CPrE/SE 419: SOFTWARE TOOLS FOR LARGE-SCALE DATA ANALYSIS, SPRING 2024

Purpose

The goal of this lab is to introduce you to 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 have the RDDs persist in memory, which will speed up computing when reusing the data from partial computations.

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 with the following and hand it in through canvas:

- Write-up report with screenshots of each experiments
- Commented Code for your program. **ONLY** give the java files **AND** jar files. Please do not resubmit the data set or the output files.

Examples

The template code, "WordCountSpark.java", contains 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.

For all the API's on RDD and PairRDD, check the links to their javadocs:

https://spark.apache.org/docs/latest/api/java/index.html

https://spark.apache.org/docs/latest/api/java/org/apache/spark/api/java/JavaRDDLike.html

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

org.apache.spark.api.java.function (Spark 3.5.1 JavaDoc)

Compile Program

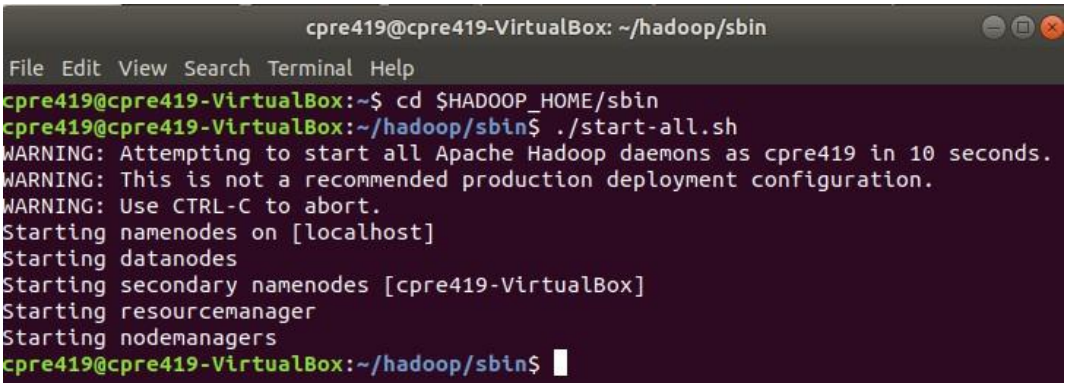
To compile your Java program, you will need to link Spark libraries and also Hadoop libraries to include them in your class path. The **recommended option** to add the libraries to your program is using Maven. The following dependencies for Spark library are needed to be added into the pom.xml file in maven project, and you can find a working template, "pom_spark.xml", under "~/Downloads" folder in VM instance:

```
<dependencies>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_2.11</artifactId>
            <version>2.4.5</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-common</artifactId>
            <version>3.1.3</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>jdk.tools</groupId>
            <artifactId>jdk.tools</artifactId>
            <version>1.8.0_242</version>
            <scope>system</scope>
            <systemPath>/usr/lib/jvm/java-8-openjdk-amd64/lib/tools.jar</systemPath>
        </dependency>
</dependencies>


<properties>
            <maven.compiler.source>1.8</maven.compiler.source>
            <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

Execute Spark Program

1. Start Virtual Machine, and start all

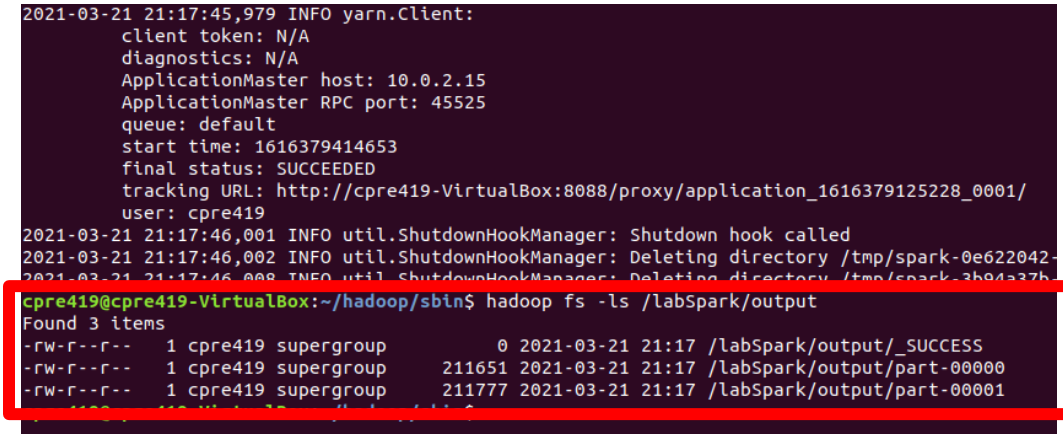


2. Create Maven project (ref: check Lab 2 document). Ensure JRE System Library is [java-8-openjdk-amd64] and also modify pom.xml as shown above. Then, finish your own code (Note: Eclipse is pre-installed in VM, and you can finish your code directly in VM)
3. Export your .jar file and submit to Spark.
   There is already an executable example .jar file under "~/Downloads" folder, run it by

spark-submit --class <class_name> <path_to_jar_file> <optional argument(s) depends on your code, such as input and output>

e.g. spark-submit --class WordCount ~/Downloads/WordCount_spark.jar /data/shakespeare /labSpark/output

Waiting for a while, after "final status: SUCCEEDED", you can check output



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 Shakespeare dataset, as the testing input to your program. The dataset can be found in Cybox, https://iastate.box.com/s/pots65nagg9qjbsnnsmqjws58ap8j3ng . 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.

Experiment 2 (60 points)

We will redo the firewall example in last Pig lab, except here we will write pipelined jobs in Spark.

Given two input files:

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>

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 need to finish part A and part B in the same program.