Map Reduce Workflows

Also see the customized Hadoop training courses (onsite or at public venues) – [http://courses.coreservlets.com/hadoop-training.html](http://courses.coreservlets.com/hadoop-training.html)

Customized Java EE Training: [http://courses.coreservlets.com/](http://courses.coreservlets.com/)
Hadoop, Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Android.
Developed and taught by well-known author and developer. At public venues or onsite at your location.

For live customized Hadoop training (including prep for the Cloudera certification exam), please email info@coreservlets.com

Taught by recognized Hadoop expert who spoke on Hadoop several times at JavaOne, and who uses Hadoop daily in real-world apps. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - JSF 2.2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 7 or 8 programming, custom mix of topics
  - Courses available in any state or country. Maryland/DC area companies can also choose afternoon/evening courses.
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, GWT, Hadoop, HTML5, RESTful Web Services
  - Contact info@coreservlets.com for details
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

Count Each Letter
MapReduce

A 101
B 20
C 300
....
....
Y 3
Z 1

Produce start letter that has the most occurrences

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

MapReduce Workflows

- Your choices can depend on complexity of workflows
  - Linear chain or simple set of dependent Jobs vs. Directed Acyclic Graph (DAG)

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

```java
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("Workflow-Example");

            ...
            ...
        }
        ...
    }
}
```

Manage intermediate output directory

JobControl manages workflow

2: For Each Job in the Workflow

Construct ControlledJob

```java
...
...
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
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

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(MostSeenStartLetterReducer.class);
    job.setReducerClass(MostSeenStartLetterReducer.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

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

4: Execute JobControl in a Thread

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

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

... 
... 
```
5: Wait for JobControl to Complete

```java
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");
}
```

Report results or Display errors

Simply wait for all jobs to complete

Run JobControl Example

```bash
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

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
Study MapReduce Patterns

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

Wrap-Up

Customized Java EE Training: http://courses.coreservlets.com/
Hadoop, Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Android.
Developed and taught by well-known author and developer. At public venues or onsite at your location.
Summary

• We learned how to
  – Decompose Problems into MapReduce Workflow
  – Utilize JobControl to implement MapReduce workflow

Questions?

More info:
- http://courses.coreservlets.com/hadoop-tutorial – Hadoop training courses, at public venues or onsite at your organization
- http://courses.coreservlets.com/JSP/Tutorial/jsp/ – JSP 2.2 tutorial
- http://courses.coreservlets.com/java-8-tutorial/ – Java 8 tutorial
- http://www.coreservlets.com/ – JSF 2, PrimeFaces, Java 7 or 8, Ajax, jQuery, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training

Customized Java EE Training: http://courses.coreservlets.com/
Hadoop, Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Android. Developed and taught by well-known author and developer. At public venues or onsite at your location.

© 2012 coreservlets.com and Dima May