Homework #6: Building a Distributed MapReduce Framework

Checkpoint due Thursday, December 1st
Due Thursday, Decembter 8th

In this assignment you will implement a distributed map/reduce framework. Map/reduce is a programming model for processing large data sets, typically implemented by distributing a computation across multiple worker servers where the data is stored. The computation is provided as a plug-in to the framework, and is specified as a separate map task and a reduce task. A client submits the two tasks to a master server, which is in charge of distributing the tasks across the worker servers in the system and returning the result back to the client when it is complete.

Your goals in this assignment are to:

- Gain a deep understanding of map/reduce and the challenges of building a distributed system by implementing a map/reduce framework.
- Learn how to use a map/reduce framework by implementing a simple map/reduce task that uses your framework.
- Practice network programming using `Sockets`.
- Practice parallel and concurrent programming in Java.

For this assignment we encourage you to discuss map/reduce and high-level details of your solution with the course staff and with your classmates, but you may not share your code with other students and must submit your own solution for the assignment.

This assignment includes a small checkpoint deadline (Thursday, December 1st), and is due Thursday, Decembter 8th. The usual course late policy applies for the overall deadline, but you may **not** use any late days for the checkpoint deadline.

Designing and implementing your map/reduce framework

Your framework must match the basic architecture described in lecture. Specifically, your framework should consist of three main components: a client, a master server, and multiple worker servers. The client simply submits the map and reduce tasks to the master server and waits for the master server to confirm that the computation is complete. The master server assigns the map task to worker servers that each execute the map task on some subset of a data set. The map task produces intermediate key-value results, which are partitioned by intermediate key via some configurable partitioning mechanism. In a *shuffle* phase, the intermediate results are redistributed among worker servers such that for each intermediate key, all of that key's key-value pairs are sent to a single worker server. When the shuffle phase is complete, each worker sorts its intermediate key-value pairs (by key) and executes the reduce task once per key, on the key and all values for that key. The reduce task produces final results of the computation. For more information we encourage you to see Google's original MapReduce paper.

To achieve efficient operation, map/reduce is typically coupled to an implementation of a distributed storage system that stores the source data and the results of the map/reduce computation. For this assignment we do not provide a distributed storage system and you do not need to implement one, but your solution should assume that the data is partitioned such that different worker servers can access different (replicated) data. See Piazza for an example of how you might partition data as if stored in a distributed storage system.

Your framework should be minimally robust to worker server failures, but not necessarily to master server or client failures. If a worker server crashes or otherwise fails to return a result, the master server should reassign the map or reduce tasks to other workers.

Starting this homework assignment

Before the checkpoint deadline (Thursday, December 1st) you must complete a small written assignment and also reach a milestone for the programming portion of the assignment; these tasks are described below. Please label your work for this milestone with a helpful commit message so we can easily find and evaluate it.

Your written task is to write pseudocode for a map/reduce job to find all *triangles* in a social network graph. A *triangle* in a graph is exactly what it sounds like: a subgraph consisting of three mutually-adjacent vertices. You may assume that the social network graph is represented like the social network graphs from the map/reduce lecture; the graph is a collection of text files, with a file for each person containing a sorted list of that person's friends. The output of your map/reduce task should be a single ordered triple, e.g. `(alice, bob, charlie)`, for each triangle in the graph.

Your pseudocode can be in any reasonable style; the high-level, Python-like pseudocode of the map/reduce examples from lecture is fine. Clearly label which task is your map task and which task is your reduce task, and turn in your work in `triangle.pdf` or `triangle.md` before the checkpoint deadline. Hint: There is a straightforward triangle-finding algorithm that requires multiple map/reduce iterations, but there is also a straightforward algorithm that requires just one map/reduce pass.

After you have completed the above task, you should begin work on your map/reduce framework. By the time of the checkpoint deadline your framework should be able to execute an arbitrary client-supplied map task on files accessible to the worker servers.

Completing your map/reduce framework

For the final portion of this homework, you must implement a map/reduce framework so that arbitrary map/reduce tasks can be executed on files accessible to the worker servers. When you are done, you should be able to run a master server, worker servers, and a client as separate Java applications. The client should submit a map task and reduce task to the master server, which should manage the computation, distributing tasks to the worker servers. See the evaluation section below for additional requirements for your solution.

Using your map/reduce framework

When you have completed your map/reduce implementation, you must use your map/reduce framework by executing several sample computations on your framework. These tasks are described below.

A sample word count computation

Start by implementing a sample map/reduce computation to count all occurrences of all words in a corpus of data, much like the sample computation from lecture.

In your Git repository we have provided a sample data set (partitioned into multiple files) you may use to test your clients and framework. For reference, the word "a" appears 9,976 times and "and" appears 16,299 times in that sample data.

Suggesting words based on word prefixes

Write a map/reduce client to help determine the best word completions for word prefixes. Specifically, your map/reduce task should analyze a corpus of data and output a *prefix/word* key/value pair for each prefix that appears in the corpus, where the word associated with a given prefix is the word that most frequently completes that prefix in the corpus.

For example, consider the prefix "a". Many words in a given corpus of data will have the prefix "a", since many words begin with the letter "a". For our corpus, the most common word beginning with "a" is the word "and", so the output of your map/reduce task output should include the key/value pair `a/and`. Similarly, the prefix "heav" could start one of several words—heavy, heaven, heavenly, etc. In our sample data, "heaven" appears the most frequently (87 times to be exact) of any other word that begins with the prefix "heav", so your map/reduce task output should include the key/value pair `heav/heaven`.

Use map/reduce on any interesting third problem

Use your map/reduce framework on any interesting third problem of your own choosing. You may use our sample data corpus or any other data corpus. Ideally, choose a data corpus and map/reduce computation that demonstrates the generality and power of your framework.

Evaluation

Your solution should meet the following minimal requirements:

- By the checkpoint deadline you must submit solutions for the tasks described in the section "Starting this homework assignment."
- Map workers must store their intermediate results in the file system in a human-readable format. Reduce workers must store their final results in the file system in a human-readable format, and the framework must return the location of those results to the client.
- You must use network sockets to implement all communication between the client and master, between the master and each worker, and between pairs of worker servers.
- Your framework should be designed to work with an arbitrary (but fixed) number of worker servers.
- Your framework should be robust to simple worker failures, and should compute the correct result as long as the full set of data is available on non-failed worker servers.
- Follow standard Java style guidelines and document your code with Javadocs where appropriate.

Overall, this homework is worth 100 points. You will be graded as follows:

- Checkpoint completion: 10 points.
- Working map/reduce framework using separate master/worker programs and network communication when deployed on multiple servers: 50 points.
- Robustness of map/reduce framework when some worker servers are unavailable but the full data set is available: 10 points.
- Correct implementation of map/reduce client computations: 20 points.
- Javadocs, style, and FindBugs: 10 points.