HNType : a diverse trace and migration mechanism in the block based Hierarchical Storage System named HazelNut

Ran Du, Qiulan Huang, Yaodong Cheng, Gang Chen
Institute of High Energy Physics, Chinese Academy of Sciences, Beijing, China
E-mail: {duran, huangql, chyd, cheng}@ihep.ac.cn

Abstract. HazelNut is a block based Hierarchical Storage System, in which logical data blocks are migrated among storage tiers to achieve better I/O performance. In order to choose migrated blocks, data block I/O process is traced to collect enough information for migration algorithms. There are many ways to trace I/O process and implement block migration. However, how to choose trace metrics and design block migration algorithm is a big problem for system designers. To address this problem, a diverse trace and migration mechanism HNType is proposed. HNType consists two parts, one is a diverse trace mechanism named HNType-t, and the other is a diverse migration mechanism named HNType-s. HNType-t abstracts four base elements and trace operation interfaces based on VFS design concept, which makes it feasible to customize specific trace metrics and trace operations; HNType-s presents three ways of data migration, each way of migration can use customized migration algorithms according to predefine prototypes. Based on HNType, A series of tests are conducted about how block migration is affected by different trace metrics, and three conclusions are drawn according to the experiment results. First, trace metrics of access times and access sectors are not linearly correlated, as a result, these two metrics bring different migration result; Second, I/O completion time used as trace metrics is able to improve sequential I/O by at least 10%; Third, access times used as metrics have a tendency of more migrations upwards.

1. Introduction
High Energy Physics(HEP) experiments generate massive data every year. To save these valuable data, a Hierarchical Storage Management(HSM) system is used to provide cost-effective storage service. In our institute, Castor and Lustre filesystems are adopted as offline and online storage infrastructures. With the new types of storage devices(e.g. SSDs) emerging, its possible to improve system I/O performance by incorporating new storage devices. With this background, an experiment block-based HSM system named HazelNut is developed. To decide which blocks should be migrated, data block I/O process is traced to collect enough information for migration algorithms. There are many ways to trace I/O process and implement block migration. However, how to choose trace metrics and design block migration algorithm is a big problem for system designers. To address this problem, a diverse trace and migration mechanism HNType is proposed. HNType consists two parts, one is a diverse trace mechanism named HNType-t, and the other is a diverse migration mechanism named HNType-s. With HNType, system designers can choose appropriate trace metrics according to the test results.
2. Related works
HazelNut is an HSM system in which different types of storage are organized as storage tiers. For such kind of HSM systems, most of them, e.g. Hystor[1], EDT[2], HP AutoRAID[3], have a single migration algorithm running. Comparing to this, HNType, the diverse trace and migration mechanism of HazelNut, allows multiple migration algorithms co-existed. Besides, HSM systems like TiStor[4], IBM STEPS[5], EMC Symmetrix DMX[6], IBM Tivoli Storage Manager[7], EMC DiskXtender[8] implement customized migration strategies based on files, while HNType is implemented based on blocks. OHSM[9] supplies a similar mechanism with HNType but with different implementation. In OHSM, Migration strategies are stored in XML files, while HNType supplies different migration strategies in the form of loadable kernel modules; Filesystem code modification is needed in OHSM, but HNType is implemented under the Filesystem layer, no code modification is needed.

3. HNType
3.1. HNType in HazelNut
HazelNut is a block-based Hierarchical Storage System, in which different types of storage devices are organized as different storage tiers. To place data on the appropriate tier, it is required to trace I/O of data blocks and migrate specific data blocks according to the collected I/O information. However, I/O requirements from upper layers are diverse and complicated. What metrics should be traced and what migration algorithms should be adopted are the problems faced by system designers. To solve these problems, a mechanism is needed. With this mechanism, it is convenient to set I/O trace metrics and develop migration algorithms, so that system designer are able to configure trace metrics and modify migration algorithms according to the test results.

Such a trace and migration mechanism, HNType, is proposed based on the mentioned requirements. HNType is consisted of two parts, one is a diverse trace mechanism named HNType-t, the other is a diverse migration mechanism named HNType-s. HNType-t is designed based on the Virtual Filesystem Switch(VFS). In HNType-t, the concept of Trace and Migration Type(TMT) is proposed, trace interfaces of TMT elements are also designed. On the other hand, three types of migration frames are designed by HNType-s, which are Long Term Migration Frame(LTMF), Short Term Migration UP Frame(STMUF) and Emergency Migration Down Frame(EMDF). Besides, migration interfaces are also designed by HNType-s. With the specific implementation of trace interfaces in HNType-t, and migration interfaces of HNType-s, a specific TMT is implemented. After registration in HazelNut, this TMT could be used to trace data blocks I/O and migrate appropriate data blocks.

From the view of code implementation, HNType is consisted with Linux kernel modules ended with .ko, as in the Figure 1. In Figure 1, def_hazelnut.ko, hit_hazelnut.ko, sec_hazelnut.ko and hitcp_hazelnut.ko, each of these four kernel modules stands for a specific implementation of TMT, other TMTs can be implemented and added to system by system designers.

HNType is a part of HazelNut system which is implemented in the generic block layer of Linux. HazelNut consists of three layers, HNType is the middle layer. The bottom layer is implemented as a kernel module named hazel_root.ko, the upper layer is implemented as a kernel module named hazel_main.ko. hazel_root.ko is used to construct the storage tiers, execute migration operations defined by a specific TMT of HNType, and provide sysfs interfaces to users; hazel_main.ko is used to provide ioctl interface to users and execute I/O requests from the Linux upper layers.

3.2. HNType-t
HNType-t is proposed based on the design concept of Linux Virtual Filesystem Switch(VFS). VFS is the public filesystem interface to Linux storage upper layers. To support multiple
filesystems, VFS extracts common parts of different filesystems, abstract these common parts as four basic elements (superblock, inode, dentry and vfsmount), and defines objects and operations of the elements. When a filesystem I/O request passed to VFS, VFS calls the recall functions implemented by a specific filesystem, the I/O request will be executed by the specific filesystem at the end. As long as a filesystem implements the recall function interfaces defined by VFS, it will be ready to use in Linux after registration.

Similarly, to support diverse ways of trace and migration, four basic elements, shell, branch, limb and trunk, are extracted by HNType-t. Shell element is the abstraction of data blocks; branch element, limb element and trunk element are separately the abstraction of Branch Devices, Limb Devices and Branch Devices. A Branch Device is a physical storage device; a Limb Device contains a number of Branch Devices, which stands for a storage layer; a Trunk Device is consisted with a number of Limb Devices, which stands for a virtual hybrid storage device. Like VFS, HNType-t also defines objects and operations of each element, as in Figure 2(a). For example, the objects of the shell element, is shell_m and shell_d; the operations of the shell element is the shell_ops, similar to the objects and operations of the other three elements. Figure 2(b)(c)(d)(e) list the interfaces of element operations in more details.

3.3. HNType-s
HNType-s is the other part of HNType and supports three migration frames: Long Term Migration Frame (LTMF), Short Term Migration Up Frame (STMUF), and Emergency Migration Down Frame (EMDF).

As the name implies, LTMF is usually with a long migration term (normally several hours), and suitable for fairly stable I/O requests. When the migration phase is due to arrive after a long term, LTMF will traverse the traced metrics of all data blocks, and migrate the data blocks fulfilled with migration conditions to a proper Limb Device. The flow chart of LTMF is shown in Figure 3.

On the other hand, STMUF is normally with a short migration term (usually several minutes), and applied to bursty I/O request. The migration phase of STMUF is intercepted by a short term, besides, the migration phase can be triggered if the number of to-be-migrated data blocks reached a threshold. Compared with LTMF, STMUF requires two extra lists to filter to-be-
Figure 2. Four elements of HNType-t

Figure 3. Migration flow chart of LTMF.
migrated data blocks. One list is to save the serial number of data blocks with sequential access pattern, the other list is to save the serial number of data blocks with random access pattern. When the migration phase is triggered, STMUF will migrate the data blocks recorded in the sequential list and random list, and place these data blocks to a proper Limb Device chosen by the migration algorithms. The flow chart of STMUF is shown in Figure 4.

Different with LTMF and STMUF, migration phase of EMDF is triggered by the space threshold, not time threshold. EMDF migration normally comes under three circumstances. First, when data is written to a blank data block on a Limb Device but the Limb Device reaches its space threshold; Second, when LDMF is going to migrate a data block to a Limb Device while the destination Limb Device reaches the space threshold; Third, when STMUF is migrating a data block to a proper Limb Device but the Limb Device reaches a space threshold. The same as STMUF, EMDF requires two lists to save the serial number of to-be-migrated data blocks, one list is for the data blocks with the sequential access pattern, the other one is for the data blocks with the random access pattern. Different with the STMUF, the sequential list of EMDF has a higher priority than the random list of EMDF, which is based on the following consideration. First, there is small gap of sequential I/O performance between different block storage devices, so that the I/O performance will not degrade much even if the data blocks migrate down to a lower performance device; Second, the data blocks with sequential access pattern has more time locality, which means that these data blocks has less possibility to be requested if not been accessed for a long time. When the migration phase comes, EMDF will migrate the data blocks recorded in the sequential list and random list, and put these data blocks to a proper lower performance Limb Device. The flow chart of EMDF is similar to the flow chart of STMUF, as the limited pages, more details will not be given in the paper.
4. Test result

Data block I/O trace interfaces are defined by HNType-t, data block migration frames are designed by HNType-s. When the interfaces of HNType-t and HNType-s are implemented by the system designer, a specific TMT is implemented to use.

To prove HNType is valid, we implements three TMTs which are HIT, SEC and HITCP. HIT is using access times as trace metrics; SEC uses access sectors as trace metrics; while the HITCP is using access times and I/O completion duration as trace metrics. All the three TMTs uses the LTMF as the migration frame, and adopts a migration algorithm implemented by Mark Ruijter in the btier system. The btier migration algorithm is a linear migration algorithm, the basic idea is described as the following pseudo codes.

```plaintext
for each data block {
    if(data_block_metrix >= avg_limb_device_metrix + factor1 &&
        data_block_metrix >= upper_avg_limb_device_metrix - factor2 ){
        migrate_up();
    }
    if(data_block_trace_time > trace_period &&
        data_block_metrix < avg_limb_device_metrix - factor1){
        migrate_down();
    }
}
```

Besides of the validation of HNType, an experiment is done to check if there is any migration difference if access times and access sectors are used as trace metrics separately, since these two metrics can be seen as the requirement from upper Linux storage layers. The result is shown in Figure 5.

As in Figure 5, migration up means that the migration is from the lower performance Limb Device to the higher performance Limb Device; while migration down means the reversed migration direction. Figure 5 makes statistics of migrated data block on their position before and after migration. It is clear that the migration up results between HIT and SEC are different; while the migration down results look similar between HIT and SEC, but the number of migrated data blocks are different.

As mentioned in the section 3, HNType is designed to choose trace metrics and develop migration algorithms for system designers. The second experiment of this section gives an example on how to study trace metrics and migration algorithms with HNType. In the second experiment, three TMTs, HIT, SEC, and HITCP are used to display the different migration results in two aspects, storage space utilization and average I/O performance. Test result is shown in Figure 6.

Figure 6 demonstrates that HITCP has better sequential I/O performance than the other two TMTs at least by 10%, also HITCP has more stable I/O performance. This is because that HITCP uses I/O completion duration as trace metrics, and completion time implies the fulfillment capability of storage devices. As for the system
designers, the test result means that its better to choose I/O completion duration as trace metrics. Besides, storage space utilization is different between HIT, SEC and HITCP. Comparing with the other two TMTs, HIT tends to use more space of higher performance device, but doesn't perform better than HITCP. This fact implies that using single trace metrics may not be a good idea for system designers.

Figure 6 also displays the defects of the tested tier migration algorithm: this migration algorithm doesn’t have good random I/O performance. A possible improvement proposal is to add new migration operations using the STMUF, so that more data blocks with random access pattern are migrated to higher performance storage tier, and random I/O performance will be improved. The process of analyzing test results as shown in Figure 6, is the way to study better migration algorithms for system designers.

5. Future work
HNType-s provides three migration frames: LTMF, STMUF and EMDF. Each migration frame has a switch to enable/disable the migration thread. Currently, threads of each migration frame are mutualized, which means that only one migration thread is running at one time. For example, both LTMF and STMUF are enabled, if the migration thread of LTMF is due to wake up, but if the migration thread of STMUF is running, migration thread of LTMF will sleep for a short period until the migration thread of STMUF has its work done. Later, a more complicated control mechanism will be designed to allow multiple threads of different migration frames running at the same time. This means more work on the metadata consistency.

6. Conclusions
HNType is a diverse trace and migration mechanism in a Hierarchical Storage System named HazelNut. HNType-t and HNType-s are two parts of HNType. HNType-t provides trace interfaces, while HNType-s provides
migration interfaces. With HNType-t and HNType-s, system designers can choose proper trace metrics and improve migration algorithms to get better I/O performance.

7. Acknowledgments
We wish to thank all the colleagues who contributed to the HazelNut system. This work is funded in part by The Chinese National Natural Science Foundation Contract No. 11305192, 11575223, 11375221, 11405195 and 11205177, and the Twelfth Five-year Plan of Chinese Academy of Sciences Contract No. Y2293C00S5.

References
[1] Chen F, Koufaty D A, and Zhang X. Hystor: making the best use of solid state drives in high performance storage systems. In Proceedings of the international conference on Supercomputing. pages 22–32. ACM, 2011.
[2] Guerra J, Pucha H, Glider J S, Belluomini W, and Rangaswami R. Cost effective storage using extent based dynamic tiering. In FAST, volume 11, pages 20–20, 2011.
[3] Wilkes J, Golding R, Staelin C, and Sullivan T. The hp autoraid hierarchical storage system. ACM Transactions on Computer Systems (TOCS), 14(1):108–136, 1996.
[4] Luo S, Zhang J, Zhou Y and Cheng S. TiStor:a Tiered Storage System Based on Policy Engine[J]. Journal of Chinese Computer Systems, 2015(8):1695-1699.
[5] Verma A, Sharma U, Rubas J, Pease D, Kaplan M, Jain R, Devarakonda M, and Beigi M. An architecture for lifecycle management in very large file systems. In Mass Storage Systems and Technologies, 2005. Proceedings. 22nd IEEE/13th NASA Goddard Conference on, pages 160–168. IEEE, 2005.
[6] http://www.emc.com/products/detail/hardware/symmetrix-unix-linux.htm, 2012.
[7] Tretau R, Andal A, Battaglia R, Edwards D, and Speh H. IBM Tivoli Storage Management Concepts. IBM Corporation, International Technical Support Organization, 2003.
[8] http://china.emc.com/products/detail/software/diskxtender-unix-linux.htm, 2012.
[9] Sinha S K, Agrawal R B, Agarwal V, Vashist R, Sharma R K, and Hendre S. Online hierarchical storage manager. In Linux Symposium, page 263. Citeseer, 2009.