Five Years Of Experiences With Wide-Area User Space File System: SLASH2

Beth Lynn Eicher, Zhihui Zhang, Robert Budden, J. Ray Scott, Derek Simmel, Chris Rapier
Pittsburgh Supercomputing Center
300 South Craig St.
Pittsburgh, PA 15213
{ bethlynn, zhihui, rbudden, scott, dsimmel, rapier }@psc.edu

Jared Yanovich
Google
jaredy@google.com

Michael John Killean, Madison Cooley, Michael Schwindt
UW IT Department, ARCC
1000 E. University Ave.
Laramie, WY 82071
{mkillean,mcooley3, mschwin1@uwyo.edu}

ABSTRACT

The Pittsburgh Supercomputing Center (PSC) has developed the SLASH2 file system for wide area network (WAN), geographically dispersed, research computing resources. SLASH2 is an open-source, distributed file system that has been in production use at PSC for more than five years. Over these years, the development and deployment of SLASH2 has been supported by projects funded by National Archives and Records Administration (NARA) and National Science Foundation (NSF).

While SLASH2 exemplifies many of the design elements of a state-of-the-art distributed file system (e.g., GPFS [2] and Lustre[3]), it distinguishes itself from these file systems in two important aspects. First, SLASH2 was designed from the outset to be a wide-area file system with features like data replication and UID/GID mapping. Second, SLASH2 operates entirely as a user-mode application, which provides several important advantages compared to typical kernel-mode file systems.

In this paper, we will explain why the two aspects mentioned above are important for SLASH2. We will discuss the history, the architecture, and the deployment of SLASH2 as well as valuable lessons learned in the process. Today, SLASH2 is no longer a proof-of-concept file system. It has established a track record of serving the scientific community.

Keywords

Wide-area network, distributed file system, user space development, open source.

1. INTRODUCTION

The Pittsburgh Supercomputing Center (PSC) has a long history of developing software to meet the unique challenges in HPC environments that are not necessarily addressed by the industry. To reduce the use of an industrial tape library, a file system called SLASH [4] – scalable lightweight archival storage hierarchy was developed shortly after the deployment of the LeMieux terascale system in 2001. SLASH was a disk-based caching system built in front of the tape-based archiving system to reduce its latency. In 2008, PSC developed ZEST [5], a checkpoint data storage system for large supercomputers. PSC was awarded a patent for ZEST, which has inspired further work by other researchers [6]. The experiences gained through these two successful projects have proved to be valuable for the development of SLASH2, the next generation of SLASH..

The first production use of SLASH2 was to replace the multi-petabyte tape-based archiving system in order to reduce cost and improve performance [7]. These goals were within our reach because the price of disk drives had fallen and there are no software license fees for SLASH2. As part of the project, the tape-based archiving system one of the I/O servers of the first SLASH2 installation until all the tape data was migrated to disk-based I/O servers. This enabled continuity of access to the data by users while the migration from tapes to disks was under way. Today, SLASH2 runs on several systems at PSC and elsewhere, including PSC's Bridges supercomputer, which serves over 800 client nodes with several petabytes of storage. The maximum number of files hosted by a single SLASH2 installation to date is well over 200 million.

SLASH2 has also run in production 1 with the University of Pittsburgh to connect geographically distant data centers. Other such connections have been made to other remote institutions such as Texas Area Computing Center (TACC) and the University of Wyoming, each creating a Wide Area file system over 1400 miles.

The rest of the paper is organized as follows. Section 2 reviews the architecture of SLASH2. Section 3 discusses the deployment of a SLASH2 file system to show its ease of use. Section 4 shares some lessons learned along the way. Section 5 outlines future work for SLASH2, and finally we conclude our paper in Section 6.

2. ARCHITECTURE

SLASH2 is a unique a wide-area network (WAN) file system that runs completely in the user space.

2.1 OVERVIEW

SLASH2 is composed of three different kinds of services: metadata server (MDS), I/O server (IOS), and client. A SLASH2 installation must have at least one MDS, one IOS, and one or more SLASH2 clients. It is recommended to have only one MDS for the entire deployment with at least one IOS at each participating site.

The MDS is responsible for managing namespace and enforcing data coherence in case a file is read and written by multiple clients at the same time.. To do its job, the MDS must store the attributes of files and directories. It also hands out leases to allow a client to access a file. The MDS also decides which I/O servers should provide the storage for a file. Figure 1 illustrates the relationship between the MDS, IOS and client:

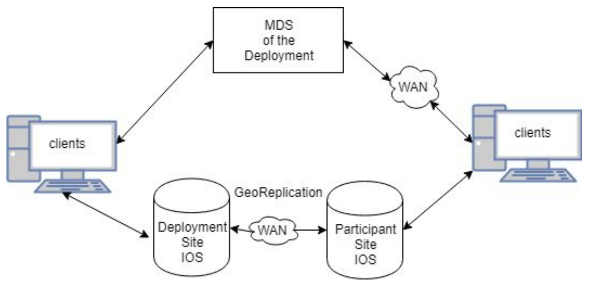

Figure 1. SLASH2 Architecture

To access a file, a client must send a request to the MDS to retrieve or create its attributes. After that, the client can communicate directly with an I/O server chosen by the MDS to read and write data for the file. As an object-based file system, I/O servers are free to allocate disk space for the file without any interference from either the MDS or the client. After a write is done, the I/O server will report the file disk usage to the MDS. If the file is later truncated or deleted, the MDS will send a request to the I/O server to reclaim the disk space used by the file. The MDS can also instruct two I/O servers (source and destination) to replicate data blocks of a file on behalf of a user request. All RPC requests in SLASH2 are checksummed to detect potential corruption during transmission and signed by a shared secret key.

In the following sections, we are going to discuss MDS, I/O server, and client in more details.

2.2 THE METADATA SERVER

In SLASH2, as in other distributed file systems, the metadata server (MDS) is the brain of the file system. It performs the following tasks in SLASH2:

• Namespace management. All namespace operations (e.g., creating a file, deleting a file, looking up a file, reading a directory) must go through an MDS. The MDS also stores attributes for all the files in a SLASH2 deployment such as the file size, modification time, etc. Multiple MDSes can cooperate to form a single namespace using the global mount feature [1].

• Storage management: The MDS stores the block map for all regular files, which means that a SLASH2 client must talk to an MDS before reading and writing any file. The MDS also decides which I/O server should receive new data based on the storage usage on each I/O server and client preferences.

• Lease management: In addition to handing out the storage map of a file upon request, the MDS also grants read or write leases to the requesting SLASH2 client. The lease is a contract between a client and an MDS that specifies which I/O server the client should contact for file I/O and for how long. Leases are managed on a per-block and per-file basis.

• Garbage collection: Whenever one or more block in a file is invalidated, the MDS will communicate with the relevant I/O servers to reclaim some storage blocks occupied by the file. Block invalidation can occur when a file is truncated or deleted. It can also happen because a more recent copy of the block has been stored on a different I/O server. The MDS must hold onto pending garbage collection requests because an I/O server may be unavailable for a while.

• Replication Management: A user can request to replicate blocks in a file to a specified I/O server. The replication engine in the MDS selects a source I/O server and asks the target I/O server to retrieve the file block directly from there. When the replication is done, the result is reported back to the MDS and the block map of the file is updated accordingly.

Overall, an MDS acts as a coordinator that does its best to delegate as much work as possible to other parties. In particular, it is not involved in the file data I/O path. To further reduce the load on an MDS, a SLASH2 client caches directory contents and leases provided by an MDS as long as possible. These strategies are also shared by many other distributed file systems. However, a key feature in SLASH2 that does not necessarily exist, at least in the same form or shape, in other distributed file systems is replication. This feature is designed to allow a user to have complete control of where to store his or her data among available I/O servers. It can be leveraged for various purposes:

• Local Performance: A client can arrange for its data to be replicated to a set of I/O servers that are geographically closer than other I/O servers for better performance. This can reduce the latency of I/O operations when SLSAH2 is used in a wide-area network.

• Data Protection: While each I/O server should run on a native file system with built-in RAID support, replication can add another dimension of protection against data loss. If a copy of a block is not available or is corrupted on an I/O server, it can be retrieved from another I/O server.

• Data Migration: If all data blocks on an I/O server have been replicated to other I/O servers, then that I/O server can be removed from a SLASH2 deployment and re-purposed.

The replication feature is very flexible and is completely driven by a user. A user can replicate a single file in its entirety. Or the user can replicate certain blocks within a file.

A replica of a block can also be added or removed at any time. The target I/O servers are selected by the user as well. Replication policies can be set on a directory and inherited by new files created in that directory. Note that the use of replication is optional. When all blocks in a file have exactly one valid copy, they can be spread across multiple I/O servers for better aggregate throughput.

2.3 THE I/O SERVER

The MDS can support a maximum of 65,536 I/O servers (IOS) to store actual file data. An I/O server can run on top of any file system that provides the POSIX file interface. In fact, all data used by an I/O server are stored under one regular directory. The underlying file system that provides the directory for SLASH2 can be a local file system (e.g., Ext4 and ZFS-on-Linux) or a distributed file system (e.g., Lustre).

The responsibilities of an I/O server are straightforward. First, it has to serve read and write requests from a SLASH2 client. Second, it has to serve a read request from a peer I/O server for replication purpose. Third, it has to process garbage collection requests from its MDS. Finally, it has to report to its MDS on its storage usage.

The I/O server is designed to be stateless, so that it can recover from a crash on its own. The data for a SLASH2 file is stored in a native file whose name and location are determined by the FID of the SLASH2 file.

An I/O server can refuse to accept new data when its storage usage has exceeded a certain limit (say 95% full). This is important because file system performance tends to drop when it becomes full. On the other hand, an MDS can be instructed to stop granting new write leases targeted to an I/O server. Both behaviors can be tweaked on the fly by a system administrator.

2.4 THE CLIENT

The SLASH2 client is a multi-threaded program that presents a POSIX interface to applications by mounting against an MDS. It is implemented as a FUSE file system using the low-level interface provided by the standard FUSE library.

The core logic of the SLASH2 client is actually encapsulated into a shared library that must be loaded into a generic file system client framework. As a result, we can unload the shared library at run time (after flushing all on-going operations) and reload a new version of the library. This opens the door for stackable processing and seamless upgrade in the future.

Historically, a FUSE file system is frowned upon because it incurs extra memory copy and context switch overhead. While we were fully aware of these arguments, we still chose to write a FUSE file system because user-space file system development is much more cost-effective. As an added benefit, we don't have to worry about compatibility issues each time a new kernel release is out. This reduces maintenance cost.

Finally, to make SLASH2 friendly in a wide-area network, a SLASH2 client has incorporated the following list of features:

• Delay and retry: A SLASH2 client stands ready to delay and retry its RPCs to ride out network glitches and server crashes. In addition, a SLASH2 client will not send out more RPC requests when there are already too many outstanding requests. The behavior of delay-and-retry and the RPC throttling mechanism can be tweaked with several runtime parameters.

• Configurable TCP keepalive and maximum segment size setting: ICMP messages are sometimes blocked by site security policy, which breaks Path MTU Discovery. In this situation, if the two end sites have an MTU mismatch, SLASH2 servers and clients cannot communicate efficiently.

• MDS routing: A SLASH2 client is able to communicate with the right MDS based on the site ID embedded in the FID of a file or a directory. This is used to support the global mount feature exported by the MDS.

• UID/GID mapping: A SLASH2 client can provide mapping between local and remote credentials to allow access to remote files from a client that lives outside of the administrative domain of the MDS. This feature is essential for SLASH2 to be used in a multi-administrative domain environment.

3. DEPLOYMENT

Since 2011, SLASH2 has been deployed on several systems at PSC and elsewhere [8]. These experiences, sometimes painful, have been valuable in improving the maintainability and the manageability of SLASH2.

3.1 SYSTEM CONFIGURATION

We have typically used commodity hardware to run SLASH2 servers and clients. On the MDS side, we configure the ZFS pool to be either two-way or three-way mirrored. On the I/O server, we use JBOD configured as a RAID (e.g., 8+3 ZFS RAIDZ3). In terms of operating system software, we have used FreeBSD and various Linux distributions including RHEL/CentOS and Ubuntu. Once everything is set up, a fresh installation or an update of SLASH2 software is as easy as the following steps shown in Figure 2:

```
$ git clone https://github.com/pscedu/slash2-stable
$ cd slash2-stable
$ make build
# make install
```

Figure 2 installation instructions

Additional details such as software package dependencies and configuration can be found on the "Quick Start" guide on the official project wiki [9].

In order for MDS, I/O servers, and clients to communicate with each other, a configuration file is used. The configuration file lists all the MDS and I/O servers in the system. The following Figure 3 shows a configuration file used by development:

```
set port=1000;
set net=tcp10;
set pref_mds="orange@PSC";
site @PSC {
    site_id = 0x123;
    site_desc = "SLASH2 @site PSC";
    fsuuid = 0x1234567812345678;
    resource orange {
    desc = "PSC MDS orange";
    type = mds;
    id = 0x11;
    nids = 128.182.99.28;
    journal = /dev/sdb; }
    resource lime {
        desc = "PSC IOS lime";
        type = standalone_fs;
        id = 0x22;
        nids = 128.182.99.27;
        fsroot = /local/lime/zhihui-s2;
    }
    resource lemon {
        desc = "PSC IOS lemon";
        type = standalone_fs;
        id = 0x25;
        nids = 128.182.99.26;
        fsroot = /local/lemon/zhihui-s2;
    }
    resource allios {
        desc = "all I/O servers";
        type = cluster_noshare_lfs;
        id = 0x1fff;
        ios = lime, lemon;
    }
}
```

Figure 3 Example SLASH2 configuration file.

To facilitate a large deployment, SLASH2 source code come with the following three shell scripts to start services at MDS, IOS, and client respectively: slashd.sh, sliod.sh, and mount_slash.sh. They can monitor the health of their respective services. In case of a crash, logs and core dumps are collected, emails are sent, and the service will be restarted automatically.

3.2 SLASH2 SPECIFIC TOOLS

All SLASH2 services run as a regular user space process on a machine. Therefore, all traditional Unix tools such as top, kill, ps, etc. can be used to manage them. In addition, SLASH2 provides three custom tools slmctl, slictl, and msctl to manage the MDS, the I/O server, and the SLASH2 client respectively.

These tools have the same look and feel and are very easy to use. They talk to their respective services through a local Unix socket. These tools can be used to check system status, collect I/O statistics, turn on and off some features, and much more. Man pages for these tools are available as well.

The following are just a few examples of how to run these tools (outputs have been tweaked to fit the format of this paper):

```
$ slmctl -p sys | tail -2
sys.uptime 23d22h51m
sys.version 42677
```

The above example shows that the MDS service has been up for more than 23 days and the software version is 42677, which is the number of git commits.

```
bash-4.2# ./msctl -s connection
resource host type flags stkvers txcr #ref uptime
=====================================
PSC
Orange orange.psc.edu mds -O- 42670 8 1 6d20h25m
lemon lemon.psc.edu local -O- 42677 8 1 4d01h28m
lime lime.psc.edu local -O- 42670 8 1 6d20h25m
```

The above command shows the service connection status from a client point of view. It is connected to an MDS and two I/O servers, all of them are at the site named PSC. Note that services do not have to run at the same version number, as long as they are compatible with each other in terms of RPC protocol. Also, they don't have to be brought online at the same time.

```
$ msctl –sop | grep peer
opstat avg rate max rate cur rate total
=======================================
peer-192.231.243.100 350B/s 10.9K/s 0B/s 196.4K
peer-192.231.243.101 38.7M/s 53.7M/s 30.7M/s 1.6G
```

The above command shows the peer connection I/O statistics for a client point of view, including the total number of bytes that have been transferred and three data transfer rates. The average rate is a running average rate, so it will go to zero after the connection is idle for a while. These tools have been integrated into the infrastructure at PSC to monitor the health of a SLASH2 file system.

4. LESSONS LEARNED

SLASH2 could not have reached its current solid state without the hard work and support of many people involved. In retrospect, there are some key lessons learned over the years.

First, exposure brings maturity. SLASH2 started out as an archive file system before it was used as a full-fledged file system subject to all kinds of workloads. Often a new use case exposes a bug or two in the code. Thankfully, over time, SLASH2 has been hardened enough to sustain our production use. Our conclusion is that using a file system in a production setting is necessary in order to bring out issues that arise in each new environment.

Second, keep it simple. Historically, we have implemented various clever algorithms or optimizations in SLASH2, even for some rare corner cases. They have been simplified or re-written over the years. Writing simple code really helps improve stability.

Third, user space development is the key to the success of SLASH2. It not only makes debugging much easier compared to kernel space development, but only affects our design and deployment decisions. A crashed daemon can be expected to restart quickly with the help of SLASH2 monitor scripts. It won't drag down the entire system, and clients can delay and retry their requests accordingly.

Fourth, wide-area networking is hard. Each service might drop its connection to another service at any given time. A RPC request can be arbitrarily delayed. So any service should learn to deal with a misbehaved partner. In addition to the technical challenges of networking, building a wide area file system will most likely involve numerous network engineering organizations in the deployment and debugging process. Our experience is that the largest part of a new deployment of a SLASH2 system is determining the networking topology and testing it for performance and resiliency. Monitoring the network must be an ongoing effort since changes in the network can have unintended consequences. We learned late in the project that having a network development engineer as a part of the project is very beneficial.

Fifth, maintainability is a priority. Many decisions, including user space development, using git version control, using monitor scripts, etc. are made to make deployment as easy as possible to reduce management overhead.

Last, when the file system spans administrative domains, there are challenges of mapping users from one domain to another. While these challenges have been met, there is a large effort involved in managing a file system that spans geographic domains.

5. FUTURE WORK

SLASH2 has been successfully employed in production for over 5 years. As of this writing, all major issues that could impact production use have been resolved. However, there are still plenty of work to do in the future.

The first item on our wishlist is to rewrite MDS to run on a kernel-resident file system instead of relying on ZFS-fuse, which is not maintained by anyone but us. This would make it easier to back up the data used by the MDS. By leveraging the open-by-file-handle feature present in recent Linux kernels, we can also improve the metadata performance.

We also want to leverage the write-back feature of FUSE file system. The I/O server can be expanded to access files in a cloud-based storage. Another work item is to complete the export and import feature that obviously can be used to transfer files between a SLASH2 file system and a non-SLASH2 file system.

6. CONCLUSION

In this paper, we have shared our experiences drawn from over 5 years of operating and improving SLASH2 in production HPC environments. We believe that SLASH2 has demonstrated its value in the scientific computing community because of its ability to logically bind existing storage systems – of different types and vendors – into a single POSIX file system. No special tools are necessary to access local and remote data.

In addition, by implementing everything in the user space, the cost of development and deployment is cut down significantly. This is critical because human cycles are becoming more valuable than machine cycles these days.

7. ACKNOWLEDGEMENT

First, the authors want to thank NARA and NSF for their support. Second, the authors want to thank the system administrators, especially Anjana Kar for her work over the years. We are also grateful for the detailed reviews by Kathy Benninger on early drafts of this paper.

SLASH2 is part of Pittsburgh Supercomputing Center's Data Exacell which is supported by National Science Foundation award number ACI-1261721, at the Pittsburgh Supercomputing Center (PSC). This grant additionally funded deployment and documentation efforts under the Research Experiences for Undergraduates with the University of Wyoming and Rocky Mountains Advanced Computing Consortium. SLASH2 applications are supported with the Extreme Science and Engineering Discovery Environment (XSEDE) under the National Science Foundation grant number OCI-1053575. We also acknowledge support by the National Science Foundation under Grant No. DBI-1458689.

8. REFERENCES

[1] John H Howard, An Overview of the Andrew File system, Proceedings of the USENIX Winter Technical Conference, Dallas, February 1988.

[2] Frank Schmuck and Roger Haskin, GPFS: A Shared-Disk File system for Large Computing Clusters, Proceedings of the FAST 2002 Conference on File and Storage Technologies, Monterey, January 2002.

[3] Philip Schwan, Lustre: Building a File system for 1,000-node Clusters, Proceedings of the Linux Symposium, Ottawa, Ontario, July 2003.

[4] Paul Nowoczynski, Nathan Stone, Jason Sommerfield, Bryon Gill, J. Ray Scott, Slash – The Scalable Lightweight Archival Storage Hierarchy, Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST'05), Monterey, April 2005.

[5] Paul Nowoczynski, Nathan Stone, Jared Yanovich, Jason Sommerfield, Zest - Checkpoint Storage System for Large Supercomputers, 3rd Petascale Data Storage Workshop, Austin, November 2008.

[6] John Bent, Garth Gibson, Gary Grider, Ben McClelland, Paul Nowoczynski, James Nunez, Milo Polte, Meghan Wingate, PLFS: A Checkpoint File system for Parallel Applications, Supercomputing '09, Portland, November 2009.

[7] Paul Nowoczynski, Jason Sommerfield, Jared Yanovich, J. Ray Scott, Zhihui Zhang, Michael Levine, The data supercell, Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment, Chicago, July 2012.

[8] Philip Blood, Anjana Kar, Jason Sommerfield, Beth Lynn Eicher, Rich Angeletti, and J. Ray Scott, Demonstrating Distributed Workflow Computing with a Federating Wide-Area File System, Practice & Experience in Advanced Research Computing Conference, New Orleans, June 2017.

[9] Beth Lynn Eicher. Quick Install Guide website, accessed on March 26, 2018: https://github.com/pscedu/slash2/wiki/Quick-Install-Guide.