Campaign Storage

Peter J. Braam
Campaign Storage, LLC
Nederland, CO, USA
Email: peter.braam@gmail.com

Dave Bonnie
Technical Lead - Campaign Storage
Los Alamos National Laboratory
Los Alamos, NM, USA
Email: dbonnie@lanl.gov

***Abstract*—Campaign Storage offers radically new approaches to data management tasks through a modular implementation and novel organization of storage features enabling management of trillions of files and exabytes of data. Challenges in the integration of commodity object stores are overcome, and a flexible indexing system for histogram style search is introduced. Campaign Storage raises questions about interfaces to govern layout, clustering, bulk movement and to control advanced data management. Its architectural choices led to an unusually short roadmap before being used in production at a major scientific computation facility.**

***Index Terms*—file system, archiving, hierarchical storage management, search, data management interfaces, HSM.**

I. Introduction

Campaign Storage [1] provides a colder storage tier for very large volumes of data - trillions of files and exabytes of data. It offers scalability in capacity and scalable performance for fast movement of data between the Campaign Storage system and high performance storage tiers such as burst buffers and parallel file systems used by compute clusters. Novel histogram style search is integrated and allows policy-based data management such as archiving. It offers snapshots and replicas to provide data preservation and recoverability and includes migration to cloud storage.

Campaign Storage offers a file system interface and is layered generically on existing distributed POSIX file systems such as GPFS or Lustre to handle metadata and it is layered on object stores to store file data. This allows many widely used data management tools to interface with the system. The software can be deployed using a broad variety of low-cost, industry standard storage hardware while leveraging cost effective existing file systems and object stores. Different object storage systems enable Campaign Storage's use in relatively high performance environments.

Los Alamos National Laboratory (LANL) implemented Campaign Storage with a file system called MarFS, layered on file systems and object stores. MarFS is supplemented with a distributed parallel data mover called "pftool". Pftool is a parallel program with functionality like rsync [39], but it offers some novel features, such as quick restarts of petabyte scale file transfers and packing of a batch of small files into a single object in MarFS. Other data movement tools such as Lustre HSM [19] movers and gridftp [40] are being adapted for MarFS.

[1] Kyle Lamb and Ben McClelland invented the name "Campaign Storage".

We believe two aspects of Campaign Storage are particularly interesting. First, the system has seen unusually rapid and successful development and roll out at LANL since 2014. At the time of writing (2017) LANL's implementation is in production at a world leading scale. Secondly, Campaign Storage's most relevant features involve advanced data management and bulk data movement operating on many files. Both the areas of bulk storage operations and data management are not optimally handled by the Unix file system interfaces, and have not yet benefited from agreed data structures and interfaces. Campaign Storage is an open source system and has a unique opportunity to explore and establish new open interfaces with wide reaching opportunities to innovate in data management.

As a topic of research, Campaign Storage explores a new tier of storage to bridge the ever widening gap in performance and capacity between the fastest memories and the archive. It explores scalability among 3 dimensions: data, metadata and directory data. It has semantics suitable to bridge storage tiers, and studies data management in an sea of data, much larger than what is contained in faster tiers. Between High Bandwidth Memories and tape, at least 3 other memory tiers are present today. Between each tier performance and cost of capacity decrease roughly an order of magnitude, while cost of bandwidth increases at a similar rate. Campaign Storage researches how to manage data across several of these tiers.

In this paper we start by reviewing the requirements for a Campaign Storage system. We then move to a discussion of the software architecture and the data movement utilities that provide novel storage management, pausing to review open-ended architectural areas, and its applications for data management using data movers. LANL's production Campaign Storage system is discussed next and we finish with alternative approaches and conclude our paper.

II. Motivation and Purpose

A. Campaign Storage

In many organizations, projects go through a period during which their data are actively processed from time to time. We refer to such a periods as a campaign. During the campaign project data are staged on nearline storage for processing and after processing data may be de-staged from nearline storage. When a campaign terminates, some or all project data may need to be archived.

We refer to a storage system to meet the requirements of (i) staging from and de-staging to and (ii) archiving as Campaign Storage. In 2014 LANL articulated its vision for Campaign Storage. Campaign Storage, like most successful storage systems, has been developed to accommodate specific use patterns. We proceed to analyze the requirements underpinning Campaign Storage.

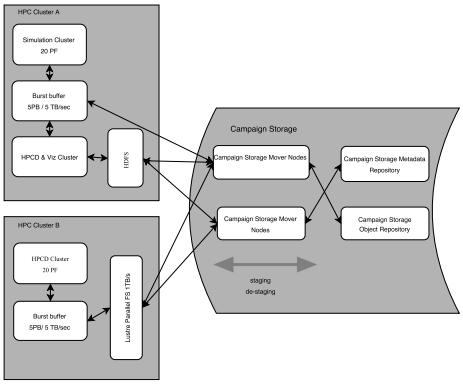

Fig. 1. Deployment of Campaign Storage in relation to other storage systems

B. Requirements Analysis

Nearline storage must offer high bandwidth, and in scientific computing burst buffers and parallel file systems meet this need. While solid state storage and the hardware on which cluster file systems are deployed may offer the most cost-effective solution for bandwidth, they command a significantly higher price for capacity compared with commodity storage. The trend to use higher performance nearline storage leads to using a system with lower capacity and hence to a requirement for a Campaign Storage system with capacity equal to a multiple of the nearline tiers, offering high durability and sufficient bandwidth for staging and de-staging.

A quantitative formulation of the requirements is obtained as follows. A leading facility implementing a 50PF system may choose to have 10PB of nearline storage, following basic system architecture scalability models. A project using the system at capacity may use a significant portion of this nearline storage, for example one third assuming that concurrently one project is being staged, one project is being de-staged and one project is processing. If this facility would pursue for example 100 projects, Campaign Storage should target 300PB of capacity and staging and de-staging IO rates should allow several petabytes of data to move in a shorter time than the processing slot offered to the project and quickly enough to limit exposure to data loss in the nearline storage. At the scale described, 10's to 100's of GB/sec of bandwidth are required. Bandwidth of traditional HSM scales with the number of (high cost) drives holding storage devices, not with the number of storage devices.

A parallel data mover is required as single nodes will not offer sufficient bandwidth to migrate data. Scientific computation sees other extremes: both extremely large files (PB in size) and extremely many files (billions per directory) may need to be handled.

When data is prepared for archival, typically a policy is created automatically or manually to select what must be retained and moved to the archive system. This policy is commonly understood and/or implemented using indexes of the file system. Traditionally such indexes and policies are created on the nearline storage systems and pose scalability challenges and load unrelated to requirements of the computational systems. Having 100's of projects with independent indexes and policies is not convenient. Campaign Storage must support scalable indexing and search to address the needs across all projects which might reside on faster storage tiers during a campaign.

A project's information provides a natural subdivision of data in a Campaign Storage system. A container or fileset abstraction with good administrative properties can be beneficial. For example, many projects seek to make their data available in public cloud systems and it can be useful to instantiate containers on single workstations instead of storage clusters.

We have described requirements arising in scientific computation environments, but similar considerations apply to other situations. For example, organizations with enterprise NAS systems face comparable cost considerations and the data on one enterprise server sometimes plays a similar role as project data in an HPC environment.

III. Architecture & Implementation

A. Architecture

Campaign Storage offers a file system interface. The Campaign Storage file system, which is called MarFS, aggregates the storage offered by multiple file system namespaces into its metadata repository (CS MR) and the storage offered by multiple object stores into its object repository (CS OR). As a result, the inodes in MarFS have a structure familiar from other file systems: the inodes in the metadata repository leverage extended attributes, and when needed also file data in the metadata repository, to describe the layout of the file data across one or more objects. Optionally, MarFS can store files directly in the metadata file system and offer POSIX IO semantics. Fig. 2 shows its layered decomposition.

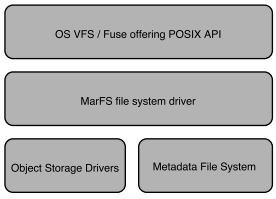

Fig. 2. Software modules in Campaign Storage

The module decomposition of Campaign Storage software is simple: MarFS is a typical filter file system: it leverages the metadata repository almost directly, with little or no transformation of API's and data. When performing IO on file data, it leverages layout information in extended attributes and

performs IO directly with one or more of the object storage systems. MarFS leaves all communication among participating clients and servers to the underlying file system and to the object stores' IO libraries. This architecture implies that Campaign Storage exclusively uses existing mature storage systems, leveraging their maturity. The layering and separation of data and metadata introduces additional latency for handling requests compared with a single integrated system.

An important consequence of this architectural decision is that both metadata and file data are accessible through widely used API's. A trade off arises from using these IO interfaces because object systems typically offer API's more constrained than the read/write POSIX interface. Object storage systems offer write once semantics ("PUT"), some offer appending writes, and fewer offer writes at random offsets and overwrites. MarFS, depending on the backend object store used, also supports a limited IO interface. This limitation can be overcome by writing file data into many smaller objects through which read-modify write operations affect only the objects overlapping with the modified region. This method is exploited by Dropbox [2].

In some instances it is best to write or read multiple files through a bulk operation presented to the object stores. For this, MarFS supports a file level IO vectorization interface, which extends the readv and writev interfaces found in the Unix system calls:

```
int copy_file_range_fv(struct copy_range *r,
unsigned int count,int flags)
struct copy_range {
        int source_fd;
        int dest_fd;
        off_t source_offset;
        off_t dest_offset;
        size_t length;
}
```

Fig. 3. File level IO vectorization

The semantics of the function is to copy a range of file extents from source file descriptors to destination file descriptors at varying offsets. This interface allows many files to be packed into one, by writing source files at an offset in the destination file. It also gives the file system complete control over the order in which extents in files should be transferred, which can be used to advantage particularly when retrieving files from a serial storage medium like tape and can be exploited as a load balancing feature by leveraging layout information.

The deployment of Campaign Storage proceeds as follows. Server nodes are configured for metadata and object repositories. MarFS client nodes are generally used for data movement and they need access to both MarFS and to other storage systems to which data will be staged. Each client node

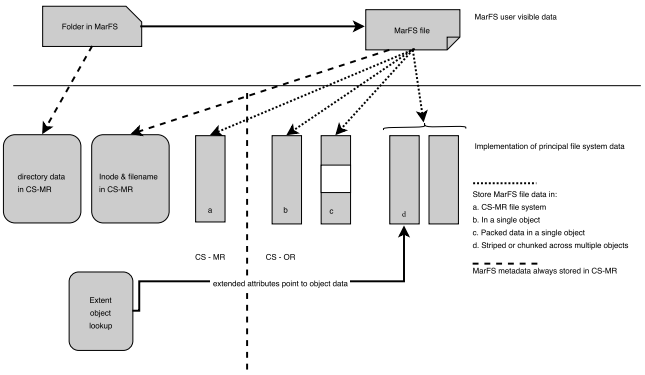

Fig. 4. Mapping file & directory data structures to data in metadata and object repositories

mounts MarFS, and the networked metadata file systems in its metadata repository will connect with the server nodes. There is no server node for Campaign Storage itself.

The architecture addresses file IO performance through the use of multiple servers and correct data placement. By default file data is placed in a single object for which the object storage service and bucket are derived from metadata at the namespace level. However, object size limitations may require the file data to be split into multiple chunks, each chunk to be stored as an object, and when this is required a lookup table mapping file offset to chunks is placed in file data on the metadata repository. POSIX semantics are required when storing sparse files and such files can be placed directly in the namespace. Other options include packing of the data of small files into a single object, while the keeping metadata for each packed file available as an individual entry in the metadata repository. Truly large files can be sharded over multiple object stores. Several other options exist, for example redundancy beyond that found in the object storage system is often required.

Campaign Storage also plans to offer several layout mechanisms for metadata. The basic layout simply uses the cluster file system of the CS MR without modifications. Storing embedded databases as an alternative form of a directory subtree can enable bulk insertions of trees of files. Very large directories should be striped over underlying CS MR directories residing in multiple namespaces. At present, only the basic layout is offered.

B. Indexing and Search

A major challenge in data management is to maintain an inventory of the file system. Traditionally a policy manager belonging to an HSM or data management system will crawl the source file system to index the file trees. Only indexing at modest speeds can be tolerated, higher rates produce too much load on the source file system and also on the database system holding the indexes. Campaign Storage makes two changes to this pattern. First, it doesn't build a general purpose index but one using coarser sets of values. Secondly it maintains indexes in the Campaign Storage file system, not in the source, leveraging MarFS' changelog data between snapshots.

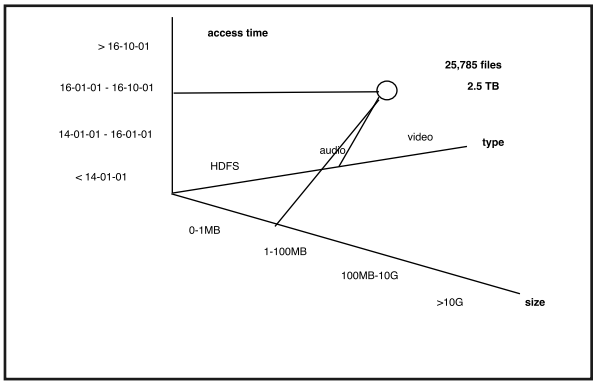

Fig. 5. A 3 dimensional histogram system for file size, access time and type histograms

The index system for Campaign Storage consists of subtree views (STVs), which are small databases, one attached to each directory. Each STV contains a database with a set of histograms. The histograms are defined by predicates. The key for a record in the database for a histogram identifies its predicate, and the values are three positive integers, where the first two integers equal the number of files and the number of directories in the subtree matching its predicate, and the third integer equals the number of bytes in matching files. Each file and directory defines such values, and the histogram of a directory is formally recursively defined as the sum of the histogram of its files and its sub-directories. Operations on the file system lead to operations on the histograms and the operations and the formation of histograms commute, hence logs of operations can update the histograms.

Examples of interesting histograms are easy to find. A traditional triple of predicates such as file size, last access time intervals and file type leads to a multi-dimensional histogram. An example in Figure 5 shows a system for a 4 x 4 x 3 histogram, which classifies files by type, access time and size. For this histogram system at most 48 records are created in each database file, possibly with a 49th record indicating "other" if the disjunction of all predicates does not match any files. As shown in figure 5 one may deduce from the content of one bucket that the applicable subtree contains 25,785 files and 2.5 TB of music. For multi dimensional histograms like these, the usual boolean algebra of search expressions exists.

Other useful examples have histograms with predicates classifying which storage servers are used for data storage. A histogram with users and groups gives not only a quota system with subtree quota usage information, but can also be used to create an identity database for the files in the subtree. A custom histogram may use predicates evaluated by file data - for example by leveraging geospatial information or image tags found in the files. We expect that there are many use cases we have not thought off which can be flexibly addressed.

Because file systems are always changing, subtree views also change and are best associated with snapshots of the file system. ZFS snapshot differentials produce the views efficiently, and ZFS maintains a parent directory id for those files for which the link counts remains 1. An auxiliary database that lists parents of files with link count > 1 is required to support hard links. Subtree views associated with snapshots are fully consistent and can form the basis of data management.

Campaign Storage offers a file system interface, hence it is compatible with most data management systems that perform ILM. Subtree views form an alternative for search to create sets for policies to act on. Building the STV index is first performed by copying the metadata of the source file system to Campaign Storage and is then updated using changelogs. In fact one sees in reports on HSM [19] that frequently before data management is performed, the database information is converted to histogram summaries. Another powerful alternative to policy databases is offered by scanning based ILM in IBM Spectrum Scale, and this can also underpin Campaign Storage.

A histogram record contains 40 bytes. Auxiliary data, such as versions and a linked list of subdirectories, may add a few bytes to the STV. Consider a file system with 10M directories and a file to directory ratio of 100:1. This might typically consume around 40GB of directory data, and adding histogram data to the directory subsystem with an average of 10,000 histogram entries per directory leads to 4TB of STV data. The Robinhood policy database was observed to consume approximately 1KB per file in the file system, for a total of 1TB. Well tuned production Robinhood database systems appear to require up to 50% of the database size in system RAM ([19]) and are then observed to ingest 100M records at rates of approximately 4,000 records/second. A single threaded application on a KVM virtual machine with 2GB of RAM and access to a single consumer grade PCI flash device ingests metadata of 500M files at a rate of 19,000 files/second into ZFS in a single threaded application. Using clusters of movers and servers, LANL scaled this to over 800M files/second in a recent study ([41]). Some of the search examples shown by Robinhood users compute a single histogram item take up to 15 minutes to execute. While these are encouraging early indicators there are unquestionably important difficult cases where our approach is at a disadvantage, and these will only be discovered with more users.

C. Campaign Storage and Cloud

An attractive use model for Campaign Storage is to associate a fileset, a concept which perhaps originated in AFS volumes [25], and an object store bucket with a particular data management task, for example the staging and archival of a particular project or of a particular user's file.

Such filesets and buckets are easily isolated and reused. A single fileset and perhaps even its history in snapshots can be stored as an object of manageable size in a cloud system, where a cloud application running ZFS can access it. The bucket with objects associated with the fileset can be replicated to a cloud system for later access. Such objects and metadata volumes can be migrated with tools like Amazon Snowball. Also a single workstation might extract a smaller

fileset including its objects, and mount it using Campaign Storage when greater interactivity and less data exchange with server systems or with the cloud is desired.

Campaign Storage supports replication, leveraging ZFS and the object storage systems, and this offers multi-location archival storage, a frequently desired feature.

A MarFS client could move data between parallel file systems or burst buffers by writing object data directly to S3 and could theoretically perform metadata operations with servers in the cloud, but it can be expected that much refined utilities are required to make this efficient.

D. Open Architectural Questions

Unix file system API's have proven sufficiently powerful to see 35 years of re-use since being introduced in the early 70's. However, in the never ending pursuit of better performance and data management, little has been done to get universally adopted API's beyond this and conceptually there is a lot of variation between storage systems. Agreement on richer API's enable the construction of portable highly efficient data movement tools, while more complete POSIX compliance will support a wide family of existing utilities.

Numerous storage systems, including Campaign Storage, all parallel file systems and object storage systems, have defined data layouts. The layout of a file can be determined at the level of an individual file, at the level of a directory subtree in the file system, or at the level of a metadata namespace. It is valuable to allocate any layout resources only at the time IO is performed and not before. Data layouts become a concern for users when detailed control of IO performance and availability of data are desired. Data layouts include data replication, erasure codes and other redundancy schemes and layouts for sections of files.

Beyond the layout of data, layout descriptors could also offer service descriptions to see if they are a good match for impending operations. Particularly in a tiered storage system many other variations may be important. In a layout descriptor, servers or their storage targets are named or referenced, and objects are named together with other data describing redundancy patterns. There is no widely adopted standardized description for layouts, and an agreed interface for this data could have very wide usage.

Archival systems manage data in a primary storage system and a related archival system. This involves archival attributes indicating what data is available in a primary and/or in a related archival storage system. Archival systems perform most operations asynchronously and depend on changelogs and indexes, for example to continue copying data to an archive subject to a policy or to replay changes such as file removals in the archive. Neither indexes nor changelogs have seen standardization.

Better definitions and interfaces are also required for operations on aggregate structures such as collections of extents in files and subsets of the namespace of a file system. This is a larger area and we will try to indicate how this could develop with a number of examples. Regarding the bulk modification of file data, we have already discussed the file level IO vectorization, which extends the per file vectorization, and successfully encoded the semantics of Campaign Storage data movers. This call does not include bulk truncate operations (and also not fallocate's "punch" implementation using `FALLOC_FL_PUNCH_HOLE` and it includes no description of concurrency. File level IO vectorization may be a sufficiently general interface for a single thread of control transferring data between two sets of linear address spaces, represented by the two sets of file descriptors. However, if communicating processes perform IO this opens up many further possibilities. Among others, collective IO [1], PLFS [6] and the ADIOS [2] IO patterns have demonstrated significant benefits of such patterns, but the nature of the data structures involved is not yet clear and may relate closely to distributed layouts of data. For example, the peer-to-peer patterns of communication found in bittorrent [38] go beyond what these descriptors offer and have demonstrated unique scalability for some problems.

Bulk handling of metadata is becoming essential in scientific computing, where it is expected that basic operations may create or move billions of files, and Campaign Storage aims to meet this requirement. This topic has been researched in a sequence of designs from CMU [26], [27], [28], and a driving design pattern for this is client side write back metadata caching, which scales creation performance with each client contributing approximately local file system rates, rapidly exceeding anything servers can offer. Small database tables are a form of containers representing subsets of filesystems. Integration of these into existing file systems using a serialized format for containers and their differentials, is a requirement for a bulk API that efficiently addresses advanced metadata operations. However, metadata retrieval may not see similar improvements. Achieving POSIX semantics when storing metadata on multiple servers is notoriously hard, even to enforce essential behavior such as not renaming a directory into its own subtree. Further complexities surround operations that move a file or directory of the containing file system into such containers or rename elements across servers. The containers containing subsets of file system metadata, their identification for bulk movement, their grafting relationships among each other as well as concurrency constraints and parallelization opportunities have seen mostly ad-hoc approaches.

E. Data Movement and HSM

A large number of data movement utilities exist and it may be surprising that a number of them leverage sparse files which are not directly supported by object storage systems used in MarFS.

Pftool is a distributed application to move data between two file systems mounted. It leverages MPI as its network protocol. The functionality pftool offers falls in 3 groups: list, copy and verify. Listing operations scan the file system in parallel, copy moves metadata and data and verify validates that the data is correct. pftool's roadmap is to offer functionality much aligned with rsync in a distributed fashion. Los Alamos also incorporated functionality in pftool to pack small files into

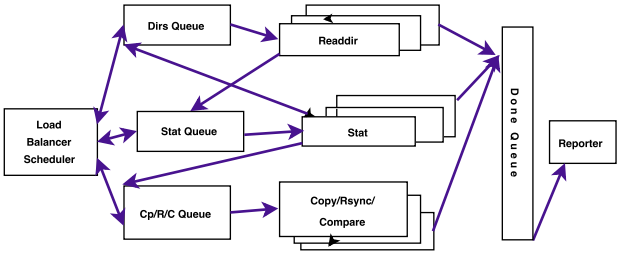

Fig. 6. pftool functional decomposition

single objects. A further important feature of pftool is to restart aborted operations efficiently, leveraging an extended attribute to communicate this to a running instance. A detailed architecture of pftool is discussed in a report [4] published by LANL.

A second data mover for Campaign Storage is a Lustre HSM mover [29]. This data mover targets an object storage system named Spectra Logic Black Pearl [30] which implements objects on tape systems. Here refined control over the interface between Lustre and MarFS is required, to ensure that batches of requests can be executed using a bulk interface, and to allow prioritization based on the order of data on tape and operational considerations. MarFS and the HSM mover provide fundamental HSM functionality that also supports the more widely used DMAPI [32] interface. While third party policy manager such as Robinhood [33] or Komprise can coexist with this solution, a new possibility is to create metadata-only copies of source file systems in MarFS and leverage the indexing mechanisms described to replace the policy manager. The ingest rate to metadata repositories appears to be easily an order bigger than to policy manager databases. Campaign Storage directly supports handling multiple archival destinations, such as faster storage for use while data is still regularly accessed while migrating to cold storage when this requirement ceases to exist.

IV. LANL'S CAMPAIGN STORAGE DEPLOYMENT

A. Basic System Architecture

Los Alamos deployed a first prototype of Campaign Storage on commodity hardware nearly three years ago in 2014. This first iteration inspired the architecture described thus, and the driving factor in its deployment was identical. Refining this architecture and completing the implementation MarFS led to the current production deployment at Los Alamos between 2014 and 2016.

The deployment of Campaign Storage at Los Alamos is composed of a 48 node commodity cluster of storage servers, a 3 node cluster of metadata servers, and a 30 node cluster of file transfer agents (FTAs). The 48 storage nodes have multiple disk enclosures attached to them with a raw capacity of approximately 64 petabytes in aggregate. The FTAs mount all of the external filesystems (NFS, Lustre, HPSS) to facilitate movement between those systems and Campaign Storage. The three metadata servers run IBM's GPFS, which allows for

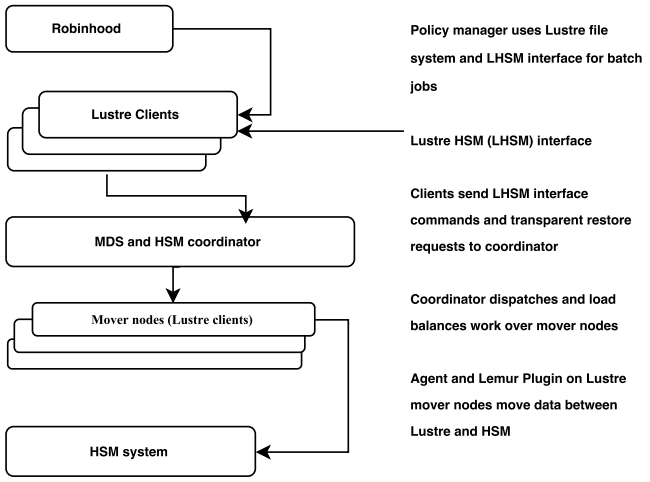

Fig. 7. Lustre HSM building blocks

both high-speed clustered metadata access as well as the full suite of information lifecycle management (ILM) tools that allow for easy management of millions to billions of user files. Current usable capacity is artificially limited to 23 petabytes to facilitate future expansion onto LANL's in-development tiered erasure model.

This production system was deployed in the fall of 2016 and provides users with approximately 20 GB/s of read and write bandwidth when transferring large datasets across multiple parallel transfers. Data is currently stored via the Scality RING software configured with 20+4 erasure and user quotas are implemented through a custom tool that interfaces directly with the GPFS internal API. Data access to the Scality RING is through an S3-like interface that utilizes curl. Each computing campaign project is segregated into its own GPFS fileset for both ease of management and expedient inode scanning for timely quota enforcement. LANL is continuing to work on user engagement in an effort to more fully utilize the newly available Campaign Storage tier. User data set sizes of hundreds of terabytes have already been created and stored within the current system.

B. Challenges in Deployment

Deploying large-scale storage systems such as the one described here typically are not without some amount of pain from both users and administrators. Additionally, deploying a system with a new software stack at a scale never tested before also generally presents its own difficulties. The production deployment of MarFS Campaign Storage at Los Alamos was not immune to these issues, however, the rollout was handled quite readily by a pair of system administrators with no loss of user data or extensive downtime.

One difficulty that we found in early user testing of the system was that standard Unix command line tools don't necessarily write sequentially. Specifically, "cp" will recognize

strings of zeros and create sparse output files by seeking ahead and not writing those buffers. Typical tools like "rsync" are also generally incapable of functioning on a purely sequential basis. Some of these optimizations can be avoided by disabling sparse file creation via flags, however, Los Alamos decided to simply restrict data movement into the system by disabling FUSE writes and only allowing users to utilize pftool for transfers. This hardened the idea to users that this wasn't a typical storage system, and that they could not expect their past workflows to translate directly. While such restrictions are common to all domain specific storage systems, we think it is attractive to gradually overcome them. Reading data through FUSE as well as manipulating metadata were left open for user interaction.

C. Future Development

While the initial goals of the Los Alamos Campaign Storage tier were modest at approximately 1 GB/s per petabyte usable, Los Alamos is continuing to pursue new storage paradigms that will increase both data safety and usable bandwidth. This will be achieved through the use of both additional erasure coding in MarFS itself as well as utilization of next-generation filesystems like ZFS for the underlying data storage.

V. Rationale and Releated Work

Many may question why a new effort like MarFS is necessary. The central requirement is to provide a scalable near-POSIX interface with adequate performance, cost, and efficiency, at this time.

Current object storage systems incorporate erasure coded data layout and excellent management and good scaling for data storage. In contrast RAID solutions do not offer adequate data protection at a truly massive scale. The object system built into parallel file systems such as Lustre or Panfs lack manageability, widely used interfaces and place load on the metadata services. Object storage systems are not suitable for exporting namespaces. While products such as Cleversafe, Scality, and EMC ViPR are moving towards the "sea of data" concept where data can have multiple personalities including POSIX, Object, and HDFS [16], such namespace solutions are not near POSIX and not efficient. MarFS offers the metadata scalability of N POSIX name spaces.

It is possible to put object storage systems under scalable file systems like GPFS [17] using a block interface over the object storage system, but the block write patterns of these parallel file systems are not well suited to benefit from these object storage systems' high performance and manageability. MarFS will be able to use any object storage system, including cloud-based services, as a back end storage repository.

The team has investigated existing open source projects, and there doesn't appear to be one that provides the needed functionality. Ceph [23] provides a file system on objects, but isn't known for scaled out metadata service. GlusterFS [24] can offer a global name space combining multiple file systems into one mount point. It also hashes the file names across the file systems, something MarFS does not yet offer. The main difference is the approach to what GlusterFS documentation refers to as unified file and object. GlusterFS has been integrated to be object storage for OpenStack Swift (for objects) and block storage for OpenStack Cinder (for blocks). Conversely, MarFS is designed to put a near-POSIX interface over any object storage system, including OpenStack Swift.

One may question if HSM (hierarchical storage management) systems such as HPSS [8] (High Performance Storage System) or DMF [9] (Data Migration Facility) can meet the requirements. These systems currently don't take advantage of the enormous industry investments in object storage. HPSS metadata performance is likely 1/10th or less of what MarFS metadata performance is expected to be. MarFS will leverage existing tools and be a small amount of code to combine these tools. MarFS will not offer all features of an HSM system, although batch utilities could move data around inside of MarFS to various kinds of storage systems directed by data management policies. Histogram based policy management appears also in products like Komprise [10] and Apple's 2016 file system [31], and appears related to mutidimensional data modeling [42]. HSM systems are not generally highly parallel. MarFS is designed for dozens to hundreds of metadata servers/name spaces and thousands or even tens of thousands of parallel data movement streams. HPSS is designed for an order of magnitude less parallelism.

There are products that are optimized for WAN and HSM metadata rates. For example, General Atomics Nirvana Storage Resource Broker, iRODS [15] (Integrated Rule Oriented Data Systems). Many parallel file systems implement POSIX files over objects with full POSIX semantics, but due to synchronization of metadata, such approaches are not known for massive parallelism in a single file.

The team has looked at name space solutions. EMC's Maginatics [14] offers a hybrid approach, but is targeted at enterprise use cases. The Camlistore [12] open-source project appears to be targeted and personal storage. Bridgestore is a POSIX name space over objects, but it places metadata in a flat space. Avere NFS [11] over objects is focused at NFS so shared file N-1 will not be high performance.

VI. Conclusions

Campaign Storage offers a straightforward architecture and promises data management, staging and archiving of massive collections of files, leveraging existing systems with best of breed qualities for metadata and data storage. It has seen an unusually quick rollout into production and can offer a radically simpler approach to HSM, including built in policy management. Standardization and development of agreed interfaces for data management and bulk data movement will contribute significantly to embrace different platforms with multiple data management tools and as an open source system Campaign Storage exemplifies the needs for this and may contribute to this significantly.

References

[1] R. Thakur, W. Gropp, and E. Lusk, "Data Sieving and Collective I/O in ROMIO," *In Proceedings of the The 7th Symposium on the Frontiers*

of Massively Parallel Computation(FRONTIERS '99). IEEE Computer Society, Washington, DC, USA, pp. 182-189, 1999.

[2] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS)," *In Proceedings of the 6th international workshop on Challenges of large applications in distributed environments* (CLADE '08). ACM, New York, NY, USA, pp.15-24, 2008.

[3] "Dropbox explanation of object layout," Stanford University, (Date last accessed February, 2017). [Online]. Available: https://www.youtube.com/watch?v=PE4gwstWhmc, Published on Sep 10, 2012.

[4] H. Chen, G. Grider, C. Scott, M. Turley, A. Torrez, K. Sanchez, and J. Bremer, "Integration experiences and performance studies of a COTS parallel archive system," IEEE Cluster 2010 Conference, pp. 166-177, 2010.

[5] B. Chamberlain, "Programming Models at the Exa-scale," Cray Inc, Cross-cutting Technologies for Computing at the Exascale, February 2nd 2010 Rockville, MD, Available: http://chapel.cray.com/presentations/Chamberlain-ProgEnv-CrossCut.pdf

[6] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate, "PLFS: a checkpoint filesystem for parallel applications," *In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09)*. ACM, New York, NY, USA, 2009.

[7] J. Bonwick, M. Ahrens, V. Henson, M. Maybee, and M. Shellenbaum, "The Zettabyte File System," Technical report, Sun Microsystems., pp. 1-13, 2003.

[8] H. Hulen, O.Graf, K. Fitzgerald, and R. W. Watson. "Storage Area Networks and the High Performance Storage System". *10th NASA Goddard Conference on Mass Storage Systems and Technologies*, 2009.

[9] Silicon Graphics International Corp,"DMF", (Date last accessed February, 2017). [Online]. Available://www.sgi.com/products/storage/tiered/dmf.html

[10] "Komprise,"(Date last accessed February, 2017). [Online]. Available: https://www.komprise.com/

[11] Avere NFS, "Fastest Performance for Cloud & On Premises," (Date last accessed February, 2017). [Online]. https://www.averesystems.com/solutions/applications/performance .

[12] B. Fitzpatrick and M. Lonjaret, Lecture Presentation, "Camlistore: Your personal storage system for life," LinuxFest Northwest 2016, Available: https://www.youtube.com/watch?v=8Dk2iVlc67M .

[13] H. Reuter, "Hiding HSM Systems from the User," *IEEE Symposium on Mass Storage Systems 1999*, pp. 215-221.

[14] Maginatics, "Directory Write Leases in MagFs," Lecture slides, (Last Date accessed February, 2017). [Online]. Available: http://maginatics.com , Mountain View, CA, United States.

[15] Open Source Data Management Software, "iRODS Consortium," (Date last accessed February, 2017). [Online]. Available: https://irods.org

[16] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," *In Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* (MSST '10), IEEE Computer Society, Washington, DC, USA, pp. 1-10,2010.

[17] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," *In Proceedings of the 1st USENIX Conference on File and Storage Technologies* (FAST '02), USENIX Association, Berkeley, CA, USA, 2002.

[18] E. Barton and A. Dilger, "High Performance Parallel I/O," Chap. 8, pp. 91-106, CRC press, Boca Raton, 2015.

[19] C. Beyer, "Robin Hood 2.5 on Lustre 2.5 with DNE, Site status and experience at German Climate Computing Centre in Hamburg," DKRZ, http://robinhood.sourceforge.net/rug16/RUG16_DKRZ.pdf, September 2016 [Date last accessed February 2017]

[20] "Cleversafe," Wikipedia, (Date last accessed February, 2017). [Online]. Available: https://en.wikipedia.org/wiki/Cleversafe

[21] "Scality," (Date last accessed February, 2017). [Online]. Available: https://scality.com/

[22] DellEMC, "Dell EMC Vipr Controller, Automate and Simplify Storage Management," (Date last Accessed February 2017). [Online]. Available: https://www.emc.com/collateral/data-sheet/h11750-emc-vipr-software-defined-storage-ds.pdf

[23] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: a scalable, high-performance distributed file system," *In Proceedings of the 7th symposium on Operating systems design and implementation* (OSDI '06), USENIX Association, Berkeley, CA, USA, pp. 307-320, 2006.

[24] Red Hat, Inc [US], "GlusterFS," (Date accessed February, 2017). [Online]. Available: https://www.redhat.com/en/technologies/storage/gluster

[25] M. Satyanarayanan, "Scalable, Secure, and Highly Available Distributed File Access," Computer, IEEE, Carnegie Mellon University, May 1990.

[26] Q. Zheng, R. Kai , G. Gibson, W. Bradley , G. Grider, "DeltaFS: Exascale File Systems Scale Better Without Dedicated Servers," *Proc. of the Tenth Parallel Data Storage Workshop (PDSW15)*, co-located with the Int. Conference for High Performance Computing, Networking, Storage and Analysis (SC15), Austin, TX, November 2015.

[27] L.Xiao, Lin, K. Ren, Q. Zheng, and G. A. Gibson. ShardFS vs. IndexFS: Replication vs. Caching Strategies for Distributed Metadata Management in Cloud Storage Systems, 2015 ACM Symposium on Cloud Computing (SOCC 2015), Aug 29-30, 2015, Hawaii.

[28] K. Ren, Q. Zheng, S. Patil, G. Gibson, "Scaling File System Metadata Performance With Stateless Caching and Bulk Insertion," *ACM/IEEE Int'l Conf. for High Performance Computing, Networking, Storage and Analysis* (SC'14), November 16-21, 2014, New Orleans, LA.

[29] Wikipedia,"Lustre HSM Tools design," (Date last accessed February 24, 2017).[Online]. Available: https://wiki.hpdd.intel.com/display/PUB/HSM+Agent+Design

[30] Spectra, "Spectra Logic Black Pearl", (Date last accessed February 24, 2017). [Online]. Available: https://www.spectralogic.com/products/blackpearl/

[31] ARS Technica, Apple File Systems (APFS), "A ZFS developer's analysis of the good and bad in Apple's new APFS file system,"(Date last accessed February 24, 2017). [Online]. Available: https://arstechnica.com/apple/2016/06/a-zfs-developers-analysis-of-the-good-and-bad-in-apples-new-apfs-file-system/

[32] DMAPI, Systems Management: Data Storage Management (XDSM) API, CAE Specification, X/Open Document Number: C42, ISBN: 1-85912-190-X,1990, (Date last accessed February 24, 2017). Available: http://pubs.opengroup.org/onlinepubs/9657099//

[33] T. Leibovici, "Taking back control of HPC file systems with Robinhood Policy Engine", CoRR, abs/1505.0144, 2015.

[34] Lustre HSM Tools design,(Date last accessed February 24, 2017). [Online]. Available: https://wiki.hpdd.intel.com/display/PUB/HSM+Agent+Design

[35] Spectra Logic Black Pearl, (Date last accessed February 24, 2017). [Online]. Available: https://www.spectralogic.com/products/blackpearl/

[36] Apple File Systems (apfs), (Date last retrieved February 24, 2017). [Online]. Available: https://arstechnica.com/apple/2016/06/a-zfs-developers-analysis-of-the-good-and-bad-in-apples-new-apfs-file-system/

[37] DMAPI,(Date last accessed February 24, 2017). [Online] Available: http://pubs.opengroup.org/onlinepubs/9657099/

[38] Bittorrent, (Date last accessed February 2017). [Online]. Available: http://bittorrent.org

[39] Andrew Tridgell, Paul Mackerras, "The rsync algorithm",(Date last accessed February 2017). [Online]. Available: https://rsync.samba.org/tech_report/

[40] Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M., "The Globus Striped GridFTP Framework and Server." *ACM/IEEE SC 2005 Conference* (SC'05). p. 54, 2005.

[41] J. Inman, W. Vining, G. Ransom and Grider, G. "MarFS, a Near-POSIX Interface to Cloud Objects", Usenix ;login: Spring 2017, Vol. 42, No. 1

[42] T.B. Pedersen, C.S. Jensen, C.E. Dyreson "A foundation for capturing and querying complex multidimensional data", Information Systems 26 (5), 383-423