Online Hierarchical Storage Manager

Sandeep K Sinha
NetApp, India
sandeepksinha@gmail.com

Rishi B Agrawal
Symantec, India
rishi.b.agarwal@gmail.com

Vineet Agarwal
checkout.vineet@gmail.com

Rohit Vashist
rohit.k.vashist@gmail.com

Rohit K Sharma
mailboxrohit19@gmail.com

Sneha Hendre
sneha.hendre@gmail.com

Abstract

Intel, Sandisk, and Samsung are investing billions of dollars into Solid State Drive technology and manufacturing capacity. Unfortunately due to the extreme cost of building the manufacturing facilities, SSD manufacturing capacity is not likely to exceed HDD manufacturing capability in the near future. Most data centre applications heavily lean toward database applications which use random read/write disk activity. For random read/write activity, the performance of SSDs is 10x to 100x that of a single rotational disk. Unfortunately, the cost is also 10x to 100x that of a single rotational disk. Due to the limited manufacturing capability of SSD, most applications are going to remain on rotational disk for the foreseeable future. Online Hierarchical Storage Manager has been developed to allow SSD and traditional HDD (including RAID) to be seamlessly merged into a single operational environment thus leveraging SSD while using only a modest amount of SSD capacity.

In an OHSM enabled environment, data is migrated to and from the high performing SSD storage to traditional storage based on various user defined policies. Thus, if widely deployed, OHSM has the ability to improve computer performance in a significant way without a commiserate increase in cost. OHSM being developed as open source software also abolishes the licensing issues and the costs involved in using storage solution software. OHSM being "online" signifies the complete abolishment of the downtime and any changes to the existing namespace.

1 Introduction

Hierarchical Storage Management is a data management technique that uses devices in an economically efficient manner, thus reducing the storage space and administrative costs associated with managing data.

OHSM is an online hierarchical storage manager for Linux which offers policy based transparent movement of data from one class of storage to another. Being the first attempt towards an open source data manager, it provides a base platform for all further developments in similar areas. It supports policy based migration of files i.e. it defines a set of policies which decide the correct placement tier for a file during its initial creation, as well as block allocation and relocation of the file from one placement tier to another. A placement tier is basically a storage class, which consist of a collection of storage devices with similar properties defined in the policy file based on its speed, cost or any other attribute. These placement tiers can be priority-ordered and can be overlapping as well. These policies are enforced on a OHSM enabled file system through an XML based placement policy file. A placement policy file contains a collection of rules which decides both, the initial file location and the circumstances under which existing files are relocated. Therefore, placement policies have been broadly categorized into placement and relocation policies.

Whenever a file is created it will be allocated in accordance to the placement policy which has been enforced on the file system. If the file fails to match any of the rules specified in the policy file, it falls into the default allocation method that is used by the underlying file system. Similarly for relocation, whenever a file matches any of the relocation policy, it is relocated from the source tier to destination tier as specified in the relocation policy file. The migration of data is non disruptive and completely transparent to the placement tiers. The placement policy file also contains the mapping information between the storage devices and the respective placement tiers to which they belong. OHSM does not impose constraints on the placement tiers as far as capacity, performance and availability are concerned.

OHSM also provides functionality to remove files on certain events which can be specified through the policy file. Defragmentation of the relocating files could also be achieved by enabling defragmentation at the time of triggering relocation. Though OHSM doesn't guarantee complete defragmentation of data blocks, it does a best-effort attempt. Relocation is an event triggered operation based on single or multiple policies selected from the set of relocation policies aiming to provide greater flexibility and usability to system administrators.

2 Design

The idea here is to leverage the underlying device topology of the block-device logical volume and use this information to optimize the block allocation methodologies used by the file system, thus achieving storage efficiency. OHSM provides a framework to implement hierarchy based storage in the existing environment using support from the device mapper. This requires some modifications to the file system's file creation and block allocation routines.

The overall design of the system is composed of a group of inter-operating modules implemented as shared libraries, daemons and kernel modules. OHSM has various components including the user interface, an XML parser, OHSM Admin and the Kernel Driver with each of them offering different functionality to support the various services offered by OHSM.

2.1 User interface

Another key component of the OHSM system from an administrator's perspective is the administrative interface. It is comprised of both a graphical as well as a command line interface. Apart from the basic functionality like enabling, disabling and querying the state of the OHSM system, it also provides the administrator an opportunity to generate, modify and validate the XML policy file through a graphical interface. The graphical interface also provides various other statistical data including state of tiers, space utilization, number of files relocated and various other information related to the each placement tier. Apart from these facilities both the interfaces have a lot of services to offer such as getting and setting various OHSM runtime tuneable features, enforcing of placement and relocation policies on file systems and many more. In all, both the GUI and CLI offer a simple and easy to use interface to the administrators.

2.2 XML Parser

OHSM is a policy based hierarchical storage manager and it uses an XML based policy file for defining all the placement and relocation policies. The XML parser is responsible not just for parsing the administrator defined XML policy file but also for validating the information provided. The XML parser takes the policy files as input and further parses it to extract various information from it like the tier-device mapping, placement and relocation policies. Then it validates that information against the device topology map and checks for any conflicting policies. In case of any errors or conflicts it either reports back or else it transforms the parsed policy information into relevant data structures and passes it to the OHSM Admin module for further processing.

2.3 OHSM Administrator

This is the central communication hub which differentiates and communicates with all other components of the OHSM system. It also helps keep the design modular and simpler. It is also responsible for all the required communications between the user space and kernel driver to facilitate all the requests through the user interface. Most of the error handling is done by the OHSM Admin. All the communication with the user-space device mapper library to get the device topology mapping also goes through the OHSM Admin. The errors received from the parser and the device mapper library are processed and converted into user readable strings and passed to the user interface.

2.4 Device Mapper API

OHSM uses the user space device mapper library in order to extract the device topology beneath the logical

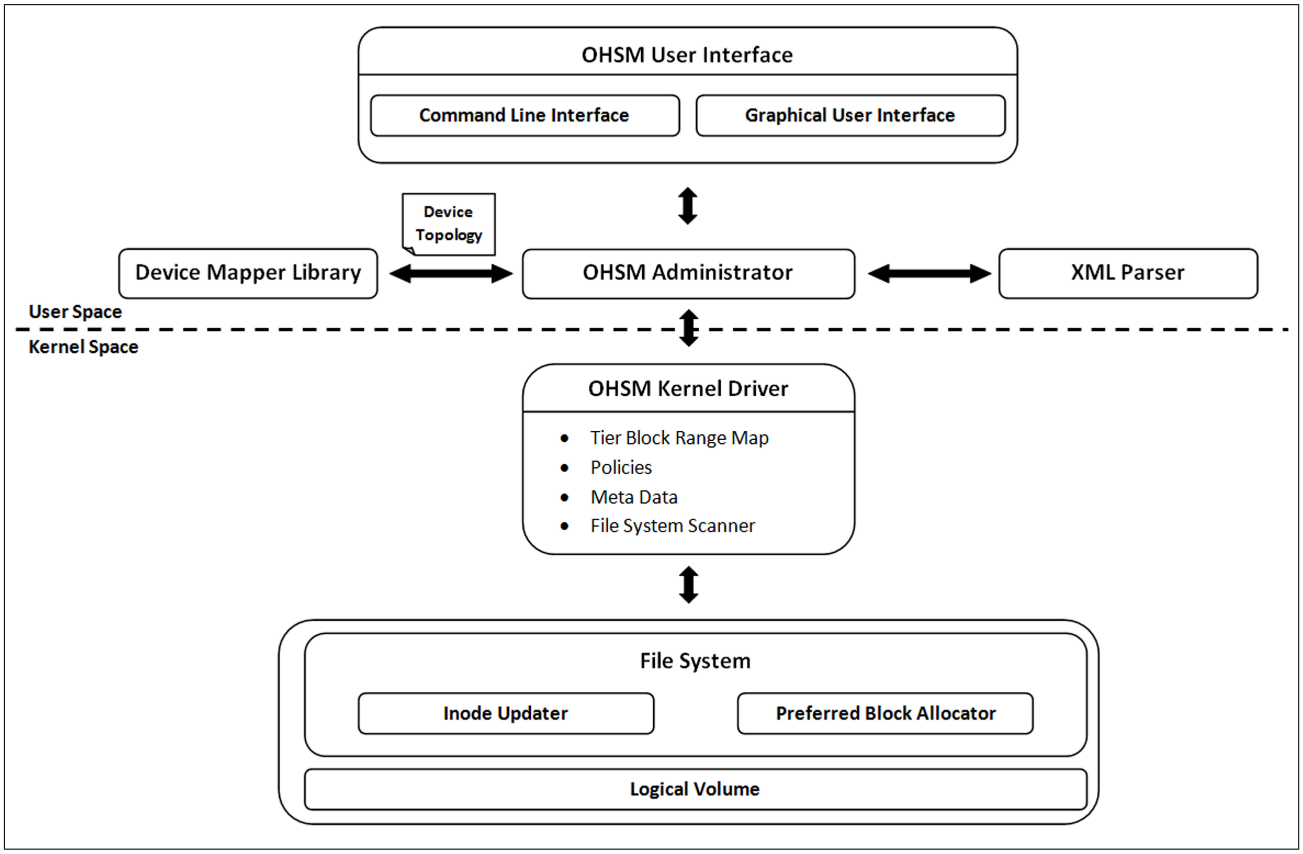

Figure 1: OHSM Architecture

volume on which the file system is already mounted. This helps the file system to optimize its data block allocation to provide better storage efficiency. The device topology along with the tier-device mapping information helps OHSM build its complete internal mapping data structures. The final mapping information resides in OHSM kernel driver. The library also offers certain other callbacks providing data regarding the underlying devices which can be used in later implementations of OHSM. This will help provide a more administrator friendly solution. This information can also be utilized to derive certain heuristics and statistical information regarding the placement tiers.

2.5 OHSM Kernel Driver

OHSM kernel driver is the most important component of the OHSM system. This is the core of the system and helps service all the requests from the user space. It stores various metadata and service routines associated with OHSM. It also holds the tier-device mapping table, placement and relocation policies associated with the file systems. The file system scanner and the complete mechanism of relocation have also been implemented as a part of the kernel driver. Apart from these, it also implements the various ioctl service routines. The kernel driver is loaded in the memory during boot time, so that the required information and functionality is available to the file system soon after mounting. Any problem with the kernel driver can lead to a complete freeze of the working OHSM. Hence special care has been taken to handle most of the error conditions gracefully. The kernel driver is the core of OHSM. However, it is the file system implementation that gives it its power. Keeping the two independent allows the file system changes to be minimally invasive and not require major OHSM specific patches to address inode updates and preferred data block allocation needs. The driver comes into picture once the administrator triggers relocation on the file system.

2.6 File System

OHSM system can work with most of the GNU/Linux file systems with some basic modifications to the way a file is created and extended. OHSM broadly divides those changes into two sub categories:

2.6.1 Inode Updater

As the name signifies, the changes revolve around the inode allocation mechanism for a file system. In an OHSM enabled environment, it is expected that the initial file creation be governed through some file creation policy. OHSM changes the way the normal file creation works and imposes an additional check of the file's physical characteristics against the placement policies specified. In case of any match, the home tier id is set for that file, eventually directing the file system to serve all the data blocks requests for that file from the pool of blocks belonging to its home tier.

2.6.2 Preferred Block Allocator

In an OHSM enabled environment, all the data block requests for a file are restricted to its home tier. This might lead to a situation wherein there is no free space left on the specified tier. In order to overcome such circumstances, OHSM offers an option for administrators to provide the tiers in a prioritized order, with the most desirable destination listed first. This change overrides the normal block allocation policy of the file system and makes sure that it follows the policies specified by the administrator, if any. Those files that don't qualify to any of the file placement criteria follow the usual semantics of file creation offered by the file system. Also, all further data block requests of such files are serviced from blocks spanning over the complete file system.

3 OHSM Value Proposition

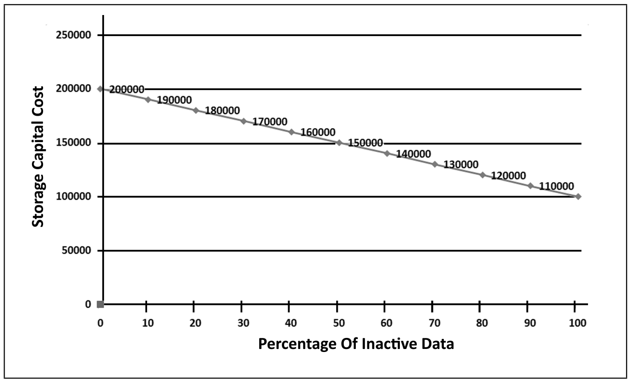

Figure 2: Value Proposition

The most broadly applicable benefit of the Online Hierarchical Storage Manager is to reduce the average online storage cost by migrating inactive files to a less-expensive placement tier in a hierarchical based storage environment. It should be assumed that the lower placement tiers have a significantly lower per-byte cost than storage in next higher placement tier. In most of the cases, the cost differential between different types of online storage creates the economic justification for such hierarchical storage environment. If the highest placement tier storage costs around $5 per gigabyte and mid range placement tier storage costs around $2 per gigabyte, an enterprise whose online data is 50% inactive could save around 30% of its storage acquisition cost by moving the inactive files to mid range placement tiers. Larger percentages of inactive files result in higher savings.

For enterprises that keep a significant amount of non-critical data online, a multi-tier storage strategy can offer substantial cost savings without adverse effects on business operations. The challenge in attaining the benefits of multi-tier storage is to get the right files on the right storage tier at the right time. OHSM achieves it very efficiently with the help of policy based allocation and relocation. The purpose of OHSM is to eliminate any administrative cost and complexity in a hierarchical storage environment by automating the relocation of files as levels of I/O activity against them rise and fall, as well as when their sizes, owners, or logical positions in the file system hierarchy change.

4 Working with OHSM

Information Lifecycle Management provides effective management of information throughout its useful lifecycle. ILM also provides strategies to allow a computing device to administer the storage systems. These strategies consist of policies which differentiate and administer data based on its usage and priority. In order to work with the Online Hierarchical Storage Manager the administrator needs to be quite aware of the various file placement and relocation policies supported by OHSM. The administrator needs to put together all this information along with the tier device mapping information into a XML file. This file is passed to OHSM to enable and enforce these policies on a file system. It should not be forgotten that OHSM also offers a graphical user interface to generate the XML policy file. The user can either use the graphical interface or the command line interface in order to enable OHSM. Below we examine various file placement and relocation policies with their respective sample XML policy grammar as supported by OHSM.

4.1 Tier Device Map

The tier device mapping information is required in order to define the set of devices that belong to a particular placement tier. Both the file placement and relocation policies are validated against the tier device map information specified in this section. All the information provided in the tier-device map section is also validated against the configuration of the system. Some of the validations involve checks for making sure that the specified devices exist on the system, all the devices are part of the same logical volume over which the file system is mounted, etc. A simple tier device mapping information:

```
<DEVICES>

  <DEV_TIER_INFO>
    <NR_TIERS>3</NR_TIERS>
    <NR_DEVICES>6</NR_DEVICES>
  </DEV_TIER_INFO>

  <DEV_TIER>
    <TIER>1</TIER>
      <DEVICE>/dev/md4</DEVICE>
      <DEVICE>/dev/md5</DEVICE>
    <TIER>2</TIER>
      <DEVICE>/dev/md3</DEVICE>
    <TIER>3</TIER>
      <DEVICE>/dev/md1</DEVICE>
      <DEVICE>/dev/md2</DEVICE>
      <DEVICE>/dev/md6</DEVICE>
  </DEV_TIER>

</DEVICES>
```

4.2 File Placement

Transparent and non disruptive relocation of data across various placement tiers is undoubtedly the most obvious use of the OHSM but it is not restricted to that. OHSM also offers various functionality to give targeted files a preferential placement at the time of file creation. We have seen an enterprise using database management system to manage their most critical data and provide that critical data a preference over all other data. OHSM currently supports four file placement policies which can be used individually or can be combined together to form various new rules. If a file qualifies for multiple placement policies then the first match prevails over all others.

For special situations where the target placement tier might run out of free blocks, the administrators have the facility to provide the preferential order of tiers for each such rule. This directs the file system to allocate blocks from a lower tier in case the target placement tier is already full. This feature is optional and can be enabled/disabled as per administrator's discretion. Currently OHSM supports a very primitive set of file placement criterion including file type, user ID, group ID and the logical placement of files in the file system hierarchy, directory name. Allocation policy based on directory name can be both recursive and non-recursive. The file placement policy can be based on the following:

4.2.1 File Type (FTYP)

In today's time, there is a strong likelihood that applications follow a pattern in the file name extension to determine the kind of data that the file holds. This pattern can be utilized to differentiate between different types of files like database files, media files and log files etc. Using the file extension we can also associate the file with different applications most of the time and derive its criticality based upon that. This information can be used to provide preferential placement to various type of files. We can also dedicate placement tiers to specific file types based on its type.

4.2.2 User ID (UID)

In a large server environment the file systems are mostly organized based more on the users rather than the applications. Consider the case of an enterprise, where different users have their home directory on the same shared file system. In such situations there can always be reasons to allot higher placement tiers to various users while restricting others to have a mid range or a lower one.

4.2.3 Group ID (GID)

Similar to file placements based on users, various groups in an organization can share a placement tier. This can be based on the criticality of data they operate on and at times the speed of data retrieval. Consider the case of an engineering team and a marketing group; it may be desirable to have the engineering data on a more reliable placement tier as compared to the marketing team. The accounting group can have opted for a separate placement tier for various other reasons.

4.2.4 Directory Name (DIR)

Often we create directories based on the current time or date. This helps us in keeping the data in a more structured manner. For instance, if someone keeps its reports for the last couple of years, there will be a number of directories present on the file system, for example, report-2007, report-2008 and report-2009 and so on. It is expected that the latest reports will be the most frequently accessed one. Though it is just an assumption, this can be used to provide placement tiers to various structured data classified on the basis of their age. File placement based on directory names can be recursive and non-recursive depending on the specification in the policy file. A simple file placement policy:

```
<ALLOCATION>

  <ALLOC_INFO>
    <NR_USERS>1</NR_USERS>
    <NR_GROUPS>2</NR_GROUPS>
    <NR_TYPES>1</NR_TYPES>
    <NR_DIRS>1</NR_DIRS>
  </ALLOC_INFO>

  <USER_TIER>
    <USER>0</USER> <TIER>1</TIER>
  </USER_TIER>

  <GROUP_TIER>
    <GROUP>0</GROUP> <TIER>1</TIER>
    <GROUP>501</GROUP> <TIER>3</TIER>
  </GROUP_TIER>

  <TYPE_TIER>
    <TYPE>ora</TYPE> <TIER>1</TIER>
  </TYPE_TIER>

  <DIR_TIER>
    <DIR>/foo</DIR> <REC>1</REC> <TIER>1</TIER>
  </DIR_TIER>

</ALLOCATION>
```

4.3 File Relocation

One of the most desirable things to have is to store inactive files on placement tiers of lesser quality so that it does not affect the applications adversely. If you look at it from the I/O performance perspective, if the file is accessed rarely and it is mostly inactive, the performance of the storage device underneath that holds it is irrelevant. This makes the ability to relocate files across placement tiers very critical and important. Also, there are situations where you have thousands of small files in a file system. It is seen that under such circumstances most of these files soon become inactive. Some of the scenarios are a document management system, a mail server or any database application using opaque data objects stored as small files. It would be highly desirable to have the ability to relocate data across placement tiers under such circumstances. OHSM currently has support for the following file relocation policy criteria.

4.3.1 File Access Age (FAA)

It signifies the time since the last access to the file which can be one of the most appropriate qualifiers for a downward relocation. Based on the time of last access to the file, it could easily be relocated to a lower placement tier. This can be useful in a search engines or mail server environment, where the files access rates go down as the time increases. This would not be a good candidate as a qualifier for relocation from lower to higher placement tier as this can be misleading at times. There can be files which are just accessed to know that it's not of use and the data is stale. Still, because of the file's access age being quite small, it can get relocated to a higher tier, which would be highly undesirable. The recent introduction to realtime in the kernel really changes how the last access time is managed in a significant way. And use of such a feature might eliminate most of the value of FAA.

4.3.2 File Modification Age (FMA)

This is a true qualifier for a relocation to happen from a lower to a higher placement tier. It can be fairly assumed that a file which has recently been modified or which has a smaller modification age would surely be accessed more frequently in the near future. Hence, this can be used for deciding upon the conditions for relocation from lower to higher placement tiers. Most of the stub based implementations for HSM also use modification age as one of the primary qualifier to bring back the data from their archival storage to their actual placement tier.

4.3.3 File Size (FSZ)

There may be various situations where it would be desirable to allow a certain size of file to reside on a specific placement tier. The reason is the limited size constraints of the higher placement tiers due to their higher costs. We move the file to a lower placement tier if the file size exceeds a specific threshold. This threshold can be based upon the amount of space that a higher placement tier has. So, the file size qualifier can be easily used to prevent situations where the higher placement tiers don't run out of space.

4.3.4 File I/O Temp (FIOT)

File I/O temperature is defined as the average number of bytes transferred to or from a file over a period of time. This is independent of the file size and is one of the more powerful qualifiers which can be used to automate the process of relocation.

4.3.5 File Access Temp (FAT)

File access temperature is defined as the ratio of the number of times the file has been accessed over a period of time. This helps us to determine the average I/O activity that is taking place on a file against all other files in the file system. Such a measure can be useful to find suitable candidates for relocation from both lower to higher placement tier and vice versa.

4.3.6 FTYP, UID and GID

Relocation policies can also be based on the file type, user ID and group ID qualifiers. Since, the initial allocation can be based on these qualifiers, there is a great chance that when these are combined with other relocation qualifiers, form a finer granularity relocation criterion. A simple file relocation policy:

```
<RELOCATION>
   <NR_RULES>1</NR_RULES>
    <RULE>
         <INFO>1</INFO>
         <RELOCATE>
              <FROM>1</FROM>
              <TO>2</TO>
            <WHEN>
              <FSIZE>50</FSIZE> <REL>LT</REL>
              <FAA>50</FAA> <REL>LT</REL>
            </WHEN>
         </RELOCATE>
    </RULE>
</RELOCATION>
```

5 Prototype Implementation for ext2/ext3

The prototype of OHSM involves basic implementation of the idea presented in the previous sections. The general concept of OHSM involves various modules and their relationship with the file system. OHSM consists of roughly four components, namely the User Interface, OHSM Admin, Kernel Driver and File system. Our prototype provides functionality for creating policy files, enabling and disabling of OHSM, and triggering relocation manually. The User interface provides a set of commands to control and monitor the various functionality offered. It also allows the user to create XML based policy files and logical volumes at the same time. In the prototype implementation the user is required to create separate policy files for allocation, relocation and tier device mapping. These policy files were required to be specified at the time of enabling OHSM on a file system. Before OHSM could be enabled, these policy files are required to be parsed and validated for any conflicts. After verifying the policy files, the information is stored in internal data structures. The OHSM Administrator uses the ioctl interface provided by OHSM kernel driver to control and administer the system. On receipt of the data structures the kernel driver replicates these data structures in the kernel, and acknowledges back to the administrator module success or errors if any. On success, EXT3_OHSM_ENABLE flag is set inside the file system's super block. When OHSM is disabled all the data structures are cleared and the flag is reset. In order to achieve this, minor changes were made to struct ext3_inode and a new flag was introduced to be used within struct ext3_super_block.

```
struct ext3_inode {
...
...
__u8 ohsm_home_tid;
__u8 ohsm_dest_tid;
};

/*
 *  Misc. file system flags
 */

#define EXT3_OHSM_ENABLE 0x0008 /* OHSM enabled */
```

When a file is created it is intercepted by the OHSM inode updater and an additional check is made against the allocation policy enforced on the file system. In case a file qualifies, its ohsm_home_tid is set to the corresponding tier id. Otherwise, it remains zero. Later, for any data block requests for files having a non-zero ohsm_home_tid, the call to the block allocation routine is diverted to OHSM's block allocation routine, if OHSM is enabled on the file system. This implementation requires variations in the existing file system structures and its block allocation strategy also known as ranged block allocation. Ranged block allocation improves proficiency of file system in restricting allocation of data blocks to a range of block groups. A table containing the map of tier against the block group ranges is maintained by OHSM kernel driver. Ranged block allocation also uses this information to allocate new data blocks for the file in a specific tier. This block group range table is used by the block allocation routine to identify the block group ranges of device hierarchy. This table is created at the time of enabling OHSM and remains active in memory until the file system remains mounted or OHSM stays enabled. At the time of unmounting or disabling of OHSM this information is dumped on disk to /etc/ohsm. This information is used later to reconstruct this table back when the file system is mounted back or OHSM is re-enabled on a file system.

In user space, the OHSM Admin uses libdevmapper to get the device topology and passes the extents of devices to the kernel driver. The driver later maps these extents to file system specific block group ranges. The ohsm_home_tid field of inode is used as an index in this table to get the specific block group range. The allocation routine bounds the data block allocation within the selected range. Figure 3 illustrates two scenarios. The left half of it illustrates the scenario when a file is created. It shows that initially the ohsm_home_tid is set to zero, which later gets updated by the inode updater where it is qualified against the various file placement policies. If qualified, the files ohsm_home_tid is updated accordingly with the specific tier id. On the right side, it shows the later scenario where there is block allocation request for a file. The block allocator checks for a non-zero ohsm_home_tid and extracts the relevant block group ranges for the same. For a file having ohsm_home_tid equal to zero, the block group range spans the complete file system. The call to the block allocation routine is diverted to Range block allocation in place of file systems normal block allocation routine, which eventually serves the purpose. In case the tier is full, the file's ohsm_home_tid is set to zero and the file system's normal block allocation routine is invoked.

Relocation currently is a triggered event and has to be started manually by the administrator. When relocation is triggered, OHSM kernel driver scans all the inodes in the file system and pushes each qualifying inode to the work queue for relocation. Prior to adding each inode to the work queue, the ohsm_dest_tid is set to the relevant tier for that inode. The workqueue handler routine is implemented in the OHSM kernel driver which picks and does the task of relocation. After the relocation is completed, the ohsm_dest_tid becomes the new ohsm_home_tid for the file. As an optimization, OHSM uses a Tricky copy and swap algorithm to complete relocation as fast as possible.

Tricky copy and swap algorithm starts by allocating a new ghost inode and reading the source inode in memory. It then takes a lock on the source inode to stop any further modifications to the inode in the course of relocation. Later, it reads the data blocks for the source inode and copies them to the destination inode's blocks, 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 data blocks by swapping them with that 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. OHSM Administrator is acknowledged of the completion of these event. See Figure 4 which illustrates the process of relocation.

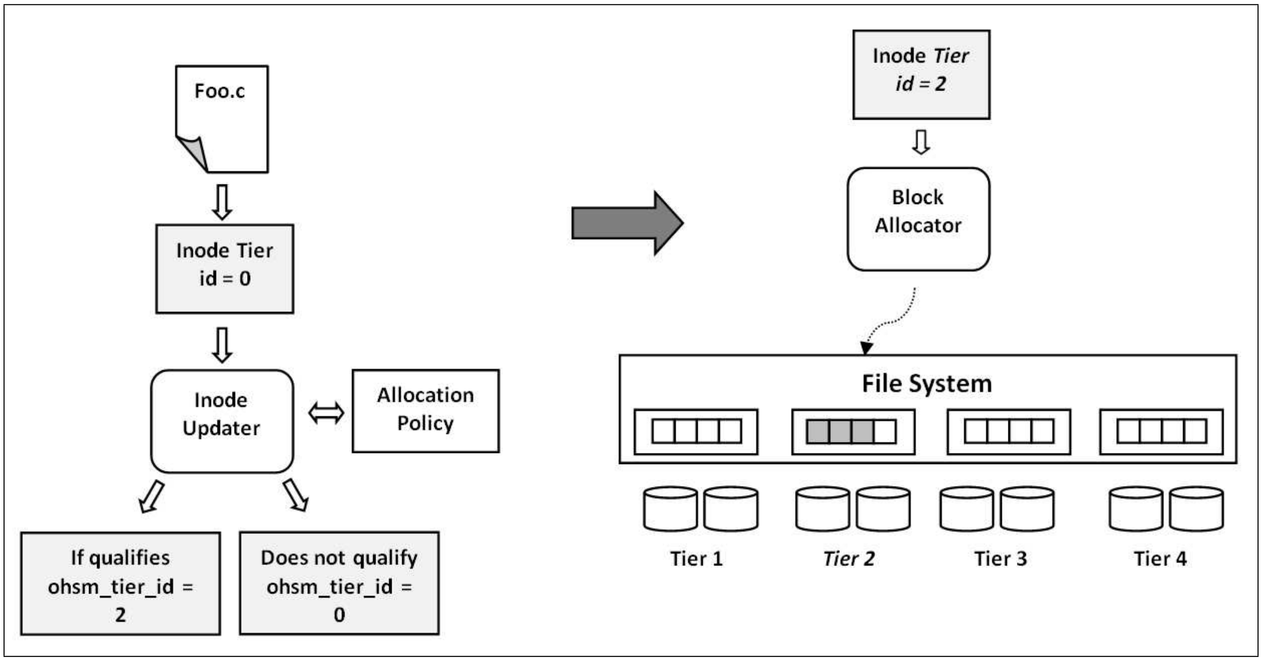

Figure 3: File placement and Block allocation mechanism

6 Issues and Concerns

During the course of OHSM's prototype implementation which was done primarily for ext2/ext3 file system, the process revealed several issues. Some of the major ones include the following:

6.1 struct ext3_super_block

Currently both the tier-block group range map and the allocation policies reside in OHSM kernel drivers. This enforces a dependency of the file system on OHSM kernel driver. We ensure to handle this situation currently by starting the OHSM services before any local file system is mounted. It required some modification to the system startup script. Since, this information is per file system this information should ideally reside in the super block of the file system. OHSM still struggles to find an easy way out of this. Since, this table can be huge and number of policies can be quite high, keeping such information in the super block is undesirable.

6.2 struct ext3_inode

Allocation and relocation are the key components of OHSM and as the object on which OHSM operates is a file, it is very tightly coupled with the inode structure. So, it was required to make on-disk changes in the struct ext3_inode in order to support allocation and relocation. We added "ohsm_home_tid" which stores the tier ID assigned upon allocation of a file and "ohsm_dest_tid" which stores the tier ID assigned during relocation of a file. Furthermore, to support different criteria of relocation like File Access Temp (FAT) and File Input Output Temp (FIOT) respective fields are to be added to the inode structure of the file system. These changes make the compact inode structure slightly bulky. These changes might also disturb any existing file system partitions present on the current system. OHSM plans to use extended attributes in order to avoid this problem and currently lacks a concrete way to handle this.

6.3 Exporting internal functions

The complete working of Online Hierarchical Storage Manager requires support from the file system on which it operates. In order to provide support to OHSM a few static functions residing inside the file system are required to be exported. This compromises the integrity of the file system code. For recent file systems like ext4 the required functionality (EXT4_IOC_MOVE_VICTIM) is soon going to be present which will help OHSM to not violate such integrity issues in the future. We are

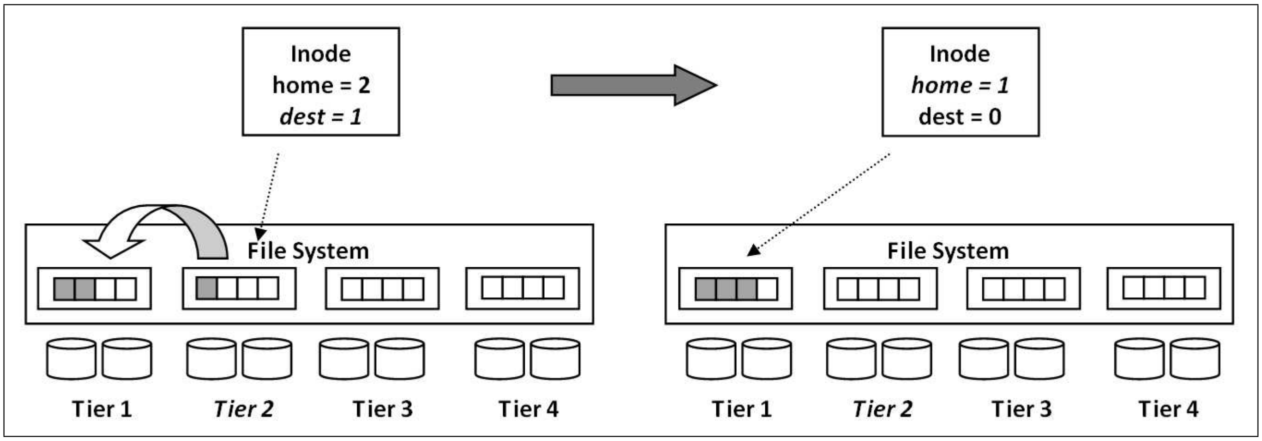

Figure 4: File relocation mechanism

trying to make OHSM completely functional with the ext4 file system during the writing of this paper. We are also monitoring and reviewing the implementation of the patches from Akira Fujita for restricted block allocation.

6.4 Crash during relocation

Currently OHSM uses a temporary inode in order to relocate an inode's data blocks from one tier to another. In case, if the system crashes during this relocation, there can be a chance of data loss. Currently we only release the original blocks after the complete relocation is completed. This reserves some space for the time during which relocation is going on. OHSM still needs a better way to handle this. Journaling may be required to overcome this problem.

6.5 Inode lock contention

During the process of relocation the inode is locked until all the data blocks attached with the inode are successfully relocated to a new destination tier. The time taken during relocation is file size dependent. This time would be more for large files, so any I/O pending on it will have to wait for that period of time and the requests may even time out. We are in the process of dividing this whole relocation into chunks of 64K in order to reduce this lock contention period.

7 One step Further

OHSM looks forward to a list of enhancements in-order to make it complete and stable. Here are a couple of them to start with:

7.1 User space implementation

The most desirable aspect with OHSM is to move as much as possible of its implementation into user space. This helps us remove the kernel components and other dependencies of the file system. It will also help maintain the source code integrity for the file system. Such an implementation would surely require a lot of support from the file system. Going ahead with an ioctl based interface would be one of the best options. Eventually the file system would need to support the OHSM based file creation and ranged block allocation. To achieve this without breaking the integrity and consistency of the code is a big challenge.

7.2 Automatic Relocation Engine

Currently relocation is a triggered event which in most of the server based environments will not be a pleasant experience for the administrators. Going a step further and designing an automatic relocation engine would be one of the most fascinating features OHSM can offer. The most important challenge in designing such an engine would be to derive the heuristics which would drive such an engine. A very frequent invocation to relocation can damage the file systems performance to a great extent. Also, a long interval between relocations can affect the storage efficiency adversely. So, we need an intelligent mechanism which could be based on the I/O and activities on the file system. Using FAA and FMA as criteria can impose hard restrictions with their values being constant, which might not always yield optimum results. FIOT and FAT can be the most efficient candidate as they are softer and can be based truly on the file activities in real time. OHSM is still looking forward to good heuristics and measures which would provide an optimum and efficient methods for making relocation a dynamic event.

7.3 Optimize mdraid Support

OHSM uses a new block allocation strategy for the file system and also has the underlying device topology. If OHSM is used over mdraid array, OHSM's block allocation strategies can be further optimized to leverage the underlying device layout. This may enhance the I/O speed over the devices in the mdraid array.

8 Conclusion

Online Hierarchical Storage Manager for GNU/Linux creates a platform and opens up various opportunities for further work in the area of Hierarchical storage for a Linux based environment. OHSM sets up the basic infrastructure where we can think of systematically merging traditional and SSD based storage devices to reduce the overall cost of the system administration and also attaining a degree of storage efficiency. Moreover due to the support for policy based migration of data, the administrative cost of managing data also reduces. The idea is to effectively reduce the cost of storage administration and at the same time keep the system efficient and consistent. OHSM with some changes to the file systems file creation and block allocation algorithms can achieve its goal of implementing a complete open source storage software solution.

9 Acknowledgment

We would like to sincerely thank all the people who helped us in this project, especially Greg Freemyer and Manish Katiyar for providing us their valuable time and support on various technical and design issues. Also we would like to thank Bharti Alatgi and Uma Nagaraj for their keen interest in OHSM from the early beginning and motivating us in the overall course of development.

10 References

1 *Retrieving Multimedia Objects From Hierarchical Storage Systems,* Eighteenth IEEE Symposium on Mass Storage Systems and Technologies, 2001.

2 *Planned Extensions to the Linux Ext2Ext3 Filesystem,* Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference.

3 *On Configuring Hierarchical Storage Structures (1998)*, Ali Esmail Dashti, Shahram Gh. In Proceedings of the Joint NASA/IEEE Mass Storage Conference.

4 *Ensuring Performance in Activity-Based File Relocation,* Wu, J.C. Bo Hong Brandt, S.A. Dept. of Comput. Sci., California Univ., Santa Cruz, CA.

5 *DHIS: discriminating hierarchical storage,* Proceedings of SYSTOR 2009: The Israeli Experimental Systems.

Proceedings of the
Linux Symposium

July 13th–17th, 2009
Montreal, Quebec
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium,*
Thin Lines Mountaineering

Programme Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium,*
Thin Lines Mountaineering
James Bottomley, *Novell*
Bdale Garbee, *HP*
Dave Jones, *Red Hat*
Dirk Hohndel, *Intel*
Gerrit Huizenga, *IBM*
Alasdair Kergon, *Red Hat*
Matthew Wilson, *rPath*

Proceedings Committee

Robyn Bergeron
Chris Dukes, *workfrog.com*
Jonas Fonseca
John ‘Warthog9’ Hawley

With thanks to

John W. Lockhart, *Red Hat*