POHMELFS: Parallel Optimized Host Message Exchange Layered File System.

Evgeniy Polyakov <zbr@ioremap.net>

Homepage: http://www.ioremap.net/projects/pohmelfs

POHMELFS first began as a network filesystem with coherent local data and metadata caches but is now evolving into a parallel distributed filesystem.

Main features of this FS include:
 * Locally coherent cache for data and metadata with (potentially) byte-range locks.
        Since all Linux filesystems lock the whole inode during writing, algorithm
        is very simple and does not use byte-ranges, although they are sent in
        locking messages.
 * Completely async processing of all events except creation of hard and symbolic
        links, and rename events.
        Object creation and data reading and writing are processed asynchronously.
 * Flexible object architecture optimized for network processing.
        Ability to create long paths to objects and remove arbitrarily huge
        directories with a single network command.
        (like removing the whole kernel tree via a single network command).
 * Very high performance.
 * Fast and scalable multithreaded userspace server. Being in userspace it works
        with any underlying filesystem and still is much faster than async in-kernel NFS one.
 * Client is able to switch between different servers (if one goes down, client
        automatically reconnects to second and so on).
 * Transactions support. Full failover for all operations.
        Resending transactions to different servers on timeout or error.
 * Read request (data read, directory listing, lookup requests) balancing between multiple servers.
 * Write requests are replicated to multiple servers and completed only when all of them are acked.
 * Ability to add and/or remove servers from the working set at run-time.
 * Strong authentification and possible data encryption in network channel.
 * Extended attributes support.

POHMELFS is based on transactions, which are potentially long-standing objects that live
in the client's memory. Each transaction contains all the information needed to process a given
command (or set of commands, which is frequently used during data writing: single transactions
can contain creation and data writing commands). Transactions are committed by all the servers
to which they are sent and, in case of failures, are eventually resent or dropped with an error.
For example, reading will return an error if no servers are available.

POHMELFS uses a asynchronous approach to data processing. Courtesy of transactions, it is
possible to detach replies from requests and, if the command requires data to be

received, the
caller sleeps waiting for it. Thus, it is possible to issue multiple read commands to different
servers and async threads will pick up replies in parallel, find appropriate transactions in the
system and put the data where it belongs (like the page or inode cache).

The main feature of POHMELFS is writeback data and the metadata cache.
Only a few non-performance critical operations use the write-through cache and
are synchronous: hard and symbolic link creation, and object rename. Creation,
removal of objects and data writing are asynchronous and are sent to
the server during system writeback. Only one writer at a time is allowed for any
given inode, which is guarded by an appropriate locking protocol.
Because of this feature, POHMELFS is extremely fast at metadata intensive
workloads and can fully utilize the bandwidth to the servers when doing bulk
data transfers.

POHMELFS clients operate with a working set of servers and are capable of balancing read-only
operations (like lookups or directory listings) between them according to IO priorities.
Administrators can add or remove servers from the set at run-time via special commands (described
in Documentation/pohmelfs/info.txt file). Writes are replicated to all servers, which are connected
with write permission turned on. IO priority and permissions can be changed in run-time.

POHMELFS is capable of full data channel encryption and/or strong crypto hashing.
One can select any kernel supported cipher, encryption mode, hash type and operation mode
(hmac or digest). It is also possible to use both or neither (default). Crypto configuration
is checked during mount time and, if the server does not support it, appropriate capabilities
will be disabled or mount will fail (if 'crypto_fail_unsupported' mount option is specified).
Crypto performance heavily depends on the number of crypto threads, which asynchronously perform
crypto operations and send the resulting data to server or submit it up the stack. This number
can be controlled via a mount option.