Implementing a Hierarchical Storage Management system in a large-scale Lustre and HPSS environment

Brett Bode, Michelle Butler, Jim Glasgow, Sean Stevens
National Center for Supercomputing Applications
University of Illinois
1205 W. Clark St. Urbana, IL 61801
{brett, mbutler, glasgow, sstevens}@illinois.edu

Nathan Schumann, Frank Zago
Cray Inc.
901 Fifth Avenue, Suite 1000
Seattle, WA 98164
{nds, fzago}@cray.com

Abstract— **HSM functionality has been available with Lustre for several releases and is an important aspect for HPC systems to provide data protection, space savings, and cost efficiencies, and is especially important to the NCSA Blue Waters system. Very few operational HPC centers have deployed HSM with Lustre, and even fewer at the scale of Blue Waters.**

This paper will describe the goals for HSM in general and detail the goals for Blue Waters. The architecture in place for Blue Waters, the collaboration with Cray, priorities for production and existing challenges will be detailed as well in the paper.

Keywords-component; formatting; style; styling; insert (key words)

I. INTRODUCTION

Current practices in capacity management of shared, high-performance file systems for HPC almost universally employ an age based purge mechanism to maintain file system usage at a level commensurate with the capability of the overall system. This practice has been the standard for many years now, but with the continued development of parallel file systems, in particular Lustre, new opportunities are emerging to use multiple tiers of storage to obtain an optimal mix of storage capacity and performance within a single file system environment.

Relying on file purging mechanisms tends to result in poor use of the capacity of these dynamic file systems. This often requires over-provisioning of file systems to allow continuous operation through the peaks and troughs created by the purge process. Managing storage solely by the date of last access can be extremely inefficient for both the system and the science team. On Blue Waters NCSA requires science teams to manually move data between the online file system and the HPSS [1] based tape environment in order to preserve data for longer than the purge policy (currently 30 days based on last access) taking time away from their research. By enabling automated capacity management of the "scratch" filesystem using Lustre's HSM [2] functionality NCSA intends to address both of these concerns.

In general, HSM functionality provides a single namespace view to multiple tiers of storage along with manual and automated data movement. The purpose of adding the complications of an HSM is to separately optimize the capacity and performance of the storage environment. The top tier can be optimized for performance while lower tiers can be optimized for capacity. Files can be resident on one or all tiers and policy based data movement moves data to the larger, slower tiers as the data ages. At some point the data may be removed from the top tier to free space on the high-performance tier for active data [3]. In order to operate on the data, the data must be resident in the top tier of storage so tools must also be provided to allow users to ensure that data is staged to the top-tier of storage before it is needed in a computational job. If the data is not on the top tier of storage an open will block and issue a retrieval request. The time the operation is blocked can be quite significant for high-latency back-ends and large files.

There have been previous HSM implementations including Tivoli Spectrum Scale (aka GPFS) [4] and GLUFS [5] are just two of a very long list of implementations of this idea of connecting the file system to an off-system tape system. While back-ends can be any type of storage including another file system, or an object store to maximize the storage per dollar the use of a tape based back-end is desired. Tape based storage poses the most dramatic difference in performance, particularly the initial latency of file access and thus is the most important environment to ensure that data is on the front-end when needed. In many of the past disk/tape HSM implementations the goal was for the spinning disk storage to be a modest sized staging area (cache) for the tape environment, often with no option provided for users to modify or utilize the data for computation and analysis within the environment. In the current generation of HSM environments the high-speed storage is intended as a primary storage component within the compute environment.

Controls for ensuring fair use of the file system evolve into quota policies and enforcement. Users can keep as much (or as little) data online, for as long as they require, as long as they are within their quotas. The automated policies must balance the goal of moving data to the back-end quickly with the large difference in performance between the storage tiers that implies that the back-end could easily be swamped with temporary data written to the front-end. Once data is copied to the back-end it is eligible for release from the front-end. The automated release policy can release data based on many

parameters, but is driven by the goal of keeping the primary file system usage at or below the desired target level.

Cray's Connector software integrates the standard Lustre HSM agent and copytool into a service that provides greater flexibility, performance, and resiliency for large data systems. A single manager can register multiple back-ends and provides controls for governing the IO workload across multiple data servers. A full plugin API allows Cray's Connector to be easily extended to support new back-end storage systems.

Of course, attempting to create a new HPC storage paradigm for a system the scale of Blue Waters is not without challenges. Issues of performance matching a 21PB, 1 TB/sec. file system, capable of producing 10's of millions of files per day, to storage systems designed for cost efficiency requires some tradeoffs. In particular, tape systems such as NCSA's HPSS environment are poor at handling large quantities of small files requiring NCSA to consider alternatives for the small file problem. Another challenge is quota management since there is no unified quota system for the single namespace in an HSM environment. Finally, the ability of HSM to recall a file automatically upon open provides convenience, but if that file must be recalled from tape the initial latency can be quite substantial. As a result, integration with the system batch scheduler to ensure that all required data is staged to the high-speed storage prior a job starting is recommended. Solutions to these challenges will be discussed along with details of the NCSA implementation.

II. NCSA GOALS

NCSA's goal for HSM is to change the operational model for the primary high-speed file system on Blue Waters. The existing model is a traditional "scratch" space where projects have a generous quota (the overall file system is significantly over allocated), but data is subject to a limited lifetime enforced by a purge policy. In the case of Blue Waters data is subject to being purged after 30 days without activity. To avoid required data being purged the science teams must manually copy data off of the scratch space to other file systems (if small), to the Blue Waters nearline tape system or to external sites. Then when the data is needed the team must manually copy it back to scratch.

Under the HSM model the file system provides a single namespace view of the high-speed disk and tape environments. Projects are subject to quotas for both disk and tape subsystems, but data is no longer automatically purged. Instead policies migrate data from disk to tape and then release the disk copy based on age and level of disk usage. The manual data movement steps to copy data in and out of the environment are reduced, but teams must add a data staging step to the beginning of their workflows to ensure that needed data is resident on the high-speed storage before the primary computational job(s) need it.

III. NCSA IMPLEMENTATION

NCSA has worked with Cray to develop a plugin to Cray's Connector software for Lustre [6] HSM enabling data transfers between Lustre and NCSA's large HPSS tape system. The data mover plugin, dubbed HTAP, allows a file system's capacity to be managed through the use of Lustre HSM and Robinhood policies [7]. As with other HSM systems under the Lustre HSM model files copied to a lower storage tier can have their data blocks released from the top tier when the space is needed in the top storage tier. Released files appear no different to 'standard' file system utilities, providing the user with a complete view of their files without regard to where the actual data blocks reside. Data movement (copying) or migration is "under the covers". Released files are automatically retrieved from the back end on demand (ie when a file is opened). This process greatly reduces the load on scientists to manage their data movement to and from the nearline systems while computing on Blue Waters, but requires them to do their own deletes instead of relying on the purge process.

When creating the data management policies, the capabilities of each storage tier along with the expected usage by science teams must be taken into account. As with most lower storage tiers in an HSM environment the Blue Waters nearline tape system cannot possibly keep up with 25,000 file creates per second or the 1 TB/s of IO that the primary disk environment can sustain. Thus, data in the disk environment must age to the point where the churn is reduced to the level sustainable by the HPSS tape back-end. Based on an analysis of Robinhood data the initial target will be to copy data to the back-end after it is seven days old. At any point after the data is copied to the back-end (and is thus dual-resident) the data blocks on the front-end can be released triggered by the need to free space on the front-end.

Another consideration is the large number of small files in the file system. The HPSS tape subsystem is poor at handling small files and very large file counts so NCSA intends to direct small files (<16MB) to an attached disk cache instead of the HPSS subsystem. The small file copy is intended for disaster recovery only so the data blocks for small files would never be released from the top storage tier. It is estimated that a 1.5 PB disk environment is sufficient for this purpose.

In order to ensure that data is resident in the high-speed storage tier when needed by a computational job additional tools will need to be provided to migrate the data to the high-speed tier and mark it as "in-use" through the use of "hints". This will be part of phase 2 of the project to develop an interface for users to influence the automated policies through the use of a .hsmrc file or some similar mechanism.

IV. DATA MANAGEMENT POLICIES

The Robinhood Policy Engine will be used to initiate and control automated LHSM data movement. Robinhood utilizes a combination of full scans of the filesystem and then running as a Lustre changelog consumer in order to maintain

a reasonably real-time database of the current state of the filesystem. Once a full scan has been completed, the changelog reader along with HSM policies can be enabled. The policies will run the required Lustre HSM commands based on specified criteria. Changelogs will be generated as Lustre HSM commands are completed. These new changelog entries will keep the HSM state of the archived files in the Robinhood DB up to date.

As Robinhood ingests the filesystem information, files will be classified based on size (i.e. small files < 16MB) along with additional criteria. The file classes will ensure that files are archived with the most optimal settings in the appropriate back-end. Policy behavior can be effected by many different parameters. These could be size, create/modify/access times, user/group acls, custom file xattrs, file path, and many more. A policy can be triggered based on a variety of criteria; anything from just a scheduled timer or on different usage thresholds of the filesystem components (user, group, ost, pool, global usage levels).

The default archive policy will be a scheduled run that is triggered frequently to ensure the HSM processes do not get too congested. The initial archive of a file from scratch will have a default delay of several days. This delay is being created to avoid archiving short-lived temporary files. Since this will be implemented on a "scratch" filesystem archiving everything in real-time as written to the filesystem would cause many unneeded transfers to the other storage tiers and slow down the overall HSM process. Since the highest storage tier is provisioned for performance and the lower tier(s) for capacity it is very unlikely that it is possible to copy more than a small fraction of the data written to the front-end to the back-end. Indeed, there are days on Blue Waters when 9+PB of data is written to the existing scratch that would require almost two days to write to tape assuming all tape resources were dedicated to writing.

A release policy should be utilized to maintain filesystem usage at acceptable levels. As with any other policy this can be affected by many criteria. The default criteria would likely be some set amount of time since a file has been accessed/modified (in other words the files accessed longest ago would be released first). A more advanced option might be to release a user's oldest files as they approach some threshold of their quota. Additional end user customizable options could also be added through specific file extended attributes (xattrs). For example, an xattr could be designated to flag a file for immediate release once archived.

Another common policy that is available is a remove policy. This policy is a little different as it is for removal of files from the archive, that have already been removed from the primary filesystem. The availability of the same criteria as other policies along with a special "rm_time" attribute can make this a very useful policy. First, by adding a delay before deleting the archived (only) copy of a file it can provide an undelete option; through the use of the rbh-undelete command. It also allows better control over file removal, through batching and scheduling.

In production, our use of LHSM policies might differ some from that of a traditional HSM. The archive policy would be relatively straight forward; copying new/modified data to the archive after some delay (likely on the order of 1 week) to ensure we do not archive short-lived temporary data. The default release policy will most likely be triggered based on filesystem pressure. The primary objective would be to keep the filesystem below a determined set utilization level in order to maintain expected performance and fair use. As the filesystem reaches an upper threshold (such as 80% utilization), the release policy would be triggered to release beginning with the oldest and largest files first. Finally, a remove policy would be used to ensure files specifically removed from the filesystem also get removed from the archive backend. There would be some delay between a delete in Lustre and the archive removal in order to not overwhelm the archive backend and other processes.

V. Cray Connector Description

Cray's Connector has two primary components, the Connector Migration Manager (CMM) and Connector Migration Agents (CMAs). The CMM registers with the Lustre file system as a copy tool. It is then responsible for queuing and scheduling Lustre HSM requests that are received from the MDT across one or more CMAs. CMAs perform all data movement within the Connector, and are also responsible for removing archive files from back-end storage if requested. An instance of the Connector will have one CMM, and at least one CMA. There is not an inherent limit to the number of CMAs that can be placed, but there are practical limits such as the number of servers available. Each CMA can execute a configurable number of simultaneous requests.

In addition to queuing and scheduling requests, the CMM also provides for status reporting of ongoing and completed transfers, performance statistics, manipulation of requests such as pause, restart, or cancel, and reprioritization of requests. The CMM can split archive or restore requests over several CMAs for increased performance. Finally, the Connector also supports checksums for data integrity. This works on both non-striped, and striped files. For striped files, each stripe segment will have an associated checksum. Once a striped checksum transfer is completed the CMM will generate a checksum of the checksums for verification at retrieval or later.

As stated previously, the Connector can support multiple storage back-ends. Initially the Connector focused support shared POSIX file systems, such as Versity Storage Manager [8]. To support NCSA's requirement of archiving data to HPSS through the Lustre HSM interface, Cray and NCSA co-

designed the plugin API used by HTAP. This API would also make it possible to support storage back-ends such as object, cloud, or other archiving systems. Plugins also can make decisions on striping of files, rather than use the globally defined stripe configuration in the Connector. For example, NCSA has different storage classes defined in HPSS that dictate file striping. The HTAP plugin can use that information to stripe files across multiple CMAs, and have the starting CMA also complete the transfer as is required with HPSS.

Plugins have two constraints, namely that the plugin can be used for several different back-ends of the same type, and that plugin methods must be reentrant as the CMA is multithreaded. Plugins can operate in one of two modes. The first, and perhaps the simplest, is to let the CMA core perform all the I/O between Lustre and the storage back-end. In this mode, the plugin need only provide functions to open or remove a file, and provide a file descriptor. Alternatively, a plugin can take care of the file transfer itself, which is typically required for back-ends that operate at an object level rather than a file level. With HTAP, NCSA has chosen the later method in which HTAP is responsible for all I/O between Lustre and HPSS providing status updates to the CMA.

VI. HPSS Connector Plugin Implementaion

The Connector plugin developed at NCSA with Cray, for HPSS, moves data directly from a Lustre Client to HPSS (and back) using the HPSS client API. No third party or intermediate software is necessary and there are no additional dependencies beyond those required for the Connector, Lustre, and HPSS. Plugin configuration is done through the Connector configuration file (/etc/tas_connector.conf) and the standard HPSS methods (/var/hpss/etc/...).

HTAP (the Hpss TAS Agent Plugin) is written in "C" as a shared library bridging the Connector and HPSS Client APIs. When the CMA starts, it loads the plugin into its process space and calls the Connector API functions defined by HTAP. The CMM handles all of the transfer scheduling and related process management, freeing the plugin to concern itself with only the data movement. Thus, HTAP is relatively small and simple. It only has to provide basic read/write functions with a few HPSS specific functions like authentication and storing file metadata using HPSS User Defined Attributes (UDAs).

All files stored by HTAP are stored as a single "admin" user in HPSS. The user is defined in the Connector configuration file and is authenticated to HPSS using the "unix-keytab" authentication mechanism. As mentioned, the file's POSIX meta-data is stored in the file's HPSS UDA – including file owner and group IDs – but this is primarily intended for informational purposes. In the event the Lustre file becomes lost or damaged, it would be possible to get it back using the information stored in the UDA and the Robinhood database. A full file system restore cannot be done using the UDA data alone. The UDA data is not guaranteed to be accurate as no effort is made to synchronize it with the Lustre meta-data after the file is archived. For example, renaming or moving a file will not be reflected in the HPSS UDA. This is functionality that could be added with the data available in the Lustre changelog but it was deemed too resource intensive for the very large and active BW scratch file system against the meta-data performance of the HPSS storage system.

Files are stored in the HPSS namespace using a fixed directory tree and a filename generated by the Connector. Several concerns led us to forgo the use of both the Lustre FID and Lustre path names in the HPSS namespace. The possibility for FID's to change during file migration operations meant that they would require extra code to track and update. Likewise, the dynamic nature of file names in user space would require constant synchronization and could become a potential performance issue for the HPSS meta-data server. The Connector generates a quasi-random path and UUID style file name using a repeatable algorithm. This mechanism provides a nicely balanced directory structure and limits, as much as possible, directory access hot-spots. It also eliminates path length and character set problems altogether. Those issues are all left with Lustre.

On an archive operation, the plugin passes the Lustre file size to HPSS to allow for size based class of service (COS) selection. The data is then transferred using the stripe size of the selected COS. On completion, the file is check summed and, on success, the plugin stores the checksum result along with other file metadata (uid, gid, file permissions, etc.) into the file's UDA structure. If any discrepancies occur, the transfer will be automatically retried. During a restore, the operation is similar except that the checksum is read from the HPSS UDA and compared to the sum computed on the newly written Lustre file.

A. Future Development

Several important pieces are still left for future development. The largest being support for HPSS quotas. While the quota implementation mechanics are reasonably straight forward, the harder part is how to handle user notification or alerting. The majority of archive operations will be asynchronous with user activity, requiring an out-of-band messaging system to be introduced. This brings in concerns about message delivery, timeliness, and throttling that all need to be addressed if the quota system is to be reliable enough for production level use.

Next on the list are extensions to the plugin's COS selection mechanisms. The current size based selection works well for us but there are cases where it can be valuable to make selections based on other criteria, such as by project, or for creating an exportable data set. This is currently envisioned as a file extended attribute that the plugin would use as an override to the default behavior. The Connector uses file xattrs for storing information as well so this fits in with existing functions that the Connector provides to the plugin. The policy engine could set attributes during policy executions. It would also be possible to allow users to set hints on their files that

could be translated into other actions by the plugin -- perhaps even for advanced functionality such as file aggregation. These user provided hints could also be used as criteria for the policy engine; possibly to set a file for immediate archival/release regardless of age, or to prevent archival of a file that is associated with an active compute job.

VII. A Word About Backup

Care must be taken when viewing an HSM system as a backup solution. By design the base concept of an HSM provides a single namespace view of the multiple tiers of the storage environment. Thus, when a file is changed or deleted that is quickly reflected in all tiers of the environment. In addition, the potentially large delay in copying data to the back-end (targeted at seven days) is much larger than the typical frequency for a “backup” solution. The HPSS plugin, as currently written, stores files into the archive exactly as presented from Lustre via the Connector Manager. When a file is modified (its contents are changed) in the Lustre file system this will trigger a new "archive" event for that file. The HPSS plugin does not attempt to detect previously written files nor does it enact any kind of versioning. It would be possible to extend the plugin to provide these functions but it would also require the development of significant new functionality beyond the plugin and the Connector. From a high level, such a system would have to:

- Monitor the Lustre change logs for file changes e.g. chown, chmod, chgrp, etc.
- Be capable of storing soft and hard link information
- Manage file data within HPSS to trim file versions that have "aged out"
- Enable the restore of a specific file versions to the existing Lustre FID and setting all appropriate attributes on the file

Of course, none of this enables any kind of "bare-metal" restore for Lustre. To affect a restore of a file system loss due to MDS data loss, for example, would still require that the MDS itself be brought back online with some version of its database intact before any files could be restored from HPSS. Currently this is a major challenge as tools are not available to snapshot the MDS or Robinhood databases. It is certainly possible that the Robinhood database could be utilized to rebuild the MDS, but significant additional development is needed to make that possible.

VIII. Test Environment

The test system is composed of a 115TB Lustre file system running on Cray's Sonnexion 3000 appliance with the Lustre client and Connector HSM agent running on a Dell R730 server. The Lustre client system also runs the Robinhood policy engine which was used to drive the HSM policies for all our testing. For performance reasons, the Robinhood database server runs on an independent Dell R730 server with SSD storage. All HSM storage utilized the existing Blue Waters Nearline storage system running HPSS 7.4.3.

IX. Results

Testing of the Connector software on Lustre 2.5 using the HPSS plugin was performed using an existing user file system from the Blue Waters test system, "JYC". The file system contained a large variety of file data and usage patterns. The client node was configured to run four(4) transfer agents (CMA) allowing file striping up to the maximum stripe width used by our HPSS classes of service. Using the Robinhood policy engine and change log reader we have executed multiple series of tests against the Connector. Early results identified that the Lustre setting "hsm.max_requests" was configured too low. This was increased from three(3) to six(6) and during further tests we observed transfer rates around 900 files/min with a sustained, aggregate transfer rate of around 900MB/s on our modestly configured client. All data transfer operations were conducted using the built-in checksum capability of the Connector. The overall data rates are quite good and near the useful bandwidth of the 10Gb link on the Lustre client. The file operation rates are lower than anticipated but it is not clear at this time if the bottleneck is with Lustre/HSM, policy execution, or HPSS meta-data performance. We consider these numbers to be preliminary and much more testing remains to be done. Our expectation is that further scaling of system resources (additional Lustre client/Connector Agent nodes and dedicated Robinhood changelog and policy engine resources) will provide continued performance improvements up to the capability of either the Lustre file system or HPSS storage system.

X. Conclusions

The NCSA HTAP and Cray TAS software are working with the standard Lustre HSM enabling automated data movement between the Lustre disk environment and the HPSS tape back-end. The performance numbers while reasonable are preliminary due to only using one data mover in a small test system. Scaling the solution to a larger number of data movers, and to larger/fuller file systems are next on the list to do. Many other studies are needed before production such as quota and queue management and the latency for files that are on tape when operating in a busy environment.

Acknowledgment

This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] High Performance Storage System http://www.hpss-collaboration.org/

[2] Hierarchical Storage Manager https://en.wikipedia.org/wiki/Hierarchical_storage_management

[3] D. Reine and M. Kahn "In search of the long-term archiving solution-tape delivers significant TCO advantage over disk" http://www.clipper.com/research/TCG2010054.pdf

[4] "Tivoli Storage Manager for Space Management" http://www-03.ibm.com/software/products/en/tivostormanaforspacmana

[5] I. Koltsidas *et al*., "Seamlessly integrating disk and tape in a multi-tiered distributed file system," *2015 IEEE 31st International Conference on Data Engineering*, Seoul, 2015, pp. 1328-1339. doi: 10.1109/ICDE.2015.7113380

[6] Lustre - http://lustre.org/

[7] Robinhood from CEA: https://github.com/cea-hpc/robinhood/wiki

[8] Versity Storage Manager http://www.versity.com/product