Indian Institute Of Technology, Delhi

COL733: Cloud Computing Technology Fundamentals

Instructor: S. C. Gupta

Assignment 3: Design for Disk Virtualization

Report

September 19, 2019

Submitted To:
S. C. Gupta
Professor
Computer Science Department

Submitted By: (Group 4)
Shantanu Verma 2016CS10373
Pradyumna Meena 2016CS10375
Manav Rao 2016CS10523
Shubham 2016CS10371

Index

1. Design (physicalDisk.py)

This is the main file where all the classes are implemented. It contains the following classes :

1.1. Block Class

It is the main data block. It contains the data stored at that block as well as the index of the block which stores it's replica.

1.2. PhysicalDisk Class

This is the main Physical Disk class. It contains all the supporter functions which would be required for updating, deleting and maintaining the disk and its data. Physical memory is stored as list of individual blocks.

1.3. VirtualDisk Class

This is the main Virtual Disk class. It abstracts out the underlying multiple physical disks by storing the mappings of virtual and physical location of the blocks.

Following methods are implemented which are called internally by the APIs which are in turn exposed to the user:

1.3.1. write()
1.3.2. read()
1.3.3. getBlockReplica()
1.3.4. setBlockReplica()

1.4. Fragment Class

This class is used to implement the fragments present in a disk. It stores the number of blocks present in a fragment and the individual blocks are recovered using the mapping stored in the DiskManager class.

1.5. Disk Class

This class is used to implement a Disk. It stores the list of commands executed on the disk, checkpoints and the patches present in the disk.

2. Virtualization (virtualDisk.py)

2.1. Consolidation and Partitioning

2.1.1. **createDisk -** This function consists of two checks before the actual function call. It first ensures checks if there is enough space to create the disk and then checks if a disk with the same id exists or not. Then a function call is made to createFragment().

2.1.2. **createFragment -** It first looks for the smallest unoccupied fragment which could accommodate the complete disk. If no such fragment is found then we need fragmentation for creating fragments. First the largest patch is first associated with the disk and then a recursive call is made to createFragment() to look for other unoccupied fragments for remaining portion of the disk.

2.1.3. **readDiskBlock -** After ensuring that the supplied disk_id and block_number are valid the main task is to translate the supplied block_number which is indexed relative to the particular virtual disk to the block_number in continuous virtual space. For this, a helper function is called which iterates over all patches associated with the disk and returns the serialized block_number. After that the read function from the VirtualDisk class is called.

2.1.4. **writeDiskBlock -** This function is also similar to readDiskBlock. After all the checks and the block_number translation have been completed a call is made to the write function of VirtualDisk class.

2.1.5. **deleteDisk -** We combine the fragments used by the corresponding disk with the unoccupied fragments. Also the number of used blocks is decreased by the number of blocks of the disk. For merging the fragments another helper function viz mergeFragments is called.

2.2. Block Replication

Block replication is implemented with the help of a pointer initialised in the Block class. Every block stores the virtual address of the replica which is initialised to -1 in the beginning. After data is inserted into the block, it is replicated in another location in the disk and each replica stores a pointer to the original block and original block stores the pointer to the replica. We have created a single replica system.

2.2.1. **readDiskBlockReplica -** This function tries to read the original disk block from its location but with the error probability of 0.1. If there is no error then the block is read and the data is returned. But if the data is corrupted, we move to the replica of the block using the replica pointer and try to read the block. At the same time since one replica is corrupted we create a new replica using the current replica from which we are reading the data and finally mark the current replica as original and new replica as its replica.

2.2.2. writeDiskBlockReplica -

Using this function we write the data into the specified location and we create a replica of the block and store it in a new location in the disk at some distance from the original block.

2.3. Snapshot and RollBack

This feature is implemented by maintaining a list of all the commands executed over the disk after its creation. It also maintains the list of checkpoints where any i^{th} index in this list stores an index from the commands list denoting the number of commands executed till that checkpoint.
Adding this feature required to append every command to the command list associated with the corresponding disk.

2.3.1. checkPoint -

Appends the current length of commands to the list of checkpoints and returns the index of the latest checkpoint.

2.3.2. Rollback -

First the list of commands and checkpoints are retrieved for the corresponding disk and then the disk is deleted. Then a new disk is made and the commands are executed till that checkpoint are executed on the new disk.