CS 5450: Networked and Distributed Computing
Lab 3
MapReduce
Spring 2024
Instructor: Professor Vitaly Shmatikov
TA: Tingwei Zhang, Aditya Manthri

Due: 11:59PM Tue, Apr 16 2024

1 Introduction

In this lab we ask you do develop a MapReduce project that performs word counting task. You need to develop a master server that can orchestrate a set of workers, assign different tasks, and accumulate the result.

Organizational details. The implementation again has to be done in C/C++. You can continue working in a group or switch a partner. You will have two new slip days for the remaining two homeworks.

1.1 Task description

Word count task: given a set of documents produce a key value output with the counts for each word. The final output file should contain the result (sorted by count) and use the following format:

key,count\n

...

You will be given few large files with the text. You can read the file line by line, ignoring all the non-latin symbols, and using lowercase for the rest. You should write your own map and reduce functions for this task.

MapReduce

Please refer to the class notes and the original MapReduce paper[1] for the details. Overall, you need to implement your own version of the system: a master node that orchestrates tasks on multiple worker nodes. Your submission code needs to support the following call:

./mapreduce --input <inputdir> --output <outputdir> --nworkers <nWorkers> --nreduce <nReduce>

This call starts a master node and $nWorkers$ processes as workers, runs MapReduce on files from *inputdir* (using $nReduce$ reduce tasks and a map task per each file), and writes all the result files including temporary into *outputdir*. There won't be more than 100 files in the *inputdir*, more than 4 workers, and more than 20 reduce tasks. All the workers will run on the same local machine with the master node and have access to the same file system. You are free to implement workers using RPC libraries, multithreading, or use communication over sockets. Please describe the details in your report.

Our version of MapReduce will contain the following stages:

1. map (runs on worker) – produce intermediate files by processing input files.
2. reduce (runs on worker) – aggregate values from the map stage and save the results.
3. merge (runs on master) – combine output from all reducers.

[1] `https://research.google/pubs/pub62/`

You don't need to start the next stage before the last one finishes. But consider that $nWorkers$ could be much smaller than number of files or *nReduce* tasks, therefore the master node needs to know of all the workers' statuses. As tasks become available, e.g. map tasks finish and reduce task can be performed, master node decides how to assign those tasks to workers. We are not going to test the failed scenarios, so you can assume that all tasks will finish eventually.

Each input file requires a separate mapping task. A worker that runs a map task reads the appropriate file, calls the map function on that file's contents, and writes the resulting key/value pairs to intermediate files for the reduce task, in the same format: "key,count\n". Importantly, your map can hash each key to pick the reduce task out of *nReduce* that will later process that key. You can pick your hash function that tries to split load equally between *nReduce* tasks (this is called partitioning phase).

There will be $nMap \times nReduce$ files after all map tasks are done, where $nMap$ equals number of files in the *inputdir*. Each file name contains a prefix, the map task number, and the reduce task number. If there are two map tasks and three reduce tasks, the map tasks will create these six intermediate files:

- map.part-0-0.txt
- map.part-0-1.txt
- map.part-0-2.txt
- map.part-1-0.txt
- map.part-1-1.txt
- map.part-1-2.txt

Each worker must be able to read files written by any other worker, as well as the input files. In this lab you'll run all the workers on the same machine, and use the local file system.

Next, the master assigns reduce tasks to workers. This task reads corresponding intermediate files and calls the reduce function. The reduce tasks produce *nReduce* result files. With $nReduce = 3$ this step will generate three files:

- reduce.part-0.txt
- reduce.part-1.txt
- reduce.part-2.txt

Don't delete map and reduce files and keep the correct naming structure: as part of the evaluation we will audit your files. All the files need to be stored in *outputdir*.

After the reduce completes, the master node calls merge() which merges all the *nReduce* files produced by the previous step into a single sorted output and saves as *output.txt*. After all tasks are completed the master shuts down all its workers and then itself.

Hand-in instructions. We expect you to submit a well-documented code repository along with the report on your implementation details that should include chosen method for master-worker communication and task scheduling. Your application needs to store all the intermediary files and produce sensible log output, e.g. when each task starts/finishes, worker assignment, etc.

The lab is due: 11:59PM Tue, Apr 16 2024