CMSC 433 Project 5: ReduceOverflow

Due May 9th (Friday), 11:59:59pm

Goal: In this project you will build a tool which takes a code file and searches data from the website StackOverflow for posts relevant to your source using the Hadoop framework for parallel data processing. The idea is for this program to act as a development aid in a basic capacity by revealing common pitfalls along with how to address them. For this project, we will focus on Android-related posts. We divide processing the data into two passes:

- Matching the provided code against post data from StackOverflow and assigning a "relevance score" to each post. This is done in `CodeCompareMR`.
- Sorting the results to form a ranking of most to least relevant and also omitting low-ranking results. This is done in `ValueSortMR`.

Getting started:

1. Get Hadoop from http://archive.apache.org/dist/hadoop/common/hadoop-0.20.2/hadoop-0.20.2.tar.gz.
2. Open the P5 skeleton in Eclipse and add the appropriate JARs by going to "Build Path" -> "Configure Build Path" and select the "Libraries" tab. Click "Add External JARs" and browse to the folder where you extracted hadoop-0.20.2. Include the following JAR files in your project:
   - hadoop-0.20.2-core.jar
   - From the \lib folder: all jars with the prefix commons-* and log4j-1.2.15.jar
3. Download and extract the archive of Android-tagged StackOverflow posts from the class webpage.

Inputs: The program takes three paths (relative to the project's base directory) as command line arguments:

- `source:`
  - This is a path to a code file. The words from this file will be used as tokens to compare against code from the StackOverflow posts.
  - For best results, it should be an Android Java file. You can use the sample we provided.
- `in:`
  - This is a path to the input directory with the StackOverflow archive data. The raw data will feed directly into Hadoop for the matching pass of map-reduce.
  - The data is separated into several CSV (comma-separated values) files. Within each CSV file, each line represents a post with the following format:
    `Id,PostTypeId,AcceptedAnswerId,ParentId,CreationDate,Score,ViewCount,Body,OwnerUserId,OwnerDisplayName,LastEditorUserId,LastEditorDisplayName,LastEditDate,LastActivityDate,Title,Tags,AnswerCount,CommentCount,FavoriteCount,ClosedDate,CommunityOwnedDate`
- `out:`
  - This is a path to a directory where files will be written at the end of map-reduce. This directory should not exist when the program is run, or Hadoop will throw an error.

Classes: There are three classes provided in the skeleton. The sections of these classes which require implementing code will be marked with `//TODO: IMPLEMENT CODE HERE`. See the Javadocs in each class for more detail.

- `Main.java`:
  This is the entry point for the program. The `TEMP_DIR` variable contains the location of the temporary directory, meant to store the output from `evaluate()` and the input for `sort()`.

    - `main(args)`
      You can change this method as necessary to test your code. We provide a base implementation for you which creates a `Configuration`, reads the parameter options, and then calls `evaluate()` and `sort()` with new `Jobs`.
- `CodeCompareMR.java`:
  This class is responsible for parsing the provided source code file into tokens and indexing it as a dictionary in `source`. It then performs map-reduce on the StackOverflow data and performs lookups on every token in every post to build a relevance score relating the post to the source. The output should be of the form `post_title, relevance` for every post. You may assume each post has a unique title.
    - `tokenizeSource(input)`
      This function has been implemented for you. It iterates over the source code file and builds `source`, a `ConcurrentHashMap` mapping valid tokens to counts of how many times they appear in the code.
    - `TokenizerMapper`
      This subclass extends `Mapper` and is responsible for parsing the StackOverflow CSVs and extracting their fields (namely the body and title). The parsing code has been provided for you. The `map()` function needs to be completed to emit the proper values.
    - `HashReducer`
      This subclass extends `Reducer` and is responsible for taking the data from `TokenizerMapper` and calculating relevancy counts for each post. This is done by looking up each token for each post and adding together the number of matching tokens from the provided source code.
    - `evaluate(job, source, input, output)`
      This function is responsible for executing the map-reduce process using `TokenizerMapper` and `HashReducer`. It has been partially implemented for you, leaving some details of the job configuration to be filled in.
- `ValueSortMR.java`:
  This class is responsible for taking a tab delimited list of `String`, `Integer` key-value pairs (like the output from `CodeCompareMR`) and performing a secondary-sort. Entries are sorted by their values, the `Integers`, **from greatest to least**, using map-reduce. Any pair with an `Integer` less than `CUTOFF` should be **omitted**. The output should still be `String`, `Integer` pairs, with only the sorting changed.
    - `SwapMapper`
      This subclass extends `Mapper` and is responsible for reading the input and mapping it such that the data is sorted properly. It works in conjunction with `SwapReducer`.
    - `SwapReducer`
      This subclass extends `Reducer` and is responsible for formatting the output correctly. It works in conjunction with `SwapMapper`.
    - `sort(job, source, input, output)`
      This function is responsible for executing the map-reduce process using `SwapMapper` and `SwapReducer`. It has been partially implemented for you, leaving some details of the job configuration to be filled in.

Running the program: You will need to provide the following command-line arguments to the program: `<code file> <data directory> <output directory>`. Before running, you must ensure that the output and temp directories do not exist/are deleted, as Hadoop will automatically generate these.

Testing: We will test your implementations of `CodeCompareMR` and `ValueSortMR` both independently and together by validating their output. There will **NOT** be public tests on the Submit Server. Sharing of tests for this project is **encouraged**.

Submission: Submit a .zip file containing your project files to the Submit Server. You **SHOULD NOT** include the temporary or output files, or any input data, with your submission.