CprE 419 Lab 7: Pipelined Data Processing Using Spark

Purpose

In this lab, the goal is to introduce you Apache Spark, a fast and general engine for big data processing. Spark provides a basic data abstraction called RDD (Resilient Distributed Dataset) which is a collection of elements, stored in a distributed manner across a cluster. Using RDD transformations, Spark is well suited for data processing in pipelines. Spark also provides rich APIs for RDDs so that users can easily operate data in parallel. In addition, users can optionally persist RDDs in memory so that it will speed up computing when reuse the data.

During this lab, you will learn:

- The Spark platform and usage of its API's (in Java)
- Write program with pipelined jobs to analyze network logs

Submission

Create a single zip archive, named by your last name, with the following and hand it in through canvas**:**

- The output file for each task generated by your program.
- Commented Code for your program. Include all source files needed for compilation.

Examples

We have 2 examples "WordCount" and "StockPrice", that contain some Spark API usages. We have already seen "WordCount" in Hadoop and Pig, the problem is to count the number of occurrences for each distinct word. In Spark, we first use **flatMap()** method to split each line of text into words and each word is as one element in the RDD. Then we use **mapToPair()** method to transform RDD into PairRDD, that converts each element into <key, value> pair where key is the word and value is one. Finally, we use **reduceByKey()** method to sum all the ones to get the number of counts for each word. Note that the function in **reduceByKey()** method is applied in associative manner, like the Combiner in Hadoop. We also provide another example "StockPrice" in the lecture which analyze the stock prices.

For all the API's on RDD and PairRDD, check the links to their javadocs:
https://spark.apache.org/docs/1.6.0/api/java/org/apache/spark/api/java/JavaRDDLike.html

https://spark.apache.org/docs/1.6.0/api/java/org/apache/spark/api/java/JavaPairRDD.html

For more examples in Java:
https://github.com/apache/spark/tree/master/examples/src/main/java/org/apache/spark/examples

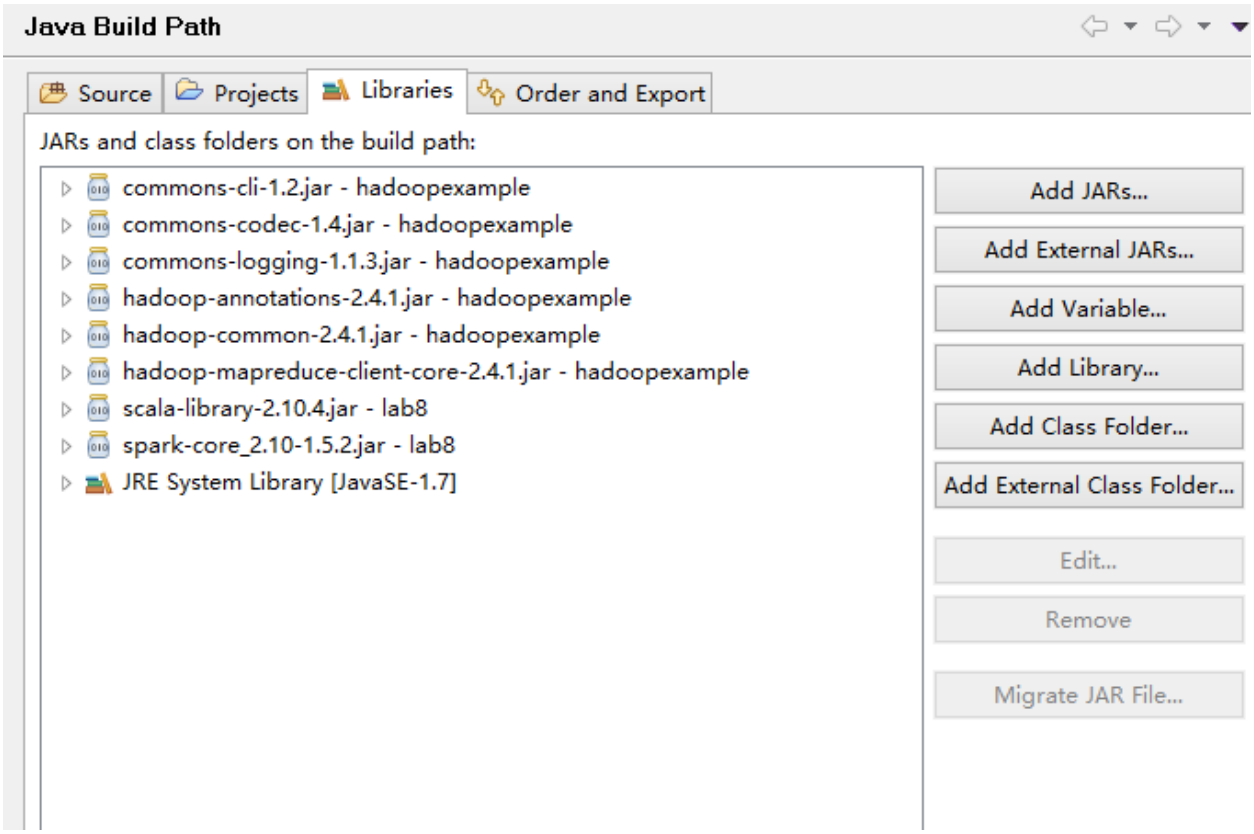
Compile and Submit Application

To compile your Java program, you will need to link Spark libraries and also Hadoop libraries to include them in your class path. The **recommended option** to add the libraries to your program is using Maven. Add the following dependency for Spark library into the pom.xml file in your maven project:

```
<!-- https://mvnrepository.com/artifact/org.apache.spark/spark-core -->
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.11</artifactId>
    <version>1.6.0</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-common -->
<dependency>
     <groupId>org.apache.hadoop</groupId>
     <artifactId>hadoop-common</artifactId>
     <version>2.6.0</version>
     <scope>provided</scope>
</dependency>
```

If you wish to manually add the libraries, find the essential jar files on Canvas, zipped in file named **sparkjars.zip** and **hadoopjars.zip**. Here is the list of all the jar files manually added to a project in Eclipse IDE:



You can submit your job by the following command (in Hadoop cluster machine, after you logged in)

```
spark-submit --class <Class-Name> <Application-Jar> [Application-Arguments]
```

Now you can view the content in the output_file.txt to see the progress of your jobs. The output can be checked via HDFS.

Viewing Job Status in WebUI

Spark also provides a Web UI to monitor submitted jobs, the address is:
http://hpc-class.its.iastate.edu:8088

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20160328000320-0005	IP Trace	160	1024.0 MB	2016/03/28 00:03:20		FINISHED	5.5 min
app-20160327234257-0004	IP Trace	160	1024.0 MB	2016/03/27 23:42:57		FINISHED	8.9 min
app-20160327233438-0003	WordCountSort in Spark	160	1024.0 MB	2016/03/27 23:34:38		FINISHED	2.0 min
app-20160327232753-0002	WordCountSort in Spark	160	1024.0 MB	2016/03/27 23:27:53		FINISHED	1.0 min
app-20160327232135-0001	WordCount in Spark	160	1024.0 MB	2016/03/27 23:21:35		FINISHED	34 s
app-20160327231345-0000	WordCount in Spark	160	1024.0 MB	2016/03/27 23:13:45		FINISHED	11 s

Experiment 1 (40 points)

In this experiment we will modify the word count example, so that the output is sorted by the number of counts in descending order. We use the Gutenberg corpus, as the testing input to your program. It is on the HDFS at the following location: /cpre419/gutenberg. Include the source code and the snapshot of first 10 lines of your output file in your submission.

Hint: To achieve sorting, you can use the **sortByKey()** method. However, key is the word but we want to sort by the counts. You can use the **mapToPair()** method to swap the key and value. You can use Shakespeare corpus to test your program first because it is a small data.

Experiment 2 (60 points)

We will redo the firewall example in lab 6 on pig, here we will write pipelined jobs in Spark.

Given two input files:
/cpre419/ip_trace – An IP trace file having information about connections received from different source IP addresses, along with a connection ID and time.
The format of IP trace file is:
<Time> <Connection ID> <Source IP> ">" <Destination IP> <protocol> <protocol dependent data>

/cpre419/raw_block - A file containing the connection IDs that were blocked
The format of block file is:
<Connection ID> <Action Taken>

Your task is to regenerate the log file by combining information from others logs that are available. The lost firewall log should contain details of all blocked connections and should be in the following format.

<Time> <Connection ID> <Source IP> <Destination IP> "Blocked"

A. (30pts) Regenerate the firewall file containing details of all blocked connections. You only need to submit your source code and the snapshot of first 10 lines your generated firewall log.

B. (30pts) Based on the previous program, generate a list of all unique source IP addresses that were blocked and the number of times that they were blocked. This list should be sorted (by the script) by the number of times that each IP was blocked in descending order. Submit your code and the snapshot of first 10 lines of your output file.

You can write part A and part B in the same program.