Open Tape for Open Compute

Slavisa Sarafijanovic, Martin Petermann, Mark A. Lantz, Robert Haas
IBM Research - Zurich, CH-8803 Rueschlikon, Switzerland
sla, map, mla, rha @zurich.ibm.com

***Abstract*—Recently developed open-source software components that facilitate the integration of tape into open compute and cloud storage environments are described. Workload trends and economic aspects motivating the use of tape are also discussed.**

Keywords—Tape hardware, tape software, filesystem storage, object storage, cloud storage.

I. Introduction

The continued exponential growth in data combined with the recent slow-down in areal density and $/GB scaling of HDD is driving interest in lower cost alternatives. Magnetic tape is well suited to meet this demand due to its high reliability, low power and very low total cost of ownership (TCO). Already in 2017, the LTO (linear tape open) program reported that the cost of LTO7 media had fallen to less than one cent per GB [1]. More recently, in a 2018 update to a previous study, Enterprise Strategy Group found the TCO of an LTO8 tape solution to be only 14% of that of an all disk solution [2]. The huge potential to continue scaling areal density and hence to further reduce cost per TB is further driving interest in tape. For example, the latest Information Storage Industry Consortium (INSIC) Tape Technology Roadmap predicts that the areal density of tape systems can continue scaling at the historical rate of about 34% CAGR until at least 2029, enabling the scaling of cartridge capacity from the current state of the art of 20TB native capacity to many hundreds of TB [3]. The potential to continue scaling tape areal density has also recently been experimentally validated with a demonstration of tape recording at 201 Gb/in^2, more than 17x the areal density of the latest 20TB cartridges [4].

Tape's low cost and scaling potential are driving widespread interest in tape by hyperscale cloud companies such as Google and MS Azure. One of the barriers to wider tape use in other cloud environments is the perception that tape is difficult to use and manage. This perception probably arose because tape is a block storage device that historically was managed using proprietary software solutions with a separate database to manage the metadata of the data stored on tape. However, recently there has been significant progress in the development of technology that makes tape easier to use, manage and integrate with disk-based infrastructure. For example, the linear tape file system (LTFS) introduced by the LTFS Consortium in 2010 is an open, self-describing file system for tape cartridges based on technology developed by IBM [5]. In its earliest form, IBM LTFS SDE[1] (single drive edition) [6], it enables a user to mount a tape cartridge as a file system and browse it using a standard file browser on Linux/Windows/Mac, including drag and drop functionality. More recently, IBM LTFS LE[1] (library edition), which is available free of charge [7], extended this functionality to an entire robotic tape library where each cartridge is visualized as a separate folder. Modern use cases and workflows have created a need to seamlessly integrate tape with disk storage rather than managing tape separately. Recently, proprietary solutions have been developed, such as LTFS EE[1] (enterprise edition) [8], that integrate disk and tape with the preservation of a common name space, enabling data to be transparently migrated between disk and tape. In the remainder of this paper we focus on two open source solutions for integrating disk and tape that provide some of the core functionalities present in proprietary solutions: LTFS Data Management (LTFS DM) for file-based storage and Swift HLM (High Latency Media) for object-based storage.

II. Open source LTFS DM for file on tape

LTFS DM is a software component that enables a standard Linux disk file system to be converted into a disk and tape file system, as shown in Fig. 1. LTFS DM exposes the original disk file system name space to the user and enables migration of file data to the locally attached tape storage, in this example via LTFS LE. Upon file access, data on tape is transparently recalled to the disk file system and served to the user/application, with the additional latency caused by tape access and transfer times. Alternatively, data on tape can be explicitly recalled, e.g. based on automated policies that cause the recall of a set of files that than become accessible with the shorter latency of disk or even SSD (Solid State Drive) access.

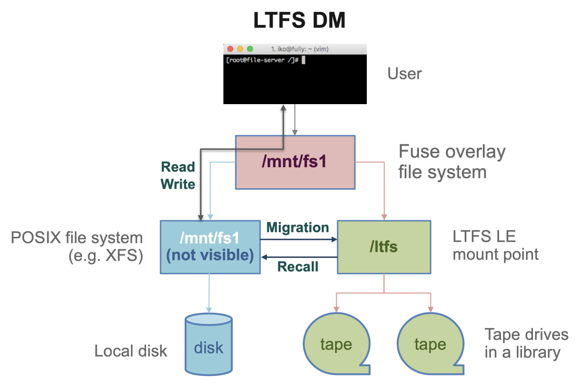

Fig. 1. *LTFS DM converts a Linux based disk file system into a disk and tape file system. Users/applications see and access the same file namespace as if accessing the original disk file system, but file data can be moved between disk and tape.*

[1] IBM LTFS SDE, LE, and EE, were recently renamed to IBM Spectrum Archive SDE, LE, and EE, respectively.

A. *LTFS DM user API/function*

Migration and explicit recall operations and the migration status check of a file are performed using the command-line interface of LTFS DM. Transparent recall operations are initiated on read, write, or truncate system calls (POSIX).

B. *LTFS DM design overview*

LTFS DM seamlessly integrates tape with standard disk file systems. This is achieved using a FUSE overlay file system that makes location of data invisible to the standard POSIX system calls, see Fig. 2. As a result, applications do not have to be rewritten to leverage tape. When a file is migrated to tape, a small stub that contains all the necessary information to recall it back from tape in the future is placed on disk. The FUSE overlay file system transparently handles changes of the data location. In addition, the original disk file path is stored on tape to recover from losses of the stub. To minimize access time to data on tape, requests are queued so that recall requests to data on an already-mounted tape cartridge are prioritized over recall requests for other cartridges, thereby minimizing the number of high latency mount operations. The LTFS DM software and documentation are available on GitHub [9].

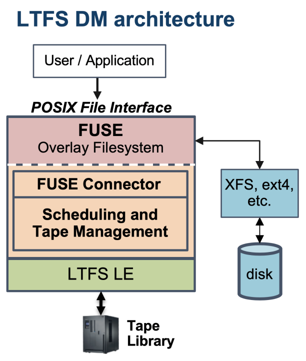

Fig. 2. *LTFS DM architecture. A FUSE overlay file system is used to map the original disk file system API and namespace but also to intercept and process user data access or data management requests.*

III. Open source SwiftHLM for object on tape

Swift High Latency Media (Swift HLM) is a software component for integrating OpenStack Swift object storage [10] with a high-latency storage backend, such as optical or tape libraries, as shown in Fig. 3. The HLM backend provides a file system that integrates a low latency media (LLM) storage such as a disk-based cache and an HLM storage such as tape, as well as an ILM (Information Lifecycle Management) function and an API for moving data between LLM and HLM.

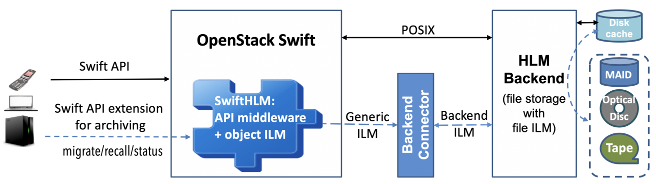

Fig. 3. *SwiftHLM extends Swift object API and provides an archiving (ILM) function for moving (migrate/recall) object data between low- and high-latency media (LLM and HLM)storage.*

A. *SwiftHLM user and backend APIs*

Users or applications store and read object data using the object API operations PUT and GET. Swift stores object data first to the LLM backend using the file system interface of the backend. Swift HLM extends the object API with a new type of archiving operations (namely migrate, recall, and status) by reusing POST and GET standard object primitives (see [11] for syntax details). It queues object or container ILM operations and processes them asynchronously by invoking the file ILM storage backend operations on the involved storage nodes of a Swift cluster.

B. *SwiftHLM internals*

Swift HLM components are shown in blue in Fig. 4 in an example of a multi-node deployment.

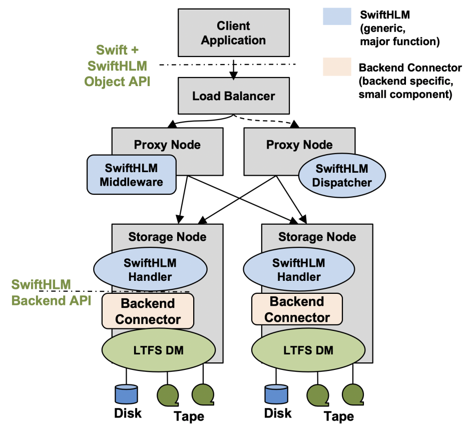

Fig. 4. *SwiftHLM components (blue) multi-node deployment using single-node LTFS DM storage backends (green) and connectors (orange).*

The SwiftHLM middleware running on Swift proxy nodes implements the archiving API operations and queues object ILM requests by storing them in a special container. Object ILM requests then get dispatched across Swift storage nodes by the Dispatcher. Finally, the Handler in each involved Swift storage node determines the file that stores a full copy or an erasure coded (EC) part of the object data and calls the backend ILM operations on the file.

Backend ILM operations are invoked via a generic backend API using a simple backend-specific connector that performs the necessary API translation. An example connector for a simulated storage backend and a connector for the LTFS DM storage backend are part of the open-sourced SwiftHLM software [11]. Any ILM-capable file-based backend can easily be supported in that fashion.

We are currently considering creating a SwiftHLM variant that supports invoking object ILM operations by writing object or container extended attributes (EAs), so to support archiving operations for both the Swift and S3 APIs of Swift. SwiftHLM internal processing could also be simplified for backends capable of intercepting and acting upon events of Swift storing object EA as file backend EA.

In conclusion, while tape is the most attractive media from a TCO perspective, the software components described here are key to its broader adoption in environments that use open filesystems or object-based storage.

References

[1] Tape cost, https://www.lto.org/2017/03/tape-and-disk-storage-what-do-they-really-cost
[2] ESG Economic Validation Summary Report, https://www.lto.org/wp-content/uploads/2018/08/ESG-Economic-Validation-Summary.pdf
[3] INSIC Tape Roadmap, http://www.insic.org
[4] S. Furrer et al., IEEE Trans. on Magn., 54, 2 (2018) 3100308
[5] LTFS standard https://www.snia.org/ltfs
[6] IBM LTFS SDE[1] https://www.ibm.com/downloads/cas/7VBLKOWZ
[7] IBM LTFS LE[1] https://www.ibm.com/downloads/cas/0O4JPLRD
[8] IBM LTFS EE[1] https://www.ibm.com/downloads/cas/MEJYJBAG
[9] LTFS DM https://github.com/ibm-research/LTFS-Data-Management
[10] OpenStack Swift https://github.com/openstack/swift
[11] SwiftHLM https://github.com/ibm-research/swifthlm