Implementing Persistent Handles in Samba

Ralph Böhme, Samba Team, SerNet
2018-09-25
Outline

Recap on Persistent Handles

Story of a genius idea: storing Persistent Handles in xattrs

The long and boring story: dbwrap

Summary of implementation status
Recap on Persistent Handles
Recap on Persistent Handles

Persistent Handles part of higher level **SMB Transparent Failover**

- What is SMB Transparent Failover?
What is SMB Transparent Failover?

One of the key features in SMB 3.0

- enables transparent SMB3 failover with Continuously Available (CA) shares
- network or server failures are completely hidden from the application
- enables storing server application data (Hyper-V vhd) on SMB3 servers
What is SMB Transparent Failover?

One of the key features in SMB 3.0

- enables transparent SMB3 failover with Continuously Available (CA) shares
- network or server failures are completely hidden from the application
- enables storing server application data (Hyper-V vhd) on SMB3 servers

Continuously Available is a share capability:

- Continuously Available (CA) share: supports Persistent Handles
- per share option to enable Persistent Handles in Windows Server
What is SMB Transparent Failover?

One of the key features in SMB 3.0

- enables transparent SMB3 failover with Continuously Available (CA) shares
- network or server failures are completely hidden from the application
- enables storing server application data (Hyper-V vhd) on SMB3 servers

Continuously Available is a share capability:

- Continuously Available (CA) share: supports Persistent Handles
- per share option to enable Persistent Handles in Windows Server

Transparent Failover server requirements

- implement Persistent Handles
- replay detection for state changing operations
Persistent Handles semantics:

- file handle state must be preserved while a client is disconnected, across network and server failures
- surviving full cluster failure/reboot not "expected" though supported by Windows (some vendors don't this)
- while a client is disconnected, all state changing modifications from other clients must be blocked
What are Persistent Handles?

Persistent Handles semantics:

- file handle state must be preserved while a client is disconnected, across network and server failures
- surviving full cluster failure/reboot not "expected" though supported by Windows (some vendors don’t this)
- while a client is disconnected, all state changing modifications from other clients must be blocked

Persistent Handles on the wire

- new flag SMB2_DHANDLE_FLAG_PERSISTENT in Durable Handle v2 create context
Persistent Handles: example

- Hyper-V server connected to SMB3 cluster

![Diagram showing Hyper-V server connected to SMB3 cluster with file:disk.vhdx]
Hyper-V server opens shared virtual disk file
Hyper-V server has successfully opened vhdx
Persistent Handles: example, cont.

- SMB3 Cluster node crashes
Another client connects...
...and tries to open the same vhdx
Persistent Handles: example, cont.

- Server finds disconnected PH and fails open
Persistent Handles: example, cont.

- First client reconnects session and persistent file handle
Persistent Handles: example, cont.

- Second client retries open...
Persistent Handles: example, cont.

...and it succeeds this time
Persistent Handles: takeaway

Takeaway:

- store filehandle state on stable clustered storage or distribute it in memory to other nodes
- update open code to check for disconnected persistent handles
- go the full circle: replay detection
- done
I started thinking about how to tackle this about a year ago.

**Assumptions:**

- support Persistent Handles only for certain workloads (like MS):
  - workloads with minimal metadata overhead: Hyper-V, MS-SQL
- storing Persistent Handle can thus be slower then other file handles
- ignore problem of local access or via other protocols
I started thinking about how to tackle this about a year ago.

Assumptions:

- support Persistent Handles only for certain workloads (like MS):
  - workloads with minimal metadata overhead: Hyper-V, MS-SQL
- storing Persistent Handle can thus be slower than other file handles
- ignore problem of local access or via other protocols

Implementation ideas:

- somehow reuse existing Samba database backends
- this was presented at SambaXP 2018
- an update on this is presented in this talk
- another idea emerged: store PH state in xattrs
- the idea was too good to be true...
Story of a genius idea: storing Persistent Handles in xattrs
PH in xattr: genius idea
Genius idea to store PH in xattr:

- "all" operation that can affect PH state are path based
- when processing contending opens, fetch the xattr and check PH state
- when processing PH reconnect, use the path from the SMB request instead of the Persistent FileId
- the latter violates MS-SMB2, but it should be ok
What could possibly go wrong?
MS-FSA 2.1.5.14.11 FileRenameInformation.

If Open.File.FileType is DirectoryFile, determine whether Open.File contains open files as specified in section 2.1.4.2. If Open.File contains open files as specified in section 2.1.4.2, the operation MUST be failed with NTATUS_ACCESS_DENIED.
We got a problem:

- renaming a directory requires checking for open files underneath it
- Samba cheats here even without PH:
  - "strict rename = false" (default)
  - Samba only checks opens in the process doing the rename
- we shouldn’t cheat on this with Persistent Handles
- but to get it right would require traversing the filesystem
We got a problem:

- renaming a directory requires checking for open files underneath it
- Samba cheats here even without PH:
  - "strict rename = false" (default)
  - Samba only checks opens in the process doing the rename
- we shouldn’t cheat on this with Persistent Handles
- but to get it right would require traversing the filesystem

Flogging a dead horse:

- build a logical xattr tree rooted at the shared directory?
- forward pointers in xattrs to the next path component
- atomic update of the hierarchical tree?
- ...urks!
PH in xattr: git br -D ph-in-xattr
The long and boring story: dbwrap
Basic idea:

- provide per-record persistency semantics by combining volatile and persistent dbwrap backends
- db what?
dbwrap: db what?

What is dbwrap?

- Samba uses TDB databases to store various internal bits
- TDB is a fast key/value store
- shared memory mapped hashtable with chaining
- TDB is not clustered, so for clustering ctdb was invented
- a sane API was needed to abstract away locking details and non-clustered vs clustered usecase
- voilà: dbwrap: an API with backends (TDB, ctdb, . . .)
At the dbwrap API layer we implement two distinct modes of operation per database, selected when opening:

**Persistent:**

- enforces transactions, ACID, **slow**: store takes 100 ms
At the dbwrap API layer we implement two distinct modes of operation per database, selected when opening:

**Persistent:**
- enforces transactions, ACID, **slow**: store takes 100 ms

**Volatile:**
- no transactions, single key atomic updates, **fast**: few ms
- ACID without D:
  - the first opener wipes the db
  - looses all records on cluster reboot
- volatile model used heavily by smbd to maintain SMB and FSA layer state
Combine a volatile and persistent database:

- non-persistent records: unchanged behaviour
- store uses a new flag DBWRAP_PERSISTENT to request persistence record:
  - new ctdb control CTDB_CONTROL_PUSH_RECORD which pushes record to volatile dbs of all nodes
  - record stored as kind of backup in the persistent db
- first opener of a db restores records from persistent db to volatile db
- details are more complicated then this
The easy bits:

- Samba will always set `SMB2_GLOBAL_CAP_PERSISTENT_HANDLES`, supporting PH in clustered and non-clustered configs
- New per share option "persistent handles = yes|no" (default no) that optionally sets `SMB2_SHARE_CAP_CONTINUOUS_AVAILABILITY`
- Clustered Samba always sets `SMB2_SHARE_CAP_SCALEOUT` which implies active/active cluster
The easy bits:

- Samba will always set `SMB2_GLOBAL_CAP_PERSISTENT_HANDLES`, supporting PH in clustered and non-clustered configs.
- New per share option "persistent handles = yes|no" (default no) that optionally sets `SMB2_SHARE_CAP_CONTINUOUS_AVAILABILITY`.
- Clustered Samba always sets `SMB2_SHARE_CAP_SCALEOUT` which implies active/active cluster.

What to do about `SMB2_SHARE_CAP_CLUSTER`:

- `SMB2_SHARE_CAP_CLUSTER`: implies you’re a cluster and support Witness.
- We don’t support Witness, but according to MS that’s ok.
Persistent Handles have an associated timeout:

- assumption: ok to preserve longer than requested
- storing them in a persistent db on disk means we need a reliable scavenging
Persistent Handles have an associated timeout:

- assumption: ok to preserve longer then requested
- storing them in a persistent db on disk means we need a reliable scavenging

cleanupd to the rescue:

- enhance existing cleanupd who already does such stuff for brlocks
- SMB service processes (smbd) ask ctdb to send crash notifications if they crash
- cleanupd registers for SMB service process crash notifications
- cleanupd also registers for cluster topology change notifications
- on startup one cleanupd in a cluster is selected as the master cleaner
SMB service process (smbd) crash:

- cleanupd receives crash notification
scavenging: server failure types

SMB service process (smbd) crash:
- cleanupd receives crash notification

Cluster node crash:
- cleanupd receives cluster topology change notification
scavenging: server failure types

SMB service process (smbd) crash:
- cleanupd receives crash notification

Cluster node crash:
- cleanupd receives cluster topology change notification

cleanupd actions triggered by notifications:
- whenever cleanupd receives any of these notifications or becomes master it iterates over all PH and schedules scavenging of disconnected PH
Summary of implementation status
Implementation status

- dbwrap: 41 patches
- implement Persistent Handles ontop of dbwrap: ca. 90 patches
- diffstat: 101 files changed, 5572 insertions(+), 462 deletions(-)
- PH reconnect works
- protecting disconnected PH works
- cleanup works
- passes basic Persistent Handle test of MS Protocol Testsuite:
Microsoft Test Suite

Passes basic Persistent Handle tests in the MS Protocol Test Suite
To be done, part 1

- all patches still WIP
- exact open blocking semantics (stat opens, read-only opens)
- possibly weaken the strong on-disk persistence for faster performance
- record versioning for handling structure changes
- cluster generation id for manual cleanup of PH
To be done, part 2

- persist byterange locks
- merge create replay and reconnect:
  - currently uses two databases in the backend
  - also two implementations with overlapping functionality
- implement correct write time update semantics (bug #13594)
- eventually switch to ctdb implementation as presented this year at SambaXP
- add support for clustered Samba to Samba CI (autobuild)
- tests, tests, tests...
Demo (if time permits)
Thank you!

Questions?

Ralph Böhme <slow@samba.org>

SerNet -> Sponsorbooth
https://git.samba.org/?p=slow/samba.git;a=shortlog;h=refs/heads/ph-dbwrap
https://git.samba.org/?p=slow/samba.git;a=shortlog;h=refs/heads/ph-vfs
https://git.samba.org/?p=slow/samba.git;a=shortlog;h=refs/heads/ph-fsa
https://git.samba.org/?p=slow/samba.git;a=shortlog;h=refs/heads/ph-smb
https://git.samba.org/?p=slow/samba.git;a=shortlog;h=refs/heads/ph-cleanup
https://git.samba.org/?p=slow/samba.git;a=shortlog;h=refs/heads/ph-tests
https://wiki.samba.org/index.php/New_clustering_features_in_SMB3_and_Samba
https://docs.microsoft.com/en-us/windows-server/failover-clustering/sofs-overview