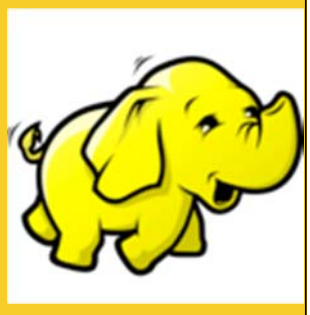
Map Reduce Workflows

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/hadoop-tutorial/

© 2012 coreservlets.com and Dima May



Agenda

- Workflows Introduction
- Decomposing Problems into MapReduce Workflow
- Using JobControl class

MapReduce Workflows

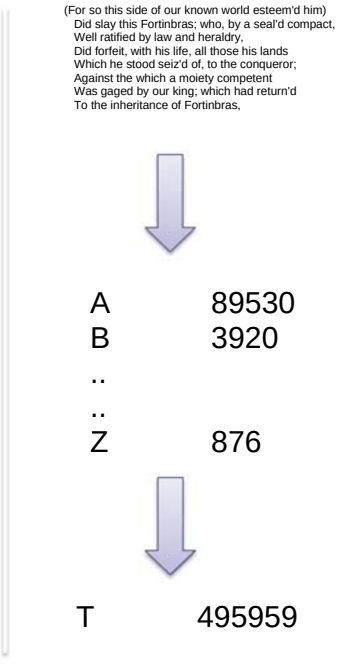
- We've looked at single MapReduce job
- Complex processing requires multiple steps
  - Usually manifest in multiple MapReduce jobs rather than complex map and reduce functions
- May also want to consider higher-level MapReduce abstractions
  - Pig, Hive, Cascading, Cascalog, Crunch
  - Focus on business logic rather than MapReduce translation
  - On the other hand you'll need to learn another syntax and methodology
- This lecture will focus on building MapReduce Workflows

Decomposing Problems into MapReduce Jobs

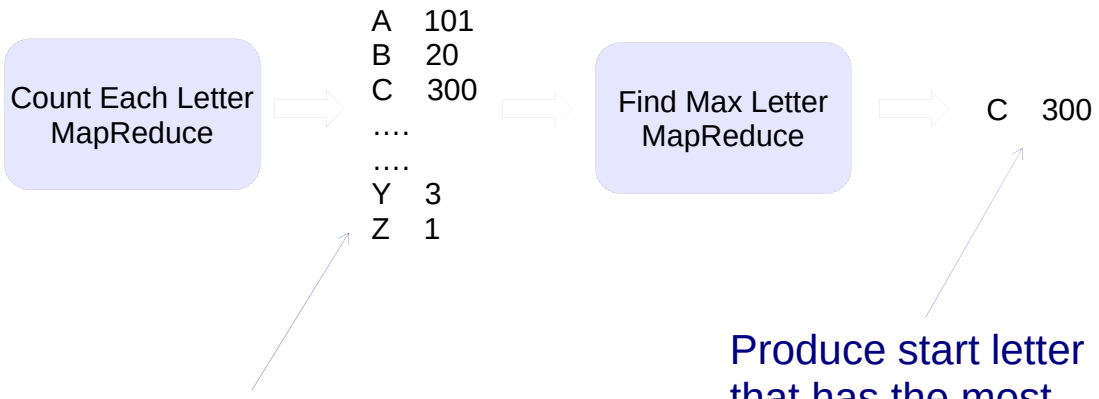
- **Small map-reduce jobs are usually better**
  - Easier to implement, test and maintain
  - Easier to scale and re-use
- **Problem:**
- **Find a letter that occurs the most in the provided body of text**

Decomposing the Problem

- **Calculate number of occurrences of each letter in the provided body of text**
- **Traverse each letter comparing occurrence count**
- **Produce start letter that has the most occurrences**



Decomposing the Problem

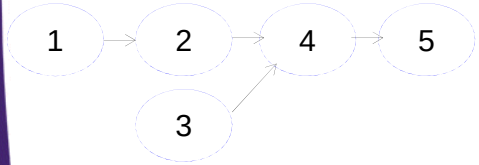


Calculate number of occurrences of each letter in the provided body of text

Produce start letter that has the most occurrences

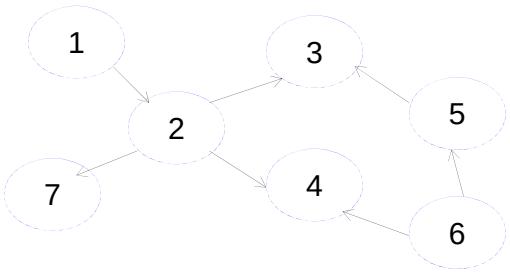
MapReduce Workflows

- **Your choices can depend on complexity of workflows**
  - Linear chain or simple set of dependent Jobs vs. Directed Acyclic Graph (DAG)
  - http://en.wikipedia.org/wiki/Directed_acyclic_graph



Simple Dependency
or
Linear chain

vs.



Directed Acyclic Graph (DAG)

MapReduce Workflows

- **JobControl class**
  - Create simple workflows
  - Represents a graph of Jobs to run
  - Specify dependencies in code
- **Oozie**
  - An engine to build complex DAG workflows
  - Runs in its own daemon
  - Describe workflows in set of XML and configuration files
  - Has coordinator engine that schedules workflows based on time and incoming data
  - Provides ability to re-run failed portions of the workflow

Workflow with JobControl

1. **Create JobControl**
   - Implements java.lang.Runnable, will need to execute within a Thread
2. **For each Job in the workflow Construct ControlledJob**
   - Wrapper for Job instance
   - Constructor takes in dependent jobs
3. **Add each ControlledJob to JobControl**
4. **Execute JobControl in a Thread**
   - Recall JobControl implements Runnable
5. **Wait for JobControl to complete and report results**
   - Clean-up in case of a failure

1: Create JobControl

```
public class MostSeenStartLetterJobControl
              extends Configured implements Tool{

   private final Logger log =
      Logger.getLogger(MostSeenStartLetterJobControl.class);

   @Override
   public int run(String[] args) throws Exception {
      String inputText = args[0];
      String finalOutput = args[1];

      String intermediateTempDir = "/" +
                     getClass().getSimpleName() + "-tmp";
      Path intermediatePath = new Path(intermediateTempDir);
       deleteIntermediateDir(intermediatePath);

       try {
          JobControl control =
             new JobControl("Worklfow-Example");
...
...
```

Manage intermediate output directory

JobControl manages workflow

2: For Each Job in the Workflow Construct ControlledJob

```
...
...
ControlledJob step1 = new ControlledJob(
       getCountJob(inputText, intermediateTempDir), null);


ControlledJob step2 = new ControlledJob(
       getMostSeenJob(intermediateTempDir, finalOutput),
                     Arrays.asList(step1));
...
...
```

Returns Job object with properly configured job

Specify dependencies, In this case step2 depends on step1

2: Build First Job

```
private Job getCountJob(String inputText, String tempOutputPath)
throws IOException {
  Job job = Job.getInstance(getConf(), "StartsWithCount");
  job.setJarByClass(getClass());

  // configure output and input source
  TextInputFormat.addInputPath(job, new Path(inputText));
  job.setInputFormatClass(TextInputFormat.class);

  // configure mapper and reducer
  job.setMapperClass(StartsWithCountMapper.class);
  job.setCombinerClass(StartsWithCountReducer.class);
  job.setReducerClass(StartsWithCountReducer.class);

  // configure output
  TextOutputFormat.setOutputPath(job, new Path(tempOutputPath));
  job.setOutputFormatClass(TextOutputFormat.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);

  return job;
}
```

Build each job the same way we've done before

2: Build Second Job

```
private Job getMostSeenJob(String intermediateTempDir,
             String finalOutput) throws IOException {
  Job job = Job.getInstance(getConf(), "MostSeenStartLetter");
  job.setJarByClass(getClass());

  // configure output and input source
  KeyValueTextInputFormat.addInputPath(job,
              new Path(intermediateTempDir));
  job.setInputFormatClass(KeyValueTextInputFormat.class);

  // configure mapper and reducer
  job.setMapperClass(MostSeenStartLetterMapper.class);
  job.setCombinerClass(MostSeendStartLetterReducer.class);
  job.setReducerClass(MostSeendStartLetterReducer.class);

  // configure output
  TextOutputFormat.setOutputPath(job, new Path(finalOutput));
  job.setOutputFormatClass(TextOutputFormat.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);

  return job;
}
```

Build each job the same way we've done before

3: Add Each ControlledJob to JobControl

```
...
...
control.addJob(step1);
control.addJob(step2);
...
...
```


4: Execute JobControl in a Thread

```
...
...
Thread workflowThread = new Thread(control,
          "Workflow-Thread");

workflowThread.setDaemon(true);
workflowThread.start();

...
...
```

5: Wait for JobControl to Complete

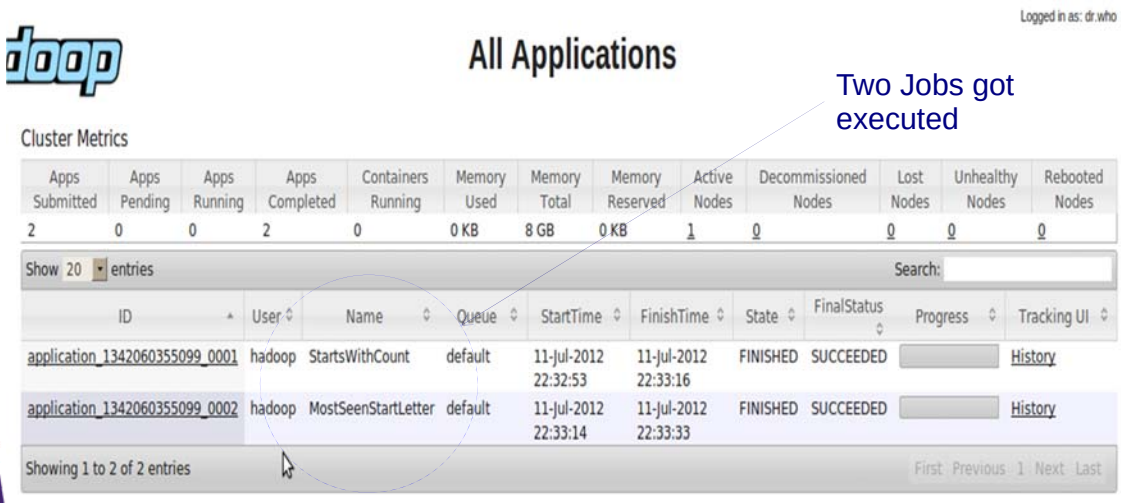
```
...
...
while (!control.allFinished()){
  Thread.sleep(500);
}

if ( control.getFailedJobList().size() > 0 ){
  log.error(control.getFailedJobList().size() +
       " jobs failed!");

  for ( ControlledJob job : control.getFailedJobList()){
       log.error(job.getJobName() + " failed");
  }

} else {
  log.info("Success!! Workflow completed [" +
       control.getSuccessfulJobList().size() + "] jobs");
}
...
...
```

Simply wait for all jobs to complete

Report results or Display errors

Run JobControl Example

```
yarn jar $PLAY_AREA/HadoopSamples.jar \
      mr.workflows.MostSeenStartLetterJobControl \
      /training/data/hamlet.txt \
      /training/playArea/wordCount
```



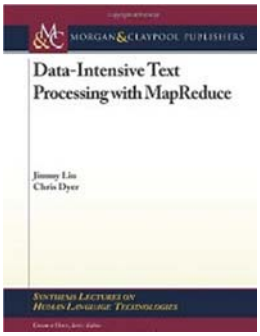
Two Jobs got executed

JobControl Workflow Result

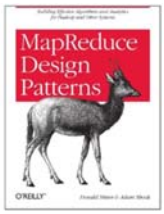
```
$ hdfs dfs -cat /training/playArea/wordCount/part-r-00000
t       3711
```


Study MapReduce Algorithms

- **"Data-Intensive Text Processing with MapReduce" by Jimmy Lin and Chris Dyer**
- **Download the book for free**
  - http://lintool.github.com/MapReduceAlgorithms/index.html
- **Buy a copy**
  - http://www.amazon.com/Data-Intensive-Processing-MapReduce-Synthesis-Technologies/dp/1608453421/ref=sr_1_1?ie=UTF8&qid=1340464379&sr=8-1&keywords=Data-Intensive+Text+Processing+with+MapReduce



Study MapReduce Patterns



MapReduce Design Patterns
Donald Miner (Author), Adam Shook (Author)
O'Reilly Media (November 22, 2012)

Wrap-Up

Summary

- **We learned how to**
  - Decompose Problems into MapReduce Workflow
  - Utilize JobControl to implement MapReduce workflow

Questions?