

COL 733 Assignment 3

Disk Virtualization

Shubham Rawat (2013CS10258)
Sahil Jindal (2013CS10253)
Prakhar Gupta (2013CS10245)
Anupam Khandelwal (2013CS10212)

13th September 2016

1 Consolidation and Partitioning

1.1 Random Read/Write to Blocks

In this part, a class of BlockInfo is maintained storing the size of the block info object. Two arrays are maintained in the main class (VFS) - one of size 200 and the other of size 300 which are collectively used to store the data. Another array containing the blockInfo of all the 500 blocks is also maintained in VFS class.

While writing to a block, its id is confirmed to be valid ($1 \leq \text{id} \leq 500$), length of block info is checked to be less than 100 bytes. Block is retrieved from one of the arrays based on the block number (from array1 if $\text{id} \leq 200$ else from array2), the blockinfo is written at the block and the "free" variable in its metadata is set to false.

While reading from a block, id of the block is confirmed valid and checked that the "free" variable in its metadata should be false. Then block number is resolved as to from which array we need to retrieve the data. Then the block information is stored in the object passed as parameter of the readBlock function.

For testing purposes, multiple read/write of blocks covering most of the corner cases like, invalid block number, length of block info ≥ 100 while writing, rewriting over blocks etc.

1.2 Create and Delete Disk

Various classes maintained in this subsection are:

- BlockInfo:
 - size: Size of block
 - free: if the block does not have data written to it
 - unallocated: whether block is allocated to a disk created by user or not
 - disk_id: id of disk to which block is allocated
- DiskInfo:
 - size: Size of the disk
 - start: mapping of first block of the disk to the position in total block space
 - disk_blocks: This function returns the range of blocks allocated to this disk
- VFS:
 - array of size 200 blocks
 - array of size 300 blocks
 - Array storing metadata for each block
 - Array containing metadata for each disk

To create disk, the API request specifies the id and size of disk required. The program checks if any disk with given id already exists or if the disk can be allocated space. If the tests pass, diskInfo object is stored at position id in the disk_metadata array containing the starting block number and size of the disk. Also the block metadata for every block in the range is changed by setting the unallocated variable to false and setting the disk_id variable to the given id.

To delete a disk the API request specifies only the id of the disk. If a disk is found with the given id, its metadata is removed from the metadata array, and block metadata for every block in the range is reset to default.

To write to a block, we are given disk id, block number w.r.t. the disk, and info to be written at the block. The program checks if the disk id given has actually been created and block number specified lies within size of the disk. Then the address of the block is converted to global address and is written to as in previous subsection.

Similarly reading of block takes place.

For testing purposes various create/delete/read/write operations were run along with some invalid operations such as specifying disk id which does not exist, while reading block/deleting disk. Here the problem of disk fragmentation is not yet handled and hence if the program does not find a continuous chunk of free memory available it return out of memory and is unable to create specified disk. This will be handled next.

1.3 Handling disk fragmentation

Here, the disk info object contains an array of blocks which are allocated to it as compared to start and size in previous case as the blocks allocated now may or may not be continuous. In VFS class an additional array is maintained containing a linked list of free blocks.

To create a disk, id and size are checked to be valid. Then size number of blocks are popped from the free blocks list and put to the blocks array in disk info metadata.

To delete a disk, the blocks are popped from the disk info metadata and added to the free blocks list.

Reading/Writing to blocks is similar to the previous cases.

For testing purposes various create/delete/read/write were performed. Also create with not enough contiguous space available was performed which created a disk successfully thus solving the problem of fragmentation.

2 Block Replication

Here block info class is modified to contain variables "error" and "replication". "error" stores if the block has had a read error and is corrupted, and "replication" variable stores the block number of the replica of the block (null if no replica stored yet). Create and delete disk operations remains same as previous create/delete operations.

While writing a block, the original block is written in the same way as the previous write block operation. Now the program finds another free block (if replication variable in block info is null) and writes the block information onto the block and sets it as the replication variable in the block info.

While reading a block, initially the information is read from the original block but there is now a random probability that the read might give an error. If the read returns an error, the "error" variable in block info metadata is set true, and block is read from the replication block if one exists. If a replication block exists and there is an error in reading from the original block another replica is created at a free block and written to the replication variable in original replica block info metadata.

For testing purposes, enough number of read/writes were performed to generate a read error and verify if information read from a replica is correct.

3 Snapshotting

In this part, an array of snapshots is maintained in the disk info metadata class. Create/Delete/Read/Write operations are same as in section 1.

To take a snapshot, create_checkpoint API is used taking disk id as its parameter. For each block in the disk, its metadata and information are stored in the snapshot in form of an array of pair objects. This snapshot is then added to the array of snapshots maintained in the disk info object and the snapshot id (size of snapshots array - 1) is returned to user for further use of this checkpoint.

To rollback to a particular snapshot, disk id and snapshot id are verified to be valid. The snapshot corresponding to the given snapshot id is loaded from the array. For each block in the disk, the metadata and information stored in the block is replaced by the metadata and information of the block stored in the snapshot.

For testing purposes, the disk contents were overwritten and then it was rolled back to the checkpoint before the overwriting and hence previous information was successfully retrieved.