18-645: How to Write Fast Code

Project #3 – Cloud Computing

Running MapReduce on Amazon Elastic Mapreduce (EMR)

Due: April 21st, 2017 (Friday), 11:59 pm EDT

The goal of this project is to use your understanding of parallel computing resources in cloud computing framework to extend two fully functional applications. The applications are NGramCount and HashtagSim.

Submission:

Please commit your code to git repository as you did in previous mini projects. Upload a zip file **to Canvas** named **18645_project3_teamXXX.zip** containing the following **3 items**:

1. The project report: 18645_project3_teamXXX.pdf
    - **Please include all team members' name and Andrew ID in the report**
2. The project jar file: 18645_project3_teamXXX.jar
3. A text file containing the AWS CLI command you used in starting the job: 18645_project3_teamXXX.txt
    - Don't worry about the S3 bucket access privilege, we will replace it to ours when evaluating your submitted work.

Application Description

1. NgramCount

An n-gram is a contiguous sequence of n items from a given sequence of text or speech (http://en.wikipedia.org/wiki/N-gram). An n-gram of size 1 is referred to as a "unigram"; size 2 is a "bigram" (or, less commonly, a "digram"); size 3 is a "trigram". Larger sizes are sometimes referred to by the value of n, e.g., "four-gram", "five-gram", and so on.

In this NgramCount application, given an input corpus, we're interested in the count of all the n-grams.

2. HashtagSim

This program analyzes the similarities between hashtags, which are used primarily in Twitter.com to label the tweets. A hashtag is denoted by a ‘#’ followed by a word. For example, a recent tweet from Barack Obama reads: “The excuses not to #ActOnClimate need to end”. In this tweet, “#ActOnClimate” is a hashtag.

Hashtags are used to categorize tweets, promote events, etc. In our program, we try to identify how similar the hashtags are. We’re using the words that co-occurred with a hashtag as its features. For example, given a tweet “#a b c”, word “b” and “c” will have both co-occurred with hashtag “#a” in the tweet for once. Given the following corpus:

```
#a b c
#a b #b
#b #c d e
#c e f
```

The co-occurrence counts for each hashtag, or put in another way, the features for the hashtags, will look like the following:

```
#a b:2;c:1;
#b b:1;d:1;e:1;
#c d:1;e:2;f:1;
```

Note that here we are treating hashtag “#b” and word “b” differently. Also, we’re using both the co-occurred words and the co-occurrence counts to represent a hashtag.

After calculating the feature vector for the hashtags, we can compute the similarity between hashtag pairs, using inner product of the feature vectors. The inner product of two feature vectors is calculated by picking all the common words and summing up the products of the co-occurrence value.

Taken the above result as example, the inner product between “#a” and “#b” will be 2, since they have only 1 shared word, “b”, and the product is 2*1. The inner product between “#b” and “#c” will be 3, which is result of 1*1 (d) + 1*2 (e). The bigger the inner product is, the more similar the two hashtags are in the given corpus.

Source Code Structure

Below is project structure:

```
$ tree
.
├── build.num
├── build.xml                                              ← Ant build file
├── lib
│   ├── commons-cli-1.2.jar
│   └── hadoop-core-1.0.3.jar -> /home/dpiao/hadoop-1.0.3/hadoop-core-1.0.3.jar
├── MANIFEST.MF
└── src
    └── mapred
        ├── filesystem
        │   └── CommonFileOperations.java
        ├── hashtagsim                                     ← HashtagSim program
        │   ├── Driver.java
        │   ├── HashtagMapper.java
        │   ├── HashtagReducer.java
        │   ├── JobMapper.java
        │   ├── JobReducer.java
        │   ├── SimilarityMapper.java
        │   └── Tokenizer.java
        ├── job
        │   └── Optimizedjob.java
        ├── main
        │   └── Entry.java                  ← Hadoop Jar entry point, do NOT change it!
        ├── ngramcount                                     ← NgramCount program
        │   ├── Driver.java
        │   ├── NgramCountMapper.java
        │   └── NgramCountReducer.java
        └── util
            ├── FileUtil.java
            ├── InputLines.java
            ├── Parser.java
            ├── SimpleParser.java
            └── Tokenizer.java
```

How to Run the Code?

Refer to Homework 3 instructions on how to build the project, running both programs on local machine, and running them on Amazon EMR cluster.

Project Goal

1. Extend NgramCount

As you might have discovered, the current NgramCount program only counts the unigrams. You will be extending the program to count n-grams, with “n” specified as an input parameter.

Modify the source files inside mapred.ngramcount package so that it takes “n” as a command line option, and outputs counts of corresponding n-grams. You are free to modify any code in the entire package (**except for Entry.java**).

This task is relatively simple - shouldn’t take more than a few lines to complete.

We’ll be testing with the following command:

```
$ hadoop jar 18645_project3_teamXXX.jar -program ngramcount -input testdata -output
testoutput -n N
```

2. Extend HashtagSim

As you might have discovered, the current HashtagSim program only computes similarities between “#job” and other hashtags. You will be extending the program so that it computes similarities between all pairs of hashtags that share at least 1 common word. In other words, you can ignore hashtags pairs that have similarity score 0, either in output, or in your computation. Input data will be the **tweets1m** dataset. **Write the top 5 hash tag pairs and their similarity scores in the report.** Again, you’re free to modify any source code in the project package (except for Entry.java).

We’ll be testing with the following command:

```
$ hadoop jar 18645_project3_teamXXX.jar -program hashtagsim -input testdata -output
testoutput -tmpdir tmp
```

3.Speed up HashtagSim

We will measure the runtime of your HashtagSim program, on a dedicated EMR cluster of **5 medium** machines (c1.medium, 1 master + 4 slaves, same as homework instructions).

Some background information on Hadoop: in Hadoop 0.23.0 a new MapReduce implementation was introduced. The new implementation (called MapReduce 2) is built on a system called YARN (Yet Another Resource Negotiator), compared to the classic MapReduce framework (also called MapReduce 1)

The MapReduce concepts and workflow you learned in this course can be well applied in both release tracks. For easy job configuration and tuning, **we will use the classic MapReduce framework (MapReduce 1) in this mini project by using ami-version 2.4.11.**

Below is an example script on how we will evaluate your submission:

```
aws emr create-cluster --name "Test cluster hashtagsim" --ami-version 2.4.11 \
--service-role EMR_DefaultRole --ec2-attributes InstanceProfile=EMR_EC2_DefaultRole \
--log-uri s3://bliu.log-uri.hashtagsim --enable-debugging \
--instance-groups
InstanceGroupType=MASTER,InstanceCount=1,InstanceType=c1.medium
InstanceGroupType=CORE,InstanceCount=4,InstanceType=c1.medium \
--steps Type=CUSTOM_JAR,Jar=s3://bliu.fastcode/18645_project3_teamXXX.jar,Args=["-
input","s3://bliu.tweets1m/tweets1m.txt","-output","s3://bliu.output/hashtag1m","-
program","hashtagsim","-tmpdir","tmp"] \
--auto-terminate
```

Remember to replace several parts of above command with appropriate URLs, etc.

For this mini project evaluation, hard requirements on EMR configuration include using **AMI version 2.4.11** and EC2 **instance type c1.medium (1 master node + 4 slave nodes)**. All other options under "aws emr create-cluster" are left free for you to explore.

The **passing time** of running the all pair HashtagSim program is **90 minutes**. You'll need to ensure the correctness of the program while trying to reduce the runtime. We'll be using the **tweets1m** dataset for testing the runtime. **Highlight your final run time in the report.**

FAQ

Q1. What is the baseline requirement for mini project 3 hashTagSim speedup?
A: Your program should be able to produce correct similarity scores (those higher than 0) for **every** hash tag pair. The running time should be within **90** minutes time.
For quick verification purpose, you are asked to write the running time and top 5 pairs with similarity scores in the report, but you need to produce the scores for all pairs.

Q2. Must the Hadoop program output file(s) be sorted?
A: No, you may choose to sort the file(s) offline. The output record order depends on your workflow design.

Q3. Is there any requirement on the order of output hashtag pair (i.e. #A #B or #B #A)?
A: No, it does not matter. However, you should NOT have both #A #B and #B #A in the program output. Also, it does not matter whether the similarity score is before or after the hashtag pairs. Either [Score #A #B] or [#A #B Score] is fine.

Q4. How the top 2 teams will be selected?
A: Base on the running time mentioned in Q1, i.e. the running time that is used in producing correct similarity scores for **all** hash tag pairs.
Hint: Here is the place where we challenge your techniques on Hadoop and MapReduce program optimization.

Q5. Can I speed up the program with some compromise on the accuracy (e.g. compromise on exact similarity scores, or even the top pairs)?
A: Yes, BUT only after meeting baseline requirement in Q1. This serves as an extension of the project task. You are encouraged to think further on how to balance computation complexity and accuracy in real world problems. You may get up to 5 bonus points if those further speed up techniques are reasonable and well explained in the report (full project score is still 100, unless you are in the top 2 teams). **Note that running time under such condition will NOT be considered in the process of selecting the top 2 teams.**

Grading criteria

1. 30% - Correctness - Correctness of the results for NgramCount (10 points) and HashtagSim (20 points). **Commit and push your code on GHC cluster machine**. Please do NOT commit the data folder!
2. 30% - Performance – Finish the extended HashtagSim program, for **"tweets1m"** data set, in a cluster of 5 medium instances (c1.medium) on Amazon EMR, in **90 mins. Highlight your run time in the report.** You get 10 points for passing this baseline.
3. 30% - Write up
    a. NgramCount
    b. HashTagSim - for each optimization technique, clearly describing:
        - What is this technique
        - What is the expected speed up?
        - What is the observed speed up?
        - An explanation of any difference between the expected and observed speed ups
4. 10% - Code quality - Good coding practices and well commented code

Two best teams that have fastest run time in extended hashTagSim program will be given bonus points and be asked to do a 10 - 15 mins presentation each on what they have explored.

P.S. For people who are interested in YARN (MapReduce 2), the fundamental idea of YARN is to split up the two major functionalities of the JobTracker, resource management and job scheduling/monitoring, into separate daemons. It maintains API compatibility with stable release of hadoop-1.x. Here is a good introduction of YARN if you want to read more : http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html