

OHSM Block Allocation **fscops@gmail.com**

OHSM relocation uses one of the six relocation algorithms which are as follows.

1. Pageflip
2. Memcopy
3. Block Number
4. Pageflip with truncate
5. Memcopy with truncate
6. Block Number with truncate

OHSM adds its own customised block allocation function to the File System. The inode contains home tier-id and destination tier-id information for a particular file. During relocation a file is checked against the relocation policies and then accordingly the destination tier-id is set in the inode. When new blocks are allocated for the relocation, the OHSM block allocation function refers destination tier-id field and allocates blocks from the destination tier.

Each of these method uses “**Tricky Copy and Swap**” algorithm, and each method is tricky in itself.

Tricky Copy and Swap Algorithm:

Tricky copy and swap algorithm starts by allocating a new ghost inode and reading the source inode in memory. It then takes lock on source inode to stop any further modifications to the inode in the course of relocation. Later, it reads the data blocks from source inode and copy them to the destination inode, block by block. The reading of source block data is done through block buffers and they are then copied to destination buffer. The destination buffer is marked dirty. Finally, when all the data is copied to the ghost inode, the source inode is re-assigned with the contents of destination inode's i_data by swapping them with the i_data of the ghost inode. The source inode now contains pointers to new data blocks. At this point, the source inode is unlocked, synced and destination inode is released. All the six relocation algorithms use this methodology, only the implementation differs.

Pageflip

The name pageflip suggests that this method works around pages. This method is an interesting one, it just changes the page pointer of destination buffers to which it maps to by the page pointer of the source data blocks, tricky isn't it.

Pageflip method reads data from source inode into page using get_block function of file system. Simultaneously it allocates data blocks for ghost inode using the same get_block function, the buffer heads of the blocks are in memory. The data of both source and ghost inodes are in different pages, the only difference is that the page of ghost inode is blank. So here's the trick, we simply change the page pointer of ghost data block buffer heads to the source page and we get the source data in ghost inode. Then, we sync the page to disk. When all the data is copied the source inode is updated as specified in the “*tricky copy and swap*” algorithm.

Memcopy

This method starts in the similar way as Pageflip does. The only difference being that for copying the contents of source data block to destination data block, memcopy function is used. This is a CPU intensive process, so it takes a little time. The destination buffers contents, now is a replica of source buffer. The buffers are then marked dirty and synced. When all the data is copied, i_data of source and ghost inode is swapped.

Block Number

This is something interesting again. It starts in the similar way as tricky copy does. Here it is not copying anything. But then the question is how to copy data? So, here is another trick. Simply change the b_blocknr and b_bdev of source data block buffer to that of destination data block buffer's b_blocknr and b_bdev. When the page is synced, the data is written into new data blocks. When all the data is copied to destination inode we swap i_data of source and ghost inode, and we are done.

Truncate Methods

The last three methods out of six, mentioned at the very beginning uses truncate methods. The methodology of pageflip, memcopy and changing b_blocknr and b_bdev is same as explained, difference lies in truncation of source inode. This code efficiently allocates data blocks for ghost inode, copies the contents from source inode and truncates the source inode. The truncate method can be used as a defrag tool. Whenever a file is relocated logically it requires the same amount of space as that of the file in destination tier, but in case of defrag the file resides in same tier. Consider a case of a 700mb fragmented file in 1200Mb partition, and space left is 300Mb. If this file is to be defragged then of course a truncate on source inode is required to get extra 400Mb. Apart from defrag, this method can also relocate files in specified tier. The disadvantage of this method is that it is data inconsistent. If the system crashes during relocation, source data will be lost.

Well, all the methods described above have some or the other side effects. But it has been our best efforts to stabilise it to the most.

For any further comments/queries mail fscops@gmail.com