Persistent Handles: approaches

Ralph Böhme, Samba Team, SerNet

2018-06-08

Outline

Persistent Handles: Recap

Persistent Handles: Samba

dbwrap approach

ctdb approach

Persistent Handles: implementation (with dbwrap)

VFS approach

Outlook

The End

SerNet

Persistent Handles: Recap

Persistent Handles: design (Part 1)

Recap: SMB3 Persistent Handles, what for?

- SMB3 client opens file
- SMB3 server maintains file handle state (locks, sharemode, leases)
- now server crashes:
  - without Persistent Handles: state is lost
  - with Persistent Handles: server somehow persists state
- client is guaranteed to be able to reestablish file handle (within bounds / timeout)
- while client is disconnected, client is guaranteed that any concurrent access to the file is blocked

Persistent Handles: design (Part 2)

Todos for Samba:

1. Persist file handle state
2. Protect disconnected file handles from concurrent access

SerNet

Persistent Handles: use case

In theory any workload would benefit

- maintaining the handle state on persistent storage is expensive
- only recommended for workloads with low metadata overhead:
  - HyperV
  - MS-SQL
  - ...
- not recommended for information worker workloads (MS-Office)

Persistent Handles: Samba

Persistent Handles: Samba (Part 1)

Last year started research on possible designs:

- support Persistent Handles only for certain workloads, similar to what MS recommends
- storing persistent handle can be slower then "normal" file handles
- ignore problem of access via other protocols (!)
- Samba has a clustered db storage layer with strong persistency guarantees
- can't we somehow use that?

Persistent Handles: Samba (Part 2)

Basic idea was to combine a volatile and a persistent database:

- use a volatile db for non-persistent handles
- use a persistent db for persistent handles
- allow choosing persistency property per record based on a flag `DBWRAP_PERSISTENT` when fetching and storing
- two designs emerged:
  1. do it in dbwrap
  2. let ctdb do it

dbwrap approach

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 API can be tricky when it comes to locking
- TDB is not clustered, so for clustering ctdb was invented
- a sane API was needed to abstract away locking and non-clustered vs clustered usecase
- voilà: dbwrap: an API with backends (TDB, ctdb, . . .)
- dbwrap used by smbd

dbwrap: clustered: volatile vs persistent

Two distinct modes of operation per clustered database, selected when opening:

- persistent:
  - enforces transactions, ACID, slow
- volatile:
  - no transactions, single key atomic updates, fast
  - ACID without D:
  - the first opener wipes the db
  - looses all records eg on cluster reboot

dbwrap: handle state in volatile dbs

Samba uses a bunch of volatile databases for handle state:

- `locking.tdb`
- `smbXsrv_open_global.tdb`
- `brlock.tdb`
- `leases.tdb`
- remember: volatile dbs can loose records, not good for Persistent Handles

dbwrap: design (Part 1)

Opening the db:

- new flag to `db_open()`:
  DBWRAP_FLAG_PER_REC_PERSISTENT

dbwrap: design (Part 2)

Fetching records:

- new flag to `dbwrap_fetch_locked()`: `DBWRAP_PERSISTENT`
- always fetch-lock the record from the volatile db first
- while holding the lock, if caller passes `DBWRAP_PERSISTENT`, look into the persistent db
- return persistent record if found, otherwise return volatile record
- the volatile db serves as a distributed lock manager (DLM) on the persistent db
- `dbwrap_fetch_locked()` takes no low-level lock on the persistent db itself
- ensures concurrent `dbwrap_record_store()` don't deadlock in the transaction commit on the persistent db

dbwrap: design (Part 3)

Storing records:

- `dbwrap_rec_store()` also uses the new `DBWRAP_PERSISTENT` flag:
  - without `DBWRAP_PERSISTENT`: store in volatile db
  - with `DBWRAP_PERSISTENT`: store in persistent db
- when changing persistency property also delete from the db with the previous state
- ensures there's always only one record per key in either the volatile or the persistent db

dbwrap: keyspace (Part 1)

Ideally the keyspace of persistent and non-persistent records would be strictly disjoint:

- if a certain key will never be stored with `DBWRAP_PERSISTENT`, we could skip checking the persistent db
- would give unchanged db access semantics and performance for shares with `persistent handles = no`

dbwrap: keyspace (Part 2)

`locking.tdb` key: dev/inode

- the problem: admin configures two shares:
  - `[foo] path = /path , persistent handles = yes`
  - `[bar] path = /path , persistent handles = no`
- oh, my! Who would you do that?
- disconnected PH on a file in share `foo`
- clients would be able to access file via share `bar`

SerNet

dbwrap: keyspace (Part 3)

`smbXsrv_open_global.tdb` key: open global id

- use uneven numbers for non-persistent handles
- even numbers for persistent handles

SerNet

dbwrap: intersecting keyspace

If we must support intersecting keyspaces:

- just always pass `DBWRAP_PERSISTENT` to `dbwrap_fetch_locked()`
- small performance overhead for always looking into the persistent db
- could be made an option, defaulting to safe behaviour

ctdb approach

ctdb support (Part 1)

New database model with support for per record persistency (kudos to Amitay):

- `CTDB_CONTROL_DB_ATTACH_PER_REC_PERSISTENT`
- ctdb opens a volatile and a persistent db

ctdb support (Part 2)

Storing records:

- store volatile records only in the volatile db
- store persistent records first in persistent (as usual: on all nodes), then in volatile db

ctdb support (Part 3)

Fetching records:

- if we're not the DMASTER of a record:
- ask the LMASTER (as usual)
- if the LMASTER has no record for the key it checks the persistent db
- if he finds a record there, copy to volatile db and hand off to requester

ctdb support (Part 4)

Recovery:

- recover persistent db first (as usual)
- recovery of volatile db:
  - collect records from all nodes
  - update records from persistent db
  - and then push records to all nodes

Persistent Handles: implementation (with dbwrap)

Implementation status

- dbwrap: 37 patches
- patches for ctdb available from Amitay next week... :)
- implement Persistent Handles ontop of dbwrap: 103 patches
- diffstat: 109 files changed, 5128 insertions(+), 769 deletions(-)
- currently `locking.tdb` and `smbXsrv_open_global.tdb` are opened with the new model
- reconnect works
- protecting disconnected persistent handles should work :-)
- timeout and cleanup should work
- all patches still WIP
- TBD: byterange locks, record versioning in `locking.tdb`, tests, ...

Demo

Demo

VFS approach

VFS: approach

dbwrap (and ctdb) approach is quite heavyweight:

- persists more bits then actually needed
- took me some time to fully understand the implications of a particular Windows Scale-Out server behaviour:

VFS: Windows cheats

Windows cheats:

- Windows doesn't grant write or handle leases on a Scale-Out Cluster
- (btw: how does this work with SMB-Direct PUSH mode?)
- Scale-Out cluster: active/active cluster
- Failover cluster: active/passive
- Clustered Samba is Scale-Out
- this greatly simplifies the implementation

VFS: persistent parts

When processing `SMB2_CREATE`, check for disconnected PH:

- if there are any: fail with `NT_STATUS_FILE_NOT_AVAILABLE`
- no need for fancy lease break delaying/blocking
- can't the required state be stored seperately?
- ideally `locking.tdb` record becomes redundant
- `SMB2_CREATE` with `DH2C` context contains the path, so we could fetch the state from anywhere using the path as primary record key
- wait: path as primary key? That's a file...
- why not just tuck the state to the file as an additional xattr?

SerNet

VFS: IDL for xattr blob

```
typedef [public] struct {
        /* SMB layer bits */
        hyper           persistent_id;
        dom_sid         owner_sid;
        GUID            create_guid;
        uint32          durable_timeout_msec;

        /* FSA layer bits */
        server_id       id;
        uint32          access_mask;
        hyper           initial_allocation_size;
        uint32          private_options;
        timeval         time;
        uint32          access_mask;
        uint32          share_access;
} smbXsrv_ph;

typedef [public] struct {
        uint32                  num_phs;
        [size_is(num_phs)]      smbXsrv_ph phs[];
        [ignore] db_record      *record;
} smbXsrv_phs;
```

SerNet

VFS: new API

```
NTSTATUS SMB_VFS_PERSISTENT_STORE(struct vfs_handle_struct *handle,
                                  files_struct *fsp);

NTSTATUS SMB_VFS_PERSISTENT_CHECK_FILE(struct vfs_handle_struct *handle,
                                       struct smb_filename *smb_fname);

NTSTATUS SMB_VFS_PERSISTENT_RECONNECT(struct vfs_handle_struct *handle,
                                      TALLOC_CTX *mem_ctx,
                                      struct smb_request *smb1req,
                                      struct smbXsrv_open *op,
                                      const char *fname,
                                      files_struct **_fsp);
```

SerNet

VFS: using the new VFS functions

- when processing `SMB2_CREATE`, `DH2Q` triggers a call to `SMB_VFS_PERSISTENT_STORE()`
- call `SMB_VFS_PERSISTENT_CHECK_FILE()` in open_file_ntcreate() under the sharemode lock to block access to files with disconnected persistent handles
- when processing `SMB2_CREATE` `DH2C` use `SMB_VFS_PERSISTENT_RECONNECT()` instead of the durable handles reconnect functions
- simple so far, unfortunately ... (see next slide)

VFS: GlobalOpenTable problem (Part 1)

MS-SMB2 mandates:

MS-SMB2 3.3.5.9.12 Handling the DH2C Create Context.
The server MUST lookup an existing Open in the GlobalOpenTable by doing a lookup with the FileId.Persistent portion of the create context. □

VFS: GlobalOpenTable problem (Part 2)

So we still need a persistent `smbXsrv_open_global.tdb`

- could use the new dbwrap backend just for this
- or open an additional `smbXsrv_persistent_global.tdb` explictly
- then one million dollar question: could we do without?

VFS: GlobalOpenTable problem (Part 3)

Or we could just ignore `MS-SMB2 3.3.5.9.12`:

- use the path from the `SMB2_CREATE` reconnect to fetch xattr
- this way we wouldn't need to use any persistent db at all
- research needed how to deal with byte-range locks, could be stored in the xattr as well
- traverse filesystem to get a list of persistent handle xattrs is not practical
- that means no tool to list stored persistent handles xattr

Outlook

Outlook

- use dbwrap approach for prototyping
- use ctdb approach in the released version
- do more research on VFS approach, reconnect already works

The End

Q&A

- Thank you!
- Questions?

SerNet

Links

1. https://git.samba.org/?p=slow/samba.git;a=shortlog;h=refs/heads/ph-tests
2. https://wiki.samba.org/index.php/New_clustering_features_in_SMB3_and_Samba
3. https://docs.microsoft.com/en-us/windows-server/failover-clustering/sofs-overview

SerNet