CS 419/519 Document Analysis
Lab and Assignment 4: Parallel Document Analysis with Spark

November 22, 2015

This lab and assignment give you a chance to use MapReduce to analyze centrality of users in social network. Through this assignment you will have better understanding of MapReduce programming as well as different centrality measures in the graph.

- Maximum marks: 10
- Programing language: Java or Python
- Assignment questions: CS419/519 Canvas Site
- Deadline: See course webpage

Marking scheme and requirements: Full marks will be given for (1) working, readable, reasonably efficient, documented code that achieves the assignment goals and (2) for providing appropriate answers to Q1-Q3 in a file `ANSWERS_lastname.pdf` submitted with your solution.

Please adhere to the collaboration policy on the course website – people you discussed the assignment solution with, or websites with source code you used should be listed in `ANSWERS_lastname.pdf`.

To verify that your code runs end-to-end, you may be asked to make some changes to your search engine during the evaluation lab and then re-evaluate results.

What/how to submit your work: Please submit `ANSWERS_lastname.pdf` and all your source code, zipped into a single file called `Assignment4_lastname.zip` and submit to **Canvas**. **Please do not submit data or other large binary files — Canvas will not accept large submissions.**

1 The Twitter Data

We will work on the data of tweets collected in one month of 2009. The data is compressed to have size about 1G, and is uploaded to Canvas. We pretend it is hard to process the data without parallel computing.

The data is stored in text files. Each tweet is a text block of 3 lines: time, user and text of the tweet. Blocks are separated by blank lines. Here is a snippet:

```
T 2009-06-11 16:59:45
U http://twitter.com/ibbored
W amberback #squarespace does?  Hot damn.  Now I want to win more.
```

Twitter users abbreviate topics with hash-tags (#squarespace) and mention others with @ sign (answerback). In this assignment, you will need to extract records like user names and hash-tags from the text, and you can use the code in the example (see Section 4).

2 Preparation for the Introductory Lab and Assignment

In the lab you'll familiarize yourself with Apache Spark. In this lab, we only focus on Spark MapReduce that run both locally and on EC2. To save our precious lab time, please set the Spark environment before the lab. All the following instructions are tested on Linux only. It should be similar for Mac and Windows. You can use either Java or Python in the lab session and your work on assignment.

Set up Spark: To install Spark, you only need to download the binary package from this link `http://spark.apache.org/downloads.html` (please select version 1.4.1 pre-built for Hadoop 2.6 or later) and unzip the package to your directory. As an option, you add the the two subdirectories `bin` and `ec2` in the environment variable `PATH` so the system can find Spark commands.

Run the Word Count example: Read the documentation "Submitting Applications" on this page:

`https://spark.apache.org/docs/1.4.1/submitting-applications.html`

If your set up is correct, you should be able to submit the Word Count example locally with the following command in the Spark directory. For java:

```
./bin/spark-submit --class org.apache.spark.examples.JavaWordCount \
                     lib/spark-examples-1.4.1-hadoop2.6.0.jar text_file
```

For Python

```
./bin/spark-submit examples/src/main/python/wordcount.py text_file
```

If successful, you should be able to see the program outputs word counts of the text file.

Set up EC2: Read the documentation "Running Spark on EC2" on this page:

`https://spark.apache.org/docs/1.4.1/ec2-scripts.html`

Finish the at least two steps in the section "Before you start". You will get a `.pem` file and a `.csv` file that contains your `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. Keep the two files at a safe place, and they will be used later. WARNING: You need to have an AWS account. If you do not have one, apply one as early as possible, since Amazon need some time to approve your application.

3 In the Introductory lab

In this introductory lab, we will run a MapReduce program to do some counting on the Twitter data.

Three data files are put on Canvas for your use. `real_tweet_data.txt.gz` is the real data file that contains tweets of one month. `small_data_sample.txt` and `tiny_data_sample.txt` are two small samples from the original data. You can debug you code on the tiny and small data files and then run on real data.

3.1 Run the example locally

In the data, some tweets have text "No Post Title" only, and these tweets are considered "bad records" here. The handout code `a4example` provides an example of counting bad records in the data. It is written in both Java (`BadRecordCount.java`) and Python (`bad_record_count.py`).

With the Java job, you need to compile the code first. When submitting the job, the parameter `--class` should be set to `a4example.BadRecordCount`.

The Python code can be directly submitted.

The submission commands for Java and Python are as follows

```
spark-submit --master local --class a4example.BadRecordCount jarfile.jar data_file
spark-submit --master local bad_record_count.py data_file
```

3.2 Run the example on EC2

After you successfully run the example on local machine, then you can try to run it on EC2 with the four steps below. You can find more details in the documentation "Running Spark on EC2".

Modify the code: You need to modify the code so all the EC2 cluster nodes can read the data. In this example, we put our data on `s3` bucket. Go to `s3` service, create a bucket, and upload the data there. In the handout code, the bucket name is `a4bucket`. To run this code, you need to change the `s3` path, the access key, and the secret access key to yours. Note that there are other places to store input files for EC2 cluster, such as AWS EBS and the HDFS of EC2 cluster.

Start the cluster: First set the access key and secret key as global environment.

```
export AWS_ACCESS_KEY_ID=ABCDEFG1234567890123
export AWS_SECRET_ACCESS_KEY=AaBbCcDdEeFGgHhIiJjKkLlMmNnOoPpQqRrSsTtU
```

Then you can start the cluster with the EC2 script.

```
./ec2/spark-ec2 --key-pair=awskey --identity-file=awskey.pem --region=us-west-2
--instance-type=t2.micro --hadoop-major-version=2 launch my-spark-cluster
```

The argument `--key-pair` specifies the name of the key pair. The argument `--identity-file` is the `.pem` file generated when you create the key pair. The argument `--region` is the region of cluster, where the region `us-west-2` is for Oregon. Instance type `t2.micro` can be used 750 instance-hours for free with a free tier account. WARNING: do not use `m1.large`, which is not free, in the Spark documentation. The option `--hadoop-major-version=2` specifies Hadoop version 2 to avoid a bug of file reading in Hadoop 1.

The cluster takes a while to start. In the last few lines of the log, you can find the master address.

Submit your job: After the cluster starts, you can login the master to submit your job. The submission is the similar to the local submission. The Spark installation is at `~/spark`. The parameter `--master` should be set to `spark://master-address:7077`. You should provide the cluster with an `s3` file, whose path is in this format: `s3n://your_bucket/data_file` .

The example shows two ways of writing the result. The first way is to collect the RDD data to a local variable on the master node and then let the master write the data to a local file. The second way is to write the RDD data directly to an HDFS file. This method should be used when the result is too large to collect. The HDFS file can be copied out with this command:

```
ephemeral-hdfs/bin/hadoop fs -getmerge /result_hdfs result_hdfs.txt
```

You are required to use the second method in your assignment to write your result.

After the job is done, you can copy the result file from the cluster to your local machine with command: `scp -i key_pair.pem`.

Destroy the cluster: Before destroy the cluster, make sure you have copied the result back. Then you can destroy the cluster with the following command.

```
./ec2/spark-ec2 --region=us-west-2 destroy my-spark-cluster
```

You should not leave a cluster on after the job is done, otherwise hours are still counted and possibly charged by AWS. Please check your bill periodically in case of unnoticed changes.

Programming tip: 1. When debugging, be aware of multiple execution threads. For example, if each mapper prints a debug message, then the order of these debug messages is not fixed. 2. When debugging, you can sample some data from the RDD variable and check whether it contains key-value pairs you expect.

4 Main Assignment (Assessed in Evaluation Meeting)

Q1. Answer the following questions (2 pts)

(a) Let `rdd1` be an RDD object, and `rdd3` is obtained by the following code:

```
rdd3 = rdd1.map(mapFunc).flatMap(flatMapFunc);
```

The function `mapFunc` is defined as

```
ElemType2 mapFunc(ElemType1 input) { do some work; return output;}
```

And the function `flatMapFunc` is defined as

```
List<ElemType3> flatMapFunc(ElemType2 input)
{
  list = [ ];
  for (i = 1 :  someLoopNum) {
    list = list.add(elem);
  }
  return list;
}
```

Is it possible to define a function `combinedMapFunc` such that the same `rdd3` can be obtained by one map step? If yes, write out the pseudo code of `combinedMapFunc` and give the line of obtaining `rdd3` with `combinedMapFunc`. Otherwise, explain why.

(b) In a Spark program, an RDD object `rdd1` calls one of these three member functions, `map`, `flatMap`, or `reduce`, and returns an RDD object `rdd2` that has less elements than `rdd1`. Which function(s) is/are possibly called? Explain why.

Q2. Build mention graph from users of a hash-tag (3pts)

The mention graph is the mention relations between users. In this graph, each user is viewed as a node. If a user `Alice` mentions another user `Bob` in her tweet(s) with the @ sign, then an undirected edge connects `Alice` and `Bob`. The edge weight is the number of mentions in the tweets.

Now we only consider tweets that contain hash-tag #turncoat. Please find all tweets that contains hash-tag #turncoat and build mention graph from these tweets with Spark MapReduce. Three functions are provided in `Util.java/util.py` for your reference. When you extract an edge from the data, you'd better dump out the corresponding tweet text, which will be used in the next question.

a provide your mapper code

b provide your reducer code

c provide top 5 edges with highest weights

Q3. Calculate centrality of users (5 pts)

In this question, your need to analyze centrality of users in the mention graph. A Python script is provided in file `centrality_plot.py` for calculating centrality and plotting graph. The input of this script is a `.csv` file that contains all edges of the graph. Each row is an edge like this: `node1, node2, 2`. To run the code, you need to install the `networkx` package first. In the plot, the node size and edge width are proportional to the user centrality and mention weight respectively. An example file `edge_list_example` is provided, and you can try the code with command: `python centrality_plot edge_list_example` .

(a) Write the graph built from tweets that contains hash-tag #turncoat into file, and then run `centrality_plot.py` to calculate centrality of users and plot the graph. Provide the graph in PDF.

(b) Write MapReduce code to list hash-tag frequency. Provide frequency of top 10 hash-tags.

(c) Choose your own hash-tag, build mention graph from tweets mentioning that hash-tag, and plot the mention graph with user centrality.

(d) Look at tweet content, do you think betweenness centrality is a good measure of key discussants of the hash-tag? Why or why not?

(e) Choose two different centrality measure of users in hash-tag discussion. How different are users' centrality values with different measures? Which is better? Why? Here is a list of `networkx` functions that calculate different centrality measures:

    `http://networkx.lanl.gov/reference/algorithms.centrality.html`