**COL 733:  CLOUD COMPUTING TECHNOLOGY FUNDAMENTALS**

**Assignment 3**

**Disk Virtualisation**

**Group 11**

**Members:**

* Ankit Shubham

2014CS10158

* Ayush Gupta

2014CS50281

* Deepak Bansal

2014CS50435

* Kapil Kumar

2014CS50736

**DISK VIRTUALISATION (CONSOLIDATION & PARTITIONING)**

1. **Read/Write of Blocks**

We have a class *BlockData*, the objects of which represent each blocks. The class *BlockMetaData* represents the meta data associated with each block. The *free* variable captures the information whether a block is free 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.

**Functions**

* **writeBlock (self, blockNum, writeData):** writes data (*writeData*) to a particular block number (*blockNum*). It initially performs the valid block number check to verify if block number lies between 1 & 500 else returns an Invalid Block Number error. Secondly, it checks whether the size of the *writeData* exceeds the maximum size limit, *blockSize* (100 bytes here). When we write to a block, we update it’s free variable to False and store the data either in *diskA* or *diskB* depending on the block number.
* **readBlock (self,blockNum,readData):** it reads data from a particular block number (*blockNum*) after performing the valid block number check and verifying whether block actually has any data to be read or is free.

**Tests**

The function *runTests()* performs multiple write and read tests 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. **Disk Creation & Deletion**

We incorporate the functionality to create or delete virtual disks over the physical disks and blocks that we have dealt with yet. Now the meta data of the block is expanded to also contain variable *allotted* that captures whether a block has been allocated to a certain disk or not and as well the disk id (*diskID*) of the disk it has been allocated to.

We define the following functions in the class *BlockMetaData:*

* **freeFromDisk()**: It sets a block free and also unallocated to any disk.
* **allotToDisk(disID)**: it allots a particular block to the disk with id as *disID.*

We add a class *DiskData* that stores meta data of disk including following variables:

* **idDisk:** ID of a disk that uniquely identifies it among all other disks.
* **sizeDisk:** stores the size of the disk i.e. number of blocks the disk has.
* **startIndexZeroIndexed:** It stores the starting block number for the disk.

**Functions**

* **createDisk (diskID, numBlocks)** : It attempts to create a disk with the id as *diskID* and number of blocks as *numBlocks*. In the current scheme, we check if our physical disk has got as many contiguous blocks else we raise an error. The *diskList* variable in the *FileSystem* class is updated to contain this disk’s id as well.
* **deleteDisk (id)** : It attempts to delete the disk with the id as *id.* After initial validity checks, it essentially calls the function, freeFromDisk() to free the blocks that are associated with this disk.
* **writeDisk (diskId, blockNum, writeData)** : It attempts to write the data, *writeData* in the virtual block number, *blockNum.* Internally, It calculates the physical block number to be actually written.
* **readDisk (diskId,blockNum,readData)** : It reads contents of the block with virtual block number as *blockNum* into *readData.* To read data from the physical blocks, it calculates the physical block number of the block and therefore, callsthe *readBlock* function.

**Tests**

We perform comprehensive tests to check the functioning of the above file system with added support for creating, deleting, reading and writing to disks. The *runBlockTests()* first tests the basic functions, *readBlock* and *writeBlock* to write and read to blocks. Once, we see them working successfully, we then perform the disks test, using the *runDiskTests()* function. It tries to create a normal disk, disk with an existing ID, delete a normal disk, delete an already deleted disk and then reads and writes from a disk. We obtain quite expected and explainable results from the above tests.

1. **Fragmentation**

The above structure of disk is quite elementary and suffers from growth and fragmentation issues. In this disk scheme, we relax the constraint to have all the blocks of a disk in contiguous storage. Therefore, we need to store