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

1.1 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.

1.2 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.

1.3 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.

1.4 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

1.5 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.

2. Design for DFS

2.1 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.

2.2 DFS Name Node and Data Node framework

In order to reduce the implementation effort, we design the DFS as master-slave mode. The name node works as the pivotal of the DFS and keep the whole picture of the system. Data node works as slave node and listen to the command from name node. Every data node will launch a server and accept the download file request from other data node. This is used when the file gets chunk and replicated and also used when the reducer need to get all the output files from different mapper.

Additionally, the name node will run with the Mapreduce master on the same machine in the same process. They will run as different thread when system boot up. Data node and Mapreduce worker node will run in the same process and run as different thread. This design will reduce our effort to achieve the communication between Mapreduce and DFS.

2.3 DFS abstract file system

The name node will keep a virtual file structure on it's own. In order to achieve this, name node will keep a file chunk map for every DFS file. The

map will keep the file chunk to the data node address, port and local path.

2.4 DFS file reading and writing

When never the data node want to create a file on the DFS, it will send request to name node and name node will create a map for this. Whenever there is a need to read a file chunk, the name node will send download request to the data node with the target address and port information. The receiving data node will send a download request to the target data node server to get the file chunk.

2.5 DFS file replication

All the files on the DFS will be replicated on a different node. Whenever there is DFS file chunk created, the name node will pick up a data node to replicate this file chunk. In order to achieve this, the name node will send a download request to the replica data node with the target file address. Name node will keep this replica information in the file meta data. When node failure occur, name node will use the replica file and replicate the replica file again.

2.6 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

We do not provide make files. You need to use eclipse to build the project. The code we submitted already has the project configuration.

Deploy

In order to run the system, you need to run the master on one machine and run the worker at several different machines. There are three configuration files as:

- masterConfig.properties: this configuration file contains the port number
  - JobSubmissionPort: the socket port to receive the job submission
  - WorkerServerPort: the conmmunication socket number for every worker
  - DataNodeServerPort: the communication port number btween nameNode and dataNode
- workerConfig.properties: this configuration file contains the master address and data node local port
  - MasterAddress: master IP address
  - MasterPort: master port

DataNodeServerPort: the same as the one in masterConfig.properties
LocalPort: the download server port for the dataNode

- clientConfig.properties: this configuration file contains the master address and reducer number. It's used by the client program
  MasterAddress: master IP address
  MasterPort: master port
  ReducerNum: the reducer number

When you deploy the system, you need to copy the all the files to the master and worker machine. Change the MasterAddress and MasterPort in the workerConfig.properties and clientConfig.properties files to your master's IP address. Then set the ReducerNum in clientConfig.properties. We set it default to 2. Mostly you do not need to change other items.

Run

First, you need to cd to bin directory, then

1. on the master machine
   java mapreduce.master.Master
2. on worker machine
   java mapreduce.WorkerNode
3. on the client machine
   java dfs.DFSClient <master ip address> <name node ip in master conf file>
   input in cmd: upload <input file path> ~ <jobname>

Part III: User Library and Simple Usage

We provide five class in the user lib which are: Mapper,Reducer,Job, FileInputFormat and FileOutputFormat.

User need to implement and inherit these classes when submit their job.

Part IV: Test Case

Notice : remember to upload your file first in the DFSClient.

1. WordCount
   This example will count word occurrence in a file. For every word in the file, there will be a value account to the number of occurrence in the file.
   Java example.WordCount WordCoun_Input.txt ~ <file line number>

2. Maximum
   This example will find the maximum number in a file.
   Java example.Maximum Maximum_Input.txt ~ <file line number>