Apache Hadoop –

A course for undergraduates

Homework Labs, Lecture 5

Lab: Testing with LocalJobRunner

Files and Directories Used in this Exercise

Eclipse project: `toolrunner`

Test data (local):
`~/training_materials/developer/data/shakespeare`

Exercise directory: `~/workspace/toolrunner`

In this lab, you will practice running a job locally for debugging and testing purposes.

In the "Using ToolRunner and Passing Parameters" lab, you modified the Average Word Length program to use ToolRunner. This makes it simple to set job configuration properties on the command line.

Run the Average Word Length program using LocalJobRunner on the command line

1. Run the Average Word Length program again. Specify `-jt=local` to run the job locally instead of submitting to the cluster, and `-fs=file:///` to use the local file system instead of HDFS. Your input and output files should refer to local files rather than HDFS files.

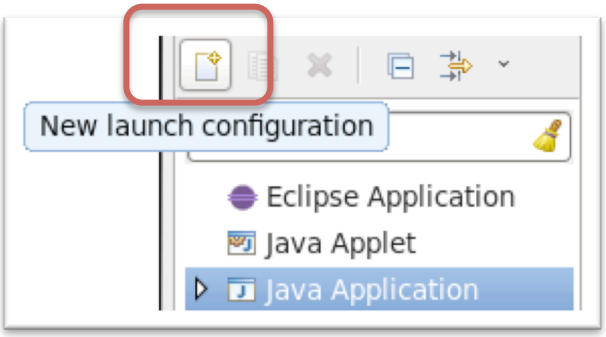
   Note: Use the program you completed in the ToolRunner lab.

```
$ hadoop jar toolrunner.jar stubs.AvgWordLength \
  -fs=file:/// -jt=local \
  ~/training_materials/developer/data/shakespeare \
  localout
```

2. Review the job output in the local output folder you specified.

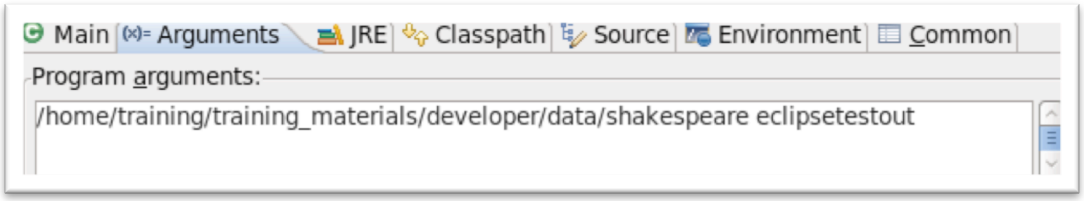
Optional: Run the Average Word Length program using LocalJobRunner in Eclipse

1. In Eclipse, locate the toolrunner project in the Package Explorer. Open the `stubs` package.
2. Right click on the driver class (`AvgWordLength`) and select **Run As > Run Configurations…**
3. Ensure that **Java Application** is selected in the run types listed in the left pane.
4. In the Run Configuration dialog, click the **New launch configuration** button:

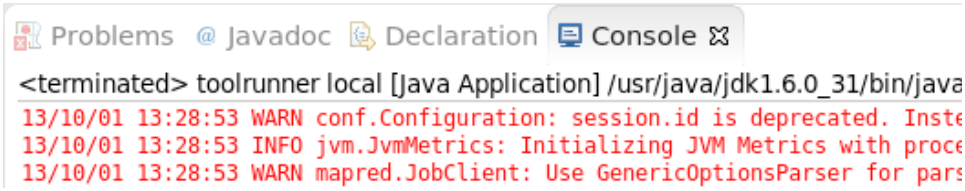


5. On the Main tab, confirm that the Project and Main class are set correctly for your project, e.g. Project: toolrunner and Main class: stubs.AvgWordLength

6. Select the Arguments tab and enter the input and output folders. (These are local, not HDFS folders, and are relative to the run configuration's working folder, which by default is the project folder in the Eclipse workspace: e.g. `~/workspace/toolrunner`.)

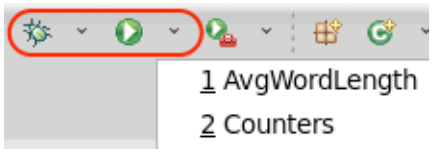


7. Click the **Run** button. The program will run locally with the output displayed in the Eclipse console window.



8. Review the job output in the local output folder you specified.

Note: You can re-run any previous configurations using the Run or Debug history buttons on the Eclipse tool bar.



This is the end of the lab.

Lab: Logging

Files and Directories Used in this Exercise

Eclipse project: `logging`

Java files:

`AverageReducer.java` (Reducer from ToolRunner)

`LetterMapper.java` (Mapper from ToolRunner)

`AvgWordLength.java` (driver from ToolRunner)

Test data (HDFS):

`shakespeare`

Exercise directory: `~/workspace/logging`

In this lab, you will practice using log4j with MapReduce.

Modify the Average Word Length program you built in the *Using ToolRunner and Passing Parameters* lab so that the Mapper logs a debug message indicating whether it is comparing with or without case sensitivity.

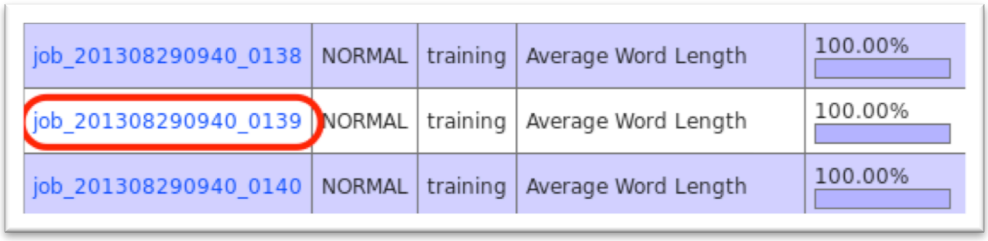
Enable Mapper Logging for the Job

1. Before adding additional logging messages, try re-running the toolrunner lab solution with Mapper debug logging enabled by adding `-Dmapred.map.child.log.level=DEBUG` to the command line. E.g.

```
$ hadoop jar toolrunner.jar stubs.AvgWordLength \
-Dmapred.map.child.log.level=DEBUG shakespeare outdir
```

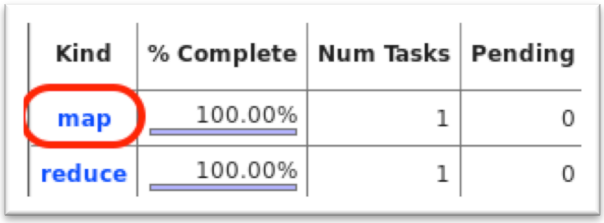
2. Take note of the Job ID in the terminal window or by using the `maprep job` command.

3. When the job is complete, view the logs. In a browser on your VM, visit the Job Tracker UI: `http://localhost:50030/jobtracker.jsp`. Find the job you just ran in the Completed Jobs list and click its Job ID. E.g.:



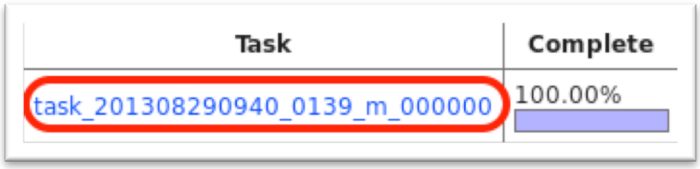
job_201308290940_0138	NORMAL	training	Average Word Length	100.00%
job_201308290940_0139	NORMAL	training	Average Word Length	100.00%
job_201308290940_0140	NORMAL	training	Average Word Length	100.00%

4. In the task summary, click map to view the map tasks.



Kind	% Complete	Num Tasks	Pending
map	100.00%	1	0
reduce	100.00%	1	0

5. In the list of tasks, click on the map task to view the details of that task.



Task	Complete
task_201308290940_0139_m_000000	100.00%

6. Under Task Logs, click “All”. The logs should include both INFO and DEBUG messages. E.g.:

Add Debug Logging Output to the Mapper

7. Copy the code from the `toolrunner` project to the `logging` project `stubs` package. (Use your solution from the ToolRunner lab.)

8. Use log4j to output a debug log message indicating whether the Mapper is doing case sensitive or insensitive mapping.

Build and Test Your Code

9. Following the earlier steps, test your code with Mapper debug logging enabled. View the map task logs in the Job Tracker UI to confirm that your message is included in the log. (Hint: Search for LetterMapper in the page to find your message.)

10. Optional: Try running map logging set to INFO (the default) or WARN instead of DEBUG and compare the log output.

This is the end of the lab.

Lab: Using Counters and a Map-Only Job

Files and Directories Used in this Exercise

Eclipse project: `counters`

Java files:

`ImageCounter.java` (driver)

`ImageCounterMapper.java` (Mapper)

Test data (HDFS):

`weblog` (full web server access log)

`testlog` (partial data set for testing)

Exercise directory: `~/workspace/counters`

In this lab you will create a Map-only MapReduce job.

Your application will process a web server's access log to count the number of times gifs, jpegs, and other resources have been retrieved. Your job will report three figures: number of gif requests, number of jpeg requests, and number of other requests.

Hints

1. You should use a Map-only MapReduce job, by setting the number of Reducers to 0 in the driver code.

2. For input data, use the Web access log file that you uploaded to the HDFS `/user/training/weblog` directory in the "Using HDFS" lab.

   Note: Test your code against the smaller version of the access log in the `/user/training/testlog` directory before you run your code against the full log in the `/user/training/weblog` directory.

3. Use a counter group such as `ImageCounter`, with names `gif`, `jpeg` and `other`.

4. In your driver code, retrieve the values of the counters after the job has completed and report them using `System.out.println`.

5. The output folder on HDFS will contain Mapper output files which are empty, because the Mappers did not write any data.

This is the end of the lab.