Week 3 Lecture Summary

**Lecture and Lab**

This week we extended our conversation about MapReduce by discussing its implementation. The MapReduce programming model operates by using key/value pairs as input to generate intermediate key/value results that are later merged/reduced to a single/set of values (as defined by the reducing function). Though MapReduce is not purely functional, the framework still makes use of functional programming paradigms which allows for automatic parallelism. Unlike a pure functional framework, MapReduce functions can produce side effects and therefore can be deterministic which aids in fault tolerance. We also briefly discussed an overview of the original MapReduce paper written by Jefffrey Dean and Sanjay Ghemwat from Google. The details of this paper are discussed below.

Our lab assignment for the week was to get familiar with google cloud. We created projects for our group, a VM instance, confirmed our ability to run Python/MRJobs scripts on the VM and learned how to run MRJobs using a Hadoop cluster and Google Cloud Dataproc.

**Readings**

**Hadoop: The Definitive Guide**, *Chapter 2: MapReduce*

Tom White

The chapter gives an hands-on overview of MapReduce using a NCDC weather dataset. It starts by using an awk program to retrieve the max temperature by year and it takes 42 minutes to run against all the files and on a single machine; this is where MapReduce comes in where the work to improve such aggregation such a parallel processing, combining, coordination of multiple machines, etc is handled by the framework. A MapReducejob is broken into two phases: map (defined by the user-defined mapper function) and reduce (defined by the user-defined reducer function). Hadoop runs the job by dividing it into tasks which are then scheduled using YARN to run on nodes in a cluster. For load balancing efforts, Hadoop divides the input into fixed-size splits and processes them in parallel. Hadoop provides users data locality optimizations and fault-tolerance given processing over multiple machines. Hadoop can run MapReduce programs written in several languages including Python and Java.

*MapReduce: Simplified Data Processing on Large Clusters*

Jefffrey Dean and Sanjay Ghemwat

As one can tell, there’s a lot of redundancy between the lectures and all of the readings. To avoid being redundant, let’s discuss information is unique to this paper:

The execution framework of MapReduce starts by splitting the input files and making several copies of the program for the machine cluster. The cluster is composed of a master and several worker nodes. The master node is responsible for assigning map and reduce tasks to idle workers. Mapper tasks’ intermediate results are written to local disk and worker nodes responsible for reducer tasks read these buffered results using RPCs. Using a keying mechanism, the reducer tasks sort the intermediate data, apply the reducer function, and write the results to a final output file. Oftentimes, the output of one MapReduce job is used as input into another. To aid in fault-tolerance, MapReduce uses a heartbeat mechanism to ensure the availability of worker nodes. In the event of failure, completed map tasks need to be re-executed due to results being stored locally, yet the same does not apply for reducer tasks. Backup tasks are used to increase the processing speed of straggler nodes. The paper ends by giving two examples of MapReduce in action by creating distributed grep and sort programs which illustrate that the biggest bottleneck in the MapReduce implementation is the network traffic related to I/O.