File-Level Replication Scope Statement

Signed-off
This document was signed-off by OpenSFS on 11th November 2013.

Introduction

The following scope statement applies to the File-Level Replication Design project within the Technical Proposal by High Performance Data Division of Intel for OpenSFS Contract SFS-DEV-003 signed Friday 23rd August, 2013.

Problem Statement

The availability and resilience of a Lustre* file system currently relies entirely on the availability and resilience of its backend storage devices, servers, software, and network. Lustre software File-Level Replication mitigates this dependency by specifying a mirroring file layout for data across multiple OSTs so that file data remains available in the event of OSS or OST failure, for whatever reason. Data integrity can be checked and repaired by comparing mirrors.

The design will anticipate a phased implementation. In the **Phase 1 Delayed Replication** implementation, one or more replica copies will be created on otherwise idle files by a generic userspace replication utility with a configurable delay after the file is last modified (e.g within a few minutes). This will turn a non-replicated file into a replicated file, and will restore the required level of replication to files that have been modified or are stored on OSTs that are unavailable for an extended time. Phase 1 clients will have **read-only** access to any replica copy, and can resend reads to a different replica object if the original read fails or times out. In Phase 1, if a file is modified the replica OST objects will be marked **out-of-sync** on the MDT. Files that are not modified after initial creation (i.e. the large majority of all files) will therefore retain resilience after initial replication. Overwritten or appended files will regain replication when the replication utility is run on them again. A replication agent will be provided that uses the Lustre Changelog to detect and resync **out-of-sync** files efficiently. Integration with external policy engines such as Robin Hood is anticipated using the HSM coordinator and agents, but outside the scope of this project.

In the **Phase 2 Immediate Replication** implementation, modified files will be updated directly by the client for every write, first to the primary replica, and also to the backup replica(s). The backup replicas will be marked **stale** on the MDT upon creation or before the file is modified, and only the primary replica will be available for reads. After all writes have successfully completed to the backup replicas the **stale** flag can be cleared from the replica(s). If writes to the primary replica fails for some reason, the client(s) will mark it **out-of-sync**, flush all of the writes to the backup replica(s), and then select a backup replica as the new primary replica. In case writes to a backup replica fails, the clients will mark it **out-of-sync**. The Phase 1 replication utility will be used to resynchronize any **out-of-sync** replica(s). This will allow replication immediately on file creation, and replica copies will be updated immediately on overwrites and appends under normal use cases. This also allows concurrent asynchronous read/write operations without complex recovery mechanisms or introducing synchronous writes or barriers between updates.

Replica OST object allocation algorithms will be developed to ensure fault isolation between replicas, and will load balance replica generation over all available OSTs. This will ensure single hardware and software failures cannot affect multiple replicas and that repair speed after such a failure scales with aggregate OST bandwidth. The layout for replicated files will be stored on the MDT inode for the file and can be specified independently for each file as is done today. Existing default directory and file layout policy will be available to users to control the level of replication. The design of actual replicated layout format stored on the MDT is handled in the Layout Enhancement project.

Design Project Goals

- **Design of a delayed replication utility**. This will set or restore the requested level of replication on a set of Lustre files by allocating and initializing new replica OST objects and performing a layout switch to include them in the original file layout when required. The design will be targeted to idle files and abandon or restart replication if the file is overwritten.
- **Design of replica and fault isolation**. Configuration metadata will be added to assign OSTs to fault domains so that replica allocation can avoid single points of failure. Replica allocation will also ensure replicas are dispersed over all available OSTs to ensure replica regeneration on OST failure can proceed at full system bandwidth. This may be implemented through administrator-specified OST pools to isolate fault domains, and then replica file copies are allocated from separate pools.
- **Client I/O stack replica reads**. The client I/O stack will be altered in this design to read from one of the replica copies, and to retry reads from another replica when read RPCs to the primary OST return failure or time out.
- **Design of administration tools**. The `lfs setstripe` tool will be extended to allow specifying replicated files. A replication utility that implements a simple delayed replication policy triggered by the filesystem ChangeLog will be included to perform delayed replication for this design. The `lfs find` utility can be used to scan the filesystem namespace for files that have objects on a failed OST and invoke

the replication utility to repair them.

In-Scope

- Design of Phase 1 read-only RAID0+1 support.
- Design of dynamic page for IO.
- Design of command line tools to form RAID0+1 and remove replicas.
- Design of ChangeLog to record out-of-date replicas.
- Design of delayed replication utility.
- Design of synchronous replica layout invalidation when modifying an existing replicated file.

Out of Scope

- Fan out page will not be implemented.
- Immediate ordered replication: every write goes to the primary replica, and backup replica(s) (which are marked stale until all writes complete). Reads are only done from the primary replica.
- Immediate asynchronous replication: only the first write needs to return, the other takes place later. Recovery depends on DAOS (Fast Forward) mechanisms and is out of scope for this project.
- Replicated files will not be available to clients not supporting the LAYOUT lock.
- Replicated files *may* be available to clients supporting the LAYOUT lock but not supporting the replication feature, but is subject to the complexity of implementing this interoperability.

Project Constraints

- Layout enhancement work is complete.
- Jinshan is the preferred engineer for this work.

Key Deliverables

- Solution Architecture.
- High-level design document.
- Implementation assessment.

Glossary

Definitions for RAID configurations are recorded here: http://en.wikipedia.org/wiki/RAID

dynamic page

Dynamic pages are for read. In CLIO, a cl_page is composed of a top and a sub page. With dynamic page implementation, the sub page of a cl_page can be changed on the fly.
In the replication design, when the LOV receives a read request from LLITE, it will pick one replica (i.e. one OSC) out to serve the request. A sub page is created and attached to the top page and then the read request will be forwarded to the corresponding OSC. The RPC request may fail if the OSC can not connect to the corresponding OST. In this case, the RPC will be returned with an error and LOV will intercept this error and reselect another replica (i.e., another OSC) to satisfy this request. During this time, the sub page of the pages in the RPC will have to be changed so they can be requeued to a different OSC.

fan-out page

Fanout pages are for write. In the design phase 2, multiple replicas are written simultaneously. By design, a top page will have multiple sub pages and each sub page is attached to the corresponding replica (and OSC thereafter). A single top page can have multiple sub pages - a so-called 'fan-out page'.

*Other names and brands may be the property of others.