Disk Virtualization

Hardik Khichi 2016CS50404
Abhishek Maderana 2016CS50399
Aniket Kumar 2016CS50397

COL733: Cloud Computing Technology Fundamentals - Assignment 3

Group Number - G20

1 Part 1 - Consolidation Partitioning

We have a class BlockData, the objects of which represent each blocks. The class BlockMetaData represents the meta data associated with each block. The isfree variable captures the information whether a block is free or not(if something is written in the block or not). The class FileSystem represents the main file system that is based on the idea of virtual disks. diskA and diskB represent the physical disks with capacity of 200 and 300 block respectively. The file system has an array blocksMetaData of 500 size which stores the meta data corresponding to each block it saves the state of the blocks such as

- free or not
- block error or block is healthy

1.1 Functions

- write(self,blockNum, blockinf) : writes data to a particular block number (blockNum) after checking for valid block number(if blockNum lies between 1 500). It also checks if the blockinf coming in to be written is within the limit of the block size else throws error. While writing to a block we update meta data of that block such as isfree variable, iscorrupted etc.
- read(self,blockNum, blockinf) : This function reads data from a particular block number (blockNum) after performing the valid block number check and verifying that data exist in that block to be read.

1.2 Testing

The function runtestcases() checks the functionality by performing multiple write and read test using the above two functions. We try to write normally to a valid block, overwrite a block, writing to invalid blocks and writing larger than blockSize data. Then, we try to read from blocks with different checks and conditions. As verifiable by executing the code, we obtain the expected and explainable results.

1.3 Disk Creation Deletion

We implemented the functionality to create or delete virtual disks over the physical disks and blocks that we working with yet. Now the blockMeta is also containing extra variables

- id : which is the id of the disk it is allocated to
- isassigned : to which disk the block is assigned

We also added a Disk class which manage the data related to disk, it stores a list of mapping to all block the disk has been assigned.

1.4 functions

- createDisk(diskID, numBlocks): This function check if Block assignment to a new disk is done by first checking if the given block requirement could be full-filled(there exist as much blocks as much required) then by checking if assigning blocks in a continuous stretch is possible if not then blocks are assigned from different fragments.
- deleteDisk(id): If there exist a disk with the given id, this function delete that disk. free the blocks assigned to that disk and set isfree of all blocks to True(no data in blocks).
- writeBlock(diskId, blockNum, buffer): After validity check of the diskId, blockNum and size of the given data in buffer, this function writes that data to the given block.
- readBlock(diskId, blockNum, buffer): After validity check of the given inputs this function reads data from the given block.

1.5 Testing

The functionality test of the above implementation of Disk creation, Deletion, writing block to disks, and reading blocks of disk is performed. After successful execution of these functions was confirmed we did the testing using runTestcases() function. This function tries some disks, writes to them, read from them, delete a disk and check for handling of fragmentation. With these we test all the functionality of the virtual disks in our file system.

2 Part 2 : Block Replication

The block replication is an essential feature of any cloud feature with realiability. We incorporate fault tolerance by the concept of block replication, in which we try to store one another replica of each block. So, in case a block gets corrupted, we can recover the information from the replica. In our case we have implemented this feature by creating two disks for each disk creation, one primary and a secondary disk. Both the disk's class will save their own mappings to blocks.

When we call the deleteDisk function both the disks (primary and secondary) gets deleted.

We also added a variable to blockMeta data to show if a block is healthy or have faced blockerror. these cases were handled by readBlock and writeBlock respectively to to create another copy of a block if primary block gets corrupted. This is done when we tries to read from a corrupted block.

We tested the functionality with help of two functions:

- corruptSurely(diskId, blockId, isprimary) : with help of this function we can surely make a block currupted to test.
- corruptRandom(diskId, blockId, isprimary) : With help of this function we can introduce errors in block with some probability.

3 Part 3 : Checkpoints with Snapshots

By creating snapshots at different times user can restore the disks to an earlier version, or say checkpoints. This is a very useful functionality as we can create multiple checkpoints of a disk at different places and then can get back to one of them whenever we require. This is implemented in two functions:

- createCheckpoint(diskId) : This function create a checkpoint of the disk with given id(diskId) and return an integer representing checkpointID. It first checks for validity of diskId and then creates a Snapshot object. This object stores the blockData and blockMetaData of the given disk as a dictionary of snapshot id and list of tuple of blockData and blockMetaData. Each disk now also stores the metadata of checkpoints as dictionary snapId and SnapShots.
- rollBack(diskId, snapId) : This function restores the given disk to the given checkpoint after testing the validity of the given input. It is done by looking into to Snapshot dictionary with given snapId and restoring the blockdata and BlockMetaData to the disk.

We also worked on an another efficient method for implementation of snapshoting in which we can just create a class SnapShot which will store the further changes in blocks after the snapshot is generated, this could be done by implementing a class with following data:

- list mapping: if a block contains data and after snapshot gets written with other data, this data could be saved in some other blocks with different block id then the given block, and a mapping for that block is saved so we can read from that block when a readBlock is called.
- hidden attribute in blockMetaData : this attribute represents that a block is containing data corresponding to some other block which can only be accessed from list mapping of snapshot class.

With this implementation if we need to restore to a previous version we need to revert back all the changes till that snapshot with the help of all the mappings in list mapping of snapshots.

This could be a git like version control on virtual disks.