.mb dedicated to metadata storage</td>
<td>0.05</td>
<td>[0.05, 0.15]</td>
</tr>
<tr>
<td>io.file.buffer.size</td>
<td>Buffer size used to read/write (intermediate) sequence files</td>
<td>4K</td>
<td>32K</td>
</tr>
<tr>
<td>mapred.child.java.opts</td>
<td>Java control options for all mapper and reducer tasks</td>
<td>-Xmx200m</td>
<td>-Xmx[200m, 300m]</td>
</tr>
<tr>
<td>mapred.inmem.merge.threshold</td>
<td>Reduce-side trigger for in-memory merging; off when 0</td>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>mapred.job.shuffle.input.buffer.percent</td>
<td>% of reducer task’s heap to buffer map outputs</td>
<td>0.7</td>
<td>{0.7, 0.8}</td>
</tr>
<tr>
<td>mapred.job.shuffle.merge.percent</td>
<td>Usage threshold of mapred.job.shuffle.input.buffer.percent to trigger reduce-side merge in parallel with the copying of map outputs</td>
<td>0.66</td>
<td>{0.66, 0.8}</td>
</tr>
<tr>
<td>mapred.job.reduce.input.buffer.percent</td>
<td>% of reducer task’s heap to buffer map outputs while applying reduce</td>
<td>0</td>
<td>{0, 0.8}</td>
</tr>
<tr>
<td>dfs.replication</td>
<td>Block replication factor in Hadoop’s HDFS filesystem</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>dfs.block.size</td>
<td>HDFS block size (equal to amount of data processed per mapper task)</td>
<td>64MB</td>
<td>128MB</td>
</tr>
</tbody>
</table>
### Hadoop 50GB TeraSort

<table>
<thead>
<tr>
<th>Row#</th>
<th>mapred. reduce.tasks</th>
<th>io.sort. factor</th>
<th>io.sort.record. percent</th>
<th>Job Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>0.10</td>
<td>1hrs, 25mins, 25sec</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>0.15</td>
<td>1hrs, 14mins, 54sec</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>500</td>
<td>0.10</td>
<td>1hrs, 7mins, 11sec</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>500</td>
<td>0.15</td>
<td>1hrs, 1mins, 1sec</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>10</td>
<td>0.10</td>
<td>1hrs, 22mins, 54sec</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
<td>10</td>
<td>0.15</td>
<td>1hrs, 4mins, 57sec</td>
</tr>
<tr>
<td>7</td>
<td>28</td>
<td>500</td>
<td>0.10</td>
<td>1hrs, 22mins, 24sec</td>
</tr>
<tr>
<td>8</td>
<td>28</td>
<td>500</td>
<td>0.15</td>
<td>1hrs, 3mins, 46sec</td>
</tr>
<tr>
<td>9</td>
<td>300</td>
<td>10</td>
<td>0.10</td>
<td>45mins, 22sec</td>
</tr>
<tr>
<td>10</td>
<td>300</td>
<td>10</td>
<td>0.15</td>
<td>35mins, 9sec</td>
</tr>
<tr>
<td>11</td>
<td>300</td>
<td>500</td>
<td>0.10</td>
<td>44mins, 38sec</td>
</tr>
<tr>
<td>12</td>
<td>300</td>
<td>500</td>
<td>0.15</td>
<td>35mins, 56sec</td>
</tr>
</tbody>
</table>

- Varying number of reduce tasks, number of concurrent sorted streams for merging, and fraction of map-side sort buffer devoted to metadata storage
• Varying number of reduce tasks for different values of the fraction of map-side sort buffer devoted to metadata storage (with io.sort.factor = 500)
Hadoop 50GB TeraSort

- Varying number of reduce tasks for different values of io.sort.factor (io.sort.record.percent = 0.05, default)
Hadoop 75GB TeraSort

- 1D projection for io.sort.factor=500
Automatic Optimization? (Not yet in Hadoop)

Map Wave 1  Map Wave 2  Map Wave 3  Reduce Wave 1  Reduce Wave 2  Reduce Wave 3

What if #reduces increased to 9?