Map-Reduce Replicate

By implementing this assignment, a student should be able to:

- understand the map/reduce programming model and be able to write their own map and reduce functions.
- understand consensus in distributed systems and implement consensus between different processes using Paxos.
- use Mutli-Paxos to replicate logs of map/reduce results to save the state of any performed computations and tolerate failures of any compute nodes.
- have experience using a cloud computing environment.

1 Introduction

MapReduce [1] is a simple programming model that is widely used to process big data in large clusters. A programmer writes a *map* function that processes key/value pairs to generate intermediate results of key/value pairs and a *reduce* function that aggregates the intermediate results from different mappers to calculate the final results. It is important to point out that the end-to-end computation can be composed of multiple map/reduce rounds.

In this assignment, you are required to implement the simplest word count example as an application of map/reduce. Assume we have multiple text files and we want to count the occurrence of each word to determine the most popular words in group of documents. You have multiple computing nodes and each node is running multiple processes to process these files. You are required to implement a Command Line Interface (CLI) where clients can submit files to your map, reduce, and replicate processes. You are required to implement the Map-Reduce Replicate assingment in a Cloud environment, eg, UCSB's Eucalyptus, or any other publically available cloud, such as AWS, AZURE, Google Cloud Engine, etc. Figure 1 illustrates a generic design for the map/reduce replicate architecture. In Section 2, we explain the details of the architecture design components. In Section 3, we explain the details of the client API. Milestones are listed in Section 4 and submission guidelines in Section 5.

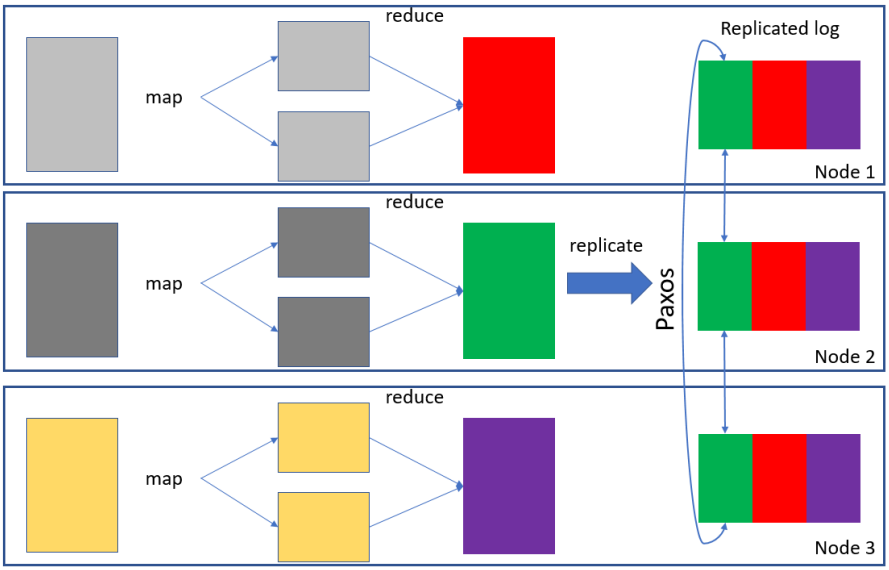

Figure 1: Map-Reduce Replicate model

2 Architecture Design

We assume that each computing node (physical or virtual machine) runs 2 mappers, 1 reducer, a CLI, and a Paxos replication module. Eucalyptus cloud computing nodes will be provided for testing. Each component is a separate process that listen on a socket to receive commands from the CLI. In the CLI, a user can submit commands like *map, reduce, ...* and based on the submitted command, the CLI sends a message to the corresponding component to execute the command.

The mapper: is a process that executes the map function. We assume that each computing node has 2 mappers running and each mapper is configured using a port number(e.g. 5001) and a unique id (e.g. 1). A mapper receives a **text** filename **F** *(e.g. file1.txt)*, an offset **O** *(e.g. 0)*, and a size **S***(e.g. 1000)*. Then, it opens the file whose name is F, reads the text starting from offset O, with the size S. The mapper splits the text on spaces into words, converts each word to an orders pair ⟨word,1⟩, and writes that to an output file named `{F}_{I}_{mapper_id}` where F is the original filename, I means intermediate, and `mapper_id` is the id of the current mapper. Table 1 shows an example of input, command, and output of a mapper.

The reducer: is a process that executes the reduce function. Each computing node runs 1 reducer and it is configured by a port number (e.g. 5003). A reduce receives a message consists of multiple **intermediate** filenames(e.g. `f1_I_1,f1_I_2`). Then it reads these files and merge the counts for each word. It could be implemented using a dictionary where a word is a keys and its count

Input File (f1)	Received Command	Output File (`f1_I_1`)
This is an example of a mapper mapping text to ordered pairs	map input.txt 0 19	⟨This,1⟩ ⟨is,1⟩ ⟨an,1⟩ ⟨example,1⟩

Table 1: An example of input/output of a mapper with an **id = 1**

is the corresponding value. If a word is not in the dictionary, it is inserted to the dictionary with a count of 1 in its first occurrence. A word count is incremented every time it occurs. A reducer should output keys and their corresponding count to a file named `{filename}_{reduced}` where filename is the original input filename. Table 2 shows an example of reducing two files to one reduced file.

Input File (`f1_I_1`)	Input File (`f1_I_2`)	Received Command	Output File (`f1_reduced`)
⟨This,4⟩ ⟨college,5⟩ ⟨an,6⟩ ⟨example,8⟩	⟨This,4⟩ ⟨are,18⟩ ⟨an,9⟩ ⟨school,4⟩	reduce `f1_I_1` `f1_I_2`	⟨This,8⟩ ⟨college,5⟩ ⟨an,15⟩ ⟨example,8⟩ ⟨are,18⟩ ⟨school,4⟩

Table 2: An example of input/output of a reducer

Paxos replication module (PRM): is a process that is responsible on replicating the reduced files with other nodes in the **same order** using Multi-Paxos [3, 2]. The PRM is configured using a port number and the set of IPs and port numbers of the other PRMs in other computing nodes. As shown in Figure 1, the PRM replicates the outputs in a log to achieve the same order in all the computing nodes. It is important to replicate log objects in the same order so that clients get the exact same results when they query the log in different computing nodes. The details of querying the log is explained in the client API in Section 3. The PRM is configured with a port number and it receives a filename to replicate (e.g. `f1_reduced`). The PRM converts the file into a log object that has the following attributes: filename and word dictionary. The PRM then tries to replicate the log object in the next available log index using Paxos protocol. If another PRM is competing for the same log position, a replicate command can fail. Therefore, the PRM has to retry replicating the log object in the following log positions until it succeeds.

3 Client API

The client API is divided into 2 categories: 1- data processing calls and 2- data query calls. In this section, we explain the details of different calls in both categories.

3.1 Data processing calls

- map *filename*
  splits the file based on its size into 2 equal parts. The split has to cut the file in a space character, **not in the middle of a word**. Then it maps each half to a mapper using message passing.

- reduce *filename1 filename2 .....*
  sends a message (using sockets) to the reducer with the provided filenames. The reducer has to reduce the intermediate files to a final reduced file.

- replicate *filename*
  sends a message to the PRM to replicate the file with other computing nodes. Notice that the PRM owns the log with all its log objects.

- stop
  moves the PRM to the stopped state. When the PRM in the stopped state, it does not handle any local replicate commands. In addition, it drops any log object replicating messages sent by other PRMs in other nodes. This is used to emulate failures and how Paxos can still achieve progress in the presence of $\frac{N}{2} - 1$ failures.

- resume
  resumes the PRM back to the active state. A PRM in the active state should actively handle local replicate commands as well as log object replicating messages received by other PRMs.

3.2 Data query calls

The CLI sends a query to the PRM, and the PRM prints the answers to these queries in its stdout. There is no need to pass the results back to the CLI. The supported data query calls are:

- total *pos1 pos2 .....*
  sums up the counts of all the word in all the log positions pos1 pos2, ...

- print
  prints the filenames of all the log objects.

- merge pos1 pos2
  apply the reduce function in log objects in positions pos1 pos2. In other words, it adds up the occurrence of words in log objects in positions pos1 and pos2 and prints each word with its corresponding count.

4 Milestones

Milestone 1 due 05/26 11:55 pm: submit a fully functioning implementation of the PRM. In addition, you should submit an implementation of the CLI

with the commands: replicate, stop, resume, total, print, and merge correctly implemented. For replicate, use hard-coded files similar to the expected output of the reduce function. There is a **5%** late policy per late day deducted from half of the project grade.

Milestone 2 due 06/12 8:00 am submit the implementation of the whole assignment with the CLI fully implemented. **There is no late submission for milestone 2**.

5 Policies and Guidelines

You may work in **two person** teams on this assignment. There is no restriction on the programming language but you have to use a platform that can run on any of the Linux CSIL machines. You should submit one tar file on Gaucho space that has a README, the source files, and an executable file that can run directly on a Linux CSIL machine according to different milestone deadlines. There should be an executable per component: CLI, mapper, reducer, and PRM. Each team will demonstrate their work in 06/12. The schedule of the demos will be published later. **It is important to work in this project as early as possible.**

References

[1] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[2] L. Lamport. Generalized consensus and paxos. Technical report, Technical Report MSR-TR-2005-33, Microsoft Research, 2005.

[3] L. Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.