18-645: How to write fast code

Project #3 – Cloud Computing

Due: March 26th, 2015 (Thursday), 11:00pm EDT, 8:00pm PDT

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: each student to submit a copy of the team report to Acatar project page with file name:
"**18645_project3_teamXXX_andrewID.pdf**".

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 + 1*2. 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!
        ├── 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-proj3-0.1-latest.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.

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-proj3-0.1-latest.jar -program hashtagsim -input testdata -output testoutput -tmpdir tmp
```

3. Speed up HashtagSim

We'll be measuring 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). Here we make a slight update on the EMR cluster starting script.

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 techniques 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**.

To apply classic MapReduce (MapReduce 1) framework in EMR, we will need to **update the "--ami-version" value in the cluster starting script from 3.3.2** (as used in HW3) **to 2.4.11** (released in Feb 2015). 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 \
--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-proj3-0.1-latest.jar,Args=["-input","s3://bliu.tweets1m/tweets1m.txt","-output","s3://bliu.output/hashtag1m","-program","hashtagsim","-tmpdir","tmp"] \
--auto-terminate
```

This "--ami-version" is the only change you need to make from the EMR starting scripts in HW3. For this mini project, 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 shorten the runtime. We'll be using the **tweets1m** dataset for testing the runtime.

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

Grading criteria

- 30% - Correctness - Correctness of the results for NgramCount and HashtagSim. **Commit and push your code on GHC cluster machine**. Please DON'T add the data folder!
- 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. Record your run time in the report.**
- 30% - Write up – Clearly describing, for each performance optimization,
    - how the speed up works
    - what is the expected speed up
    - what is the observed speed up
    - an explanation of any difference between the expected and observed speed ups
- 10% - Code quality - Good coding practices and well commented code

Guidelines for the write up:

Minimum of one 8.5x11 page write-up for each optimization. The write up should include:

- Optimization goal:
    - Hardware resources being optimized toward?
    - What is the specification of the hardware you are optimizing for?
- Optimization process:
    - Data considerations
    - Parallelization considerations
- Optimization results:
    - Performance before optimization
    - Performance after optimization

Two teams with the fastest project in the class will be asked to do a 10min presentation each on what they tried.

We will look at the code of the slowest two implementations as a class. The class will discuss why their code is slow.