15640 Project3 Report

Building a Map-Reduce Facility with specialized DFS

Name1: Jian Wang

AndrewID:jianw3

Name2: Yifan Li

AndrewID:yifanl

Part I: Design and Implementation

1. Design on MapReduce
   1. File I/O model

We assume users are using txt files as input files and expect to see the output files also as txt files. Each record inside the input file is assumed to be one line.

We offer a UserInputFile class as the key class to fetch the record in the input files on the DFS. The file operation during the map reduce process is all based on the record ID.

The user will offer their input file name and output file path in the DFS. The file I/O library will help the mapreduce facility to fetch the files and write files.

* 1. Master-Worker Framework

We use a master slave model in the map reduce core framework. There is only one master and multiple workers.

We have implemented heartbeat on each worker. The master will notice each worker’s state and schedule the tasks to them.

* 1. Scheduling

Splitting

We are doing splitting based on the fact of how many workers alive in the system and what are the maximum number of tasks they can support on them. The split is used to be combined with mapper class to form a map task.

Map Tasks dispatching

Each job’s map tasks are dispatched based on where their split file is located. We allow tasks to be queued. The queue size is as big as the maxTask parameter on each worker node.

Reduce Task dispatching

After all the map tasks are completed for a job, the master will generate the reduce task based on the reducer number that the user has required in the client configuration file. The reduce task are dispatched to the worker who still have free computing power to run the task.

* 1. Management Tool

We offer some commands to diagnose the system and DFS on the master process:

“help” show the

“ls” show the files in the DFS ROOT directory.

“ws” show the worker status.

“js <job-id>” show all the task status in specified job

“quit” quit the system

* 1. Failure and Recovery

We have designed to deal with two kind of failure.

The 1st one is task failure. In which case, if one task failed in the worker node, the system will notice this from the worker heat beat information. If the failed task is map task , the master will kill all the tasks on the worker where the task has failed and resent the tasks which on running on that worker. If the failed task is reduce task, the master will re-send the reduce task to the original reduce task’s duplication node.

The 2nd one is node failure. In this case, the master will resend all the tasks to the duplication node which is maintained by the DFS.

We haven’t successfully tested the task failure and node failure due to time issue. However, we argue our design is possible from our experimenting.

1. Design for DFS
   1. DFS upload client

This client is used by the MapReduce application developer. They will use this tool to upload their program input file to the targeted path in the DFS. Here, we only support uploading to the root dir on the DFS namenode which is “~”. The client will submit a upload request to the name node. Name node will decide how to chop the input file off and dispatch the chopped off file to the data node. The duplication of the input files are also generated in this stage.

* 1. DFS Name Node and Data Node framework
  2. DFS abstract file system
  3. DFS file reading and writing
  4. DFS file replication
  5. DFS data node failure and recovery

When data node failure is detected, the name node will set the dup node as the main node for each file that is located on that data node. New dup node will be generated here. Each task which is using these node will update their version of DFSFile and get the updated location.

This part has already been written but still haven’t been successfully tested. The code is in the NameNode class.

Part II: System Deployment and Configuration

Build

Deploy

Run

Part III: User Library and Simple Usage

Part IV: Test Case

1. WordCount
2. Maximum