SDC 18

September 24-27, 2018
Santa Clara, CA

www.storagedeveloper.org

Implementing Persistent Handles in Samba

Ralph Böhme, Samba Team, SerNet

2018-09-25

Outline

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 Continiously Available (CA) shares
- network or server failures are completey 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 Continiously Available (CA) shares
- network or server failures are completey 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

SerNet

What is SMB Transparent Failover?

One of the key features in SMB 3.0

- enables transparent SMB3 failover with Continiously Available (CA) shares
- network or server failures are completey 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

SerNet

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

What are Persistent Handles?

Persistent Handles semantics:

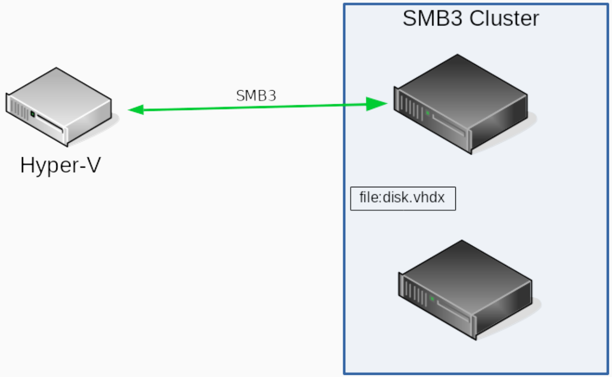
- file handle state must be preserved while a client is disconnected, across network and server failures
- surviving full cluster failure/reboot not "expected" though supported by Windows (some vendors don't this)
- while a client is disconnected, all state changing modifications from other clients must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

Persistent Handles on the wire

- new flag `SMB2_DHANDLE_FLAG_PERSISTENT` in Durable Handle v2 create context

Persistent Handles: example

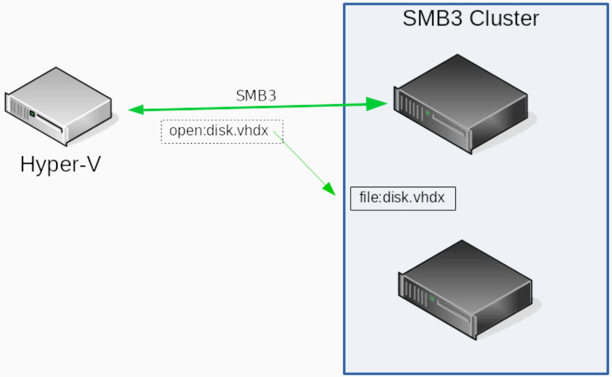
- Hyper-V server connected to SMB3 cluster



SerNet

Persistent Handles: example, cont.

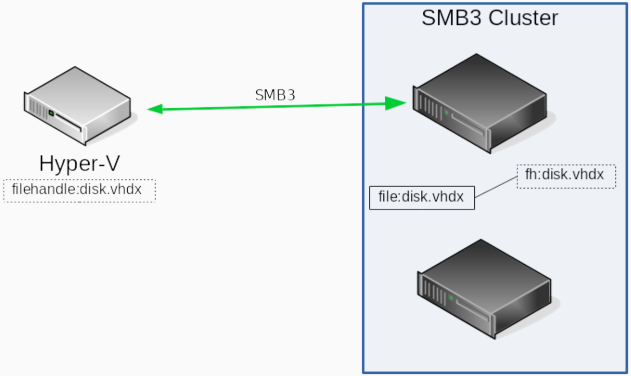
- Hyper-V server opens shared virtual disk file



SerNet

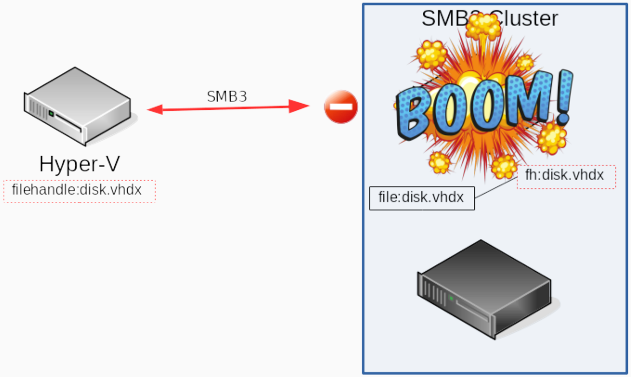
Persistent Handles: example, cont.

- Hyper-V server has successfully opened vhdx



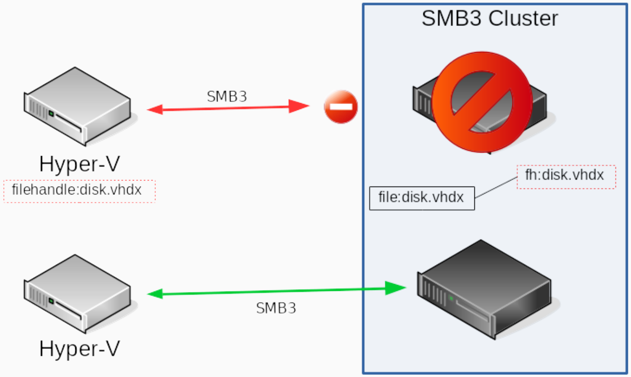
Persistent Handles: example, cont.

- SMB3 Cluster node crashes



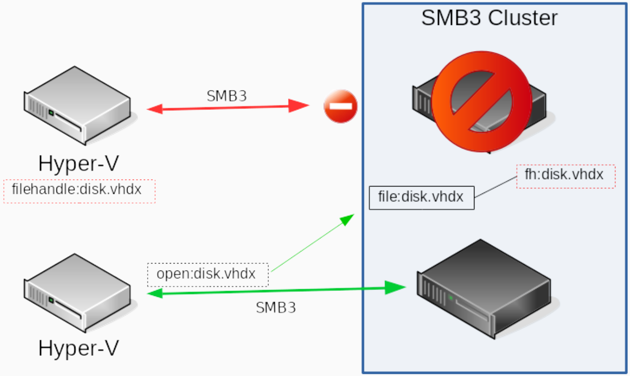
Persistent Handles: example, cont.

- Another client connects. . .



Persistent Handles: example, cont.

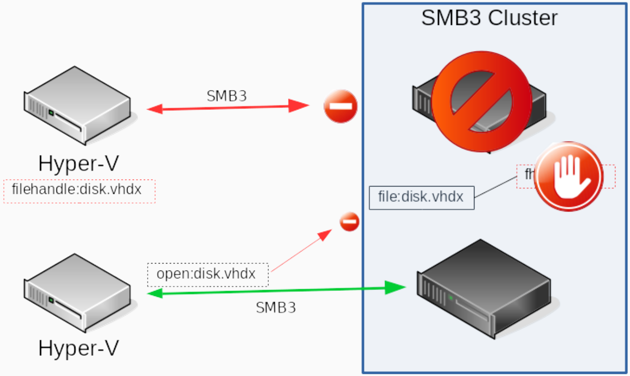
- . . . and tries to open the same vhdx



SerNet

Persistent Handles: example, cont.

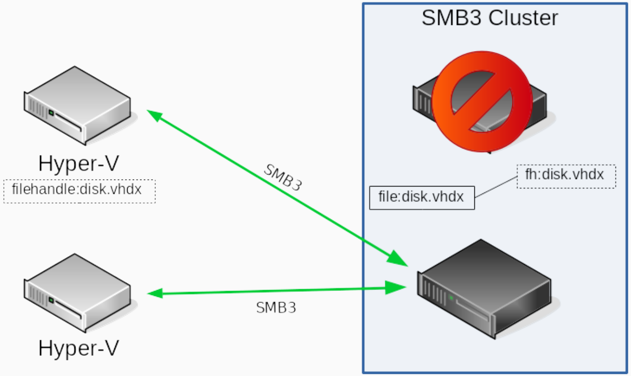
- Server finds disconnected PH and fails open



SerNet

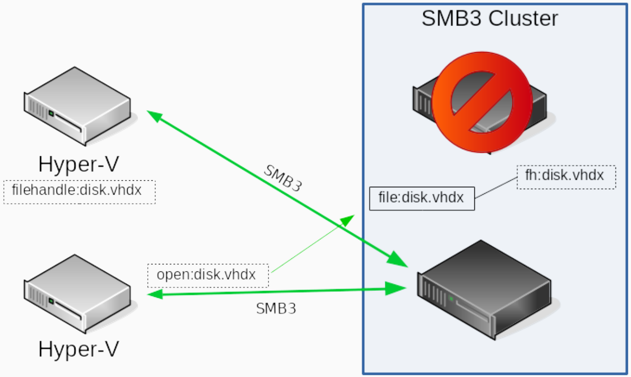
Persistent Handles: example, cont.

- First client reconnects session and persistent file handle



Persistent Handles: example, cont.

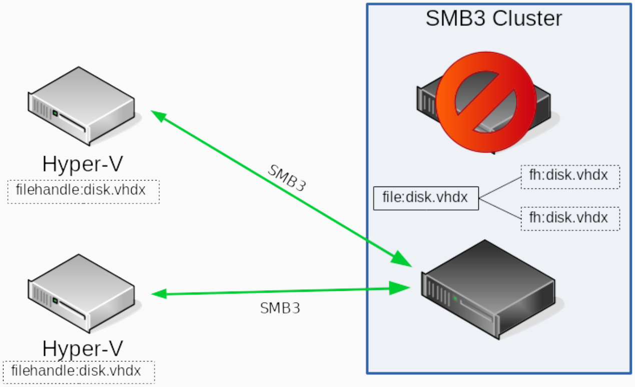
- Second client retries open. . .



SerNet

Persistent Handles: example, cont.

- . . . and it succeeds this time



SerNet

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

Persistent Handles and Samba

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

Persistent Handles and Samba

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

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 to good to be true. . .

Story of a genius idea: storing Peristent Handles in xattrs

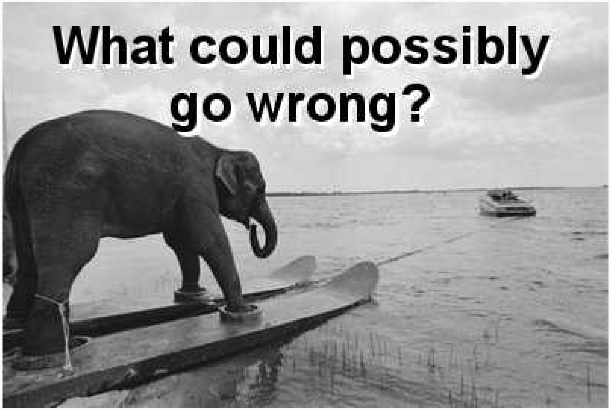
PH in xattr: genius idea

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

PH in xattr: problem alert



PH in xattr: problem alert

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. □

PH in xattr: genius idea, fail

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

PH in xattr: genius idea, fail

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

SerNet

The long and boring story: dbwrap

dbwrap: combine volatile and persistent dbs

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, . . .)

SerNet

dbwrap: clustered: volatile vs persistent

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

dbwrap: clustered: volatile vs persistent

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

dbwrap: bent to will

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

SMB3 Protocol Bits

The easy bits:

- Samba will always set `SMB2_GLOBAL_CAP_PERSISTENT_HANDLES`, supporting PH in clustered and non-clustered configs
- new per share option "peristent 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

SMB3 Protocol Bits

The easy bits:

- Samba will always set `SMB2_GLOBAL_CAP_PERSISTENT_HANDLES`, supporting PH in clustered and non-clustered configs
- new per share option "peristent 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

SerNet

scavenging: Persistent Handles timeout

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

scavenging: Persistent Handles timeout

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

scavenging: server failure types

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

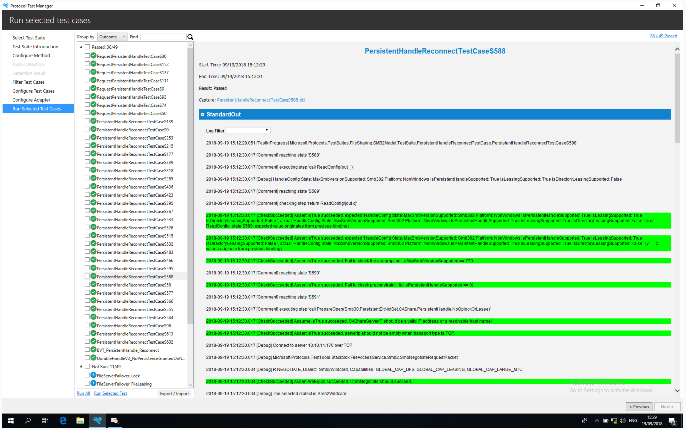
Summary of implementation status

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



```
Passed: 38/49
  RequestPersistentHandleTestCaseS30
  RequestPersistentHandleTestCaseS152
  RequestPersistentHandleTestCaseS137
  RequestPersistentHandleTestCaseS111
  RequestPersistentHandleTestCaseS0
  RequestPersistentHandleTestCaseS93
  RequestPersistentHandleTestCaseS74
  RequestPersistentHandleTestCaseS50
  PersistentHandleReconnectTestCaseS139
  PersistentHandleReconnectTestCaseS0
  PersistentHandleReconnectTestCaseS253
  PersistentHandleReconnectTestCaseS215
  PersistentHandleReconnectTestCaseS177
  PersistentHandleReconnectTestCaseS339
  PersistentHandleReconnectTestCaseS318
  PersistentHandleReconnectTestCaseS285
  PersistentHandleReconnectTestCaseS436
  PersistentHandleReconnectTestCaseS423
  PersistentHandleReconnectTestCaseS395
  PersistentHandleReconnectTestCaseS367
  PersistentHandleReconnectTestCaseS533
  PersistentHandleReconnectTestCaseS526
  PersistentHandleReconnectTestCaseS515
  PersistentHandleReconnectTestCaseS502
  PersistentHandleReconnectTestCaseS483
  PersistentHandleReconnectTestCaseS466
  PersistentHandleReconnectTestCaseS595
  PersistentHandleReconnectTestCaseS588
  PersistentHandleReconnectTestCaseS58
  PersistentHandleReconnectTestCaseS577
  PersistentHandleReconnectTestCaseS566
  PersistentHandleReconnectTestCaseS555
  PersistentHandleReconnectTestCaseS544
  PersistentHandleReconnectTestCaseS96
  PersistentHandleReconnectTestCaseS613
  PersistentHandleReconnectTestCaseS602
  BVT_PersistentHandle_Reconnect
  DurableHandleV2_NoPersistenceGrantedOnN...
Not Run: 11/49
  FileServerFailover_Lock
  FileServerFailover_FileLeasing
```

PersistentHandleReconnectTestCaseS588

Start Time: 09/19/2018 15:12:29

End Time: 09/19/2018 15:12:31

Result: Passed

Capture: PersistentHandleReconnectTestCaseS588.etl

StandardOut

```
2018-09-19 15:12:29.051 [TestInProgress] Microsoft.Protocols.TestSuites.FileSharing.SMB2Model.TestSuite.PersistentHandleReconnectTestCase.PersistentHandleReconnectTestCaseS588

2018-09-19 15:12:30.017 [Comment] reaching state 'S588'

2018-09-19 15:12:30.017 [Comment] executing step 'call ReadConfig(out _)'

2018-09-19 15:12:30.017 [Debug] HandleConfig State: MaxSmbVersionSupported: Smb302 Platform: NonWindows IsPersistentHandleSupported: True IsLeasingSupported: True IsDirectoryLeasingSupported: False

2018-09-19 15:12:30.017 [Comment] reaching state 'S589'

2018-09-19 15:12:30.017 [Comment] checking step 'return ReadConfig/[out c]'

2018-09-19 15:12:30.017 [CheckSucceeded] Assert.IsTrue succeeded. expected 'HandleConfig State: MaxSmbVersionSupported: Smb302 Platform: NonWindows IsPersistentHandleSupported: True IsLeasingSupported: True
IsDirectoryLeasingSupported: False ', actual 'HandleConfig State: MaxSmbVersionSupported: Smb302 Platform: NonWindows IsPersistentHandleSupported: True IsLeasingSupported: True IsDirectoryLeasingSupported: False ' (c of
ReadConfig, state S589; expected value originates from previous binding)

2018-09-19 15:12:30.017 [CheckSucceeded] Assert.IsTrue succeeded. expected 'HandleConfig State: MaxSmbVersionSupported: Smb302 Platform: NonWindows IsPersistentHandleSupported: True IsLeasingSupported: True
IsDirectoryLeasingSupported: False ', actual 'HandleConfig State: MaxSmbVersionSupported: Smb302 Platform: NonWindows IsPersistentHandleSupported: True IsLeasingSupported: True IsDirectoryLeasingSupported: False ' (v == c;
values originate from previous binding)

2018-09-19 15:12:30.017 [CheckSucceeded] Assert.IsTrue succeeded. Fail to check the assumption : v.MaxSmbVersionSupported == 770

2018-09-19 15:12:30.017 [Comment] reaching state 'S590'

2018-09-19 15:12:30.017 [CheckSucceeded] Assert.IsTrue succeeded. Fail to check preconstraint : !(c.IsPersistentHandleSupported == 0)

2018-09-19 15:12:30.017 [Comment] reaching state 'S591'

2018-09-19 15:12:30.017 [Comment] executing step 'call PrepareOpen(Smb30,PersistentBitNotSet,CAShare,PersistentHandle,NoOplockOrLease)'

2018-09-19 15:12:30.017 [CheckSucceeded] Assume.IsTrue succeeded. CAShareServerIP should be a valid IP address or a resolvable host name!

2018-09-19 15:12:30.017 [CheckSucceeded] Assert.IsTrue succeeded. serverIp should not be empty when transport type is TCP.

2018-09-19 15:12:30.017 [Debug] Connect to server 10.10.11.170 over TCP

2018-09-19 15:12:30.017 [Debug] Microsoft.Protocols.TestTools.StackSdk.FileAccessService.Smb2.SmbNegotiateRequestPacket

2018-09-19 15:12:30.034 [Debug] R NEGOTIATE, Dialect=Smb2Wildcard, Capabilities=GLOBAL_CAP_DFS, GLOBAL_CAP_LEASING, GLOBAL_CAP_LARGE_MTU

2018-09-19 15:12:30.034 [CheckSucceeded] Assert.AreEqual succeeded. ComNegotiate should succeed.

2018-09-19 15:12:30.034 [Debug] The selected dialect is Smb2Wildcard
```

SerNet

To be done

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

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

Demo (if time permits)

Q&A

Thank you!

Questions?

Ralph Böhme <slow@samba.org>

SerNet -> Sponsorbooth

Links

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