Locking.tdb without locks?

SambaXP 2016
Berlin

Volker Lendecke

Samba Team / SerNet

2016-05-12

Small tdb intro

- tdb (Trivial (Tridge) Data Base) is a shared writer key-value store
- API similar to dbm
- tdb is implemented as a hash table with a linked list overflow
- Shared mmap with locks per hash list
- Optimized for heavy small read/write traffic
- Lots of tuning done in recent years
  - Freelist traffic reduced by dead records
  - Freelist fragmentation reduced
- You knew all this, right?

Locking.tdb in a nutshell

- Locking.tdb is (still?) our central open-file database
- It is very heavily contended
- Locking.tdb protects atomic opens/closes
    - create/setattr/setacl/unlink
- For open and close, a tdb record is locked
- brlock.tdb is locked while locking.tdb is locked
    - Two records locked simultaneously – deadlock?
    - DBWRAP_LOCK_ORDER maintains lock ordering
- Metadata operations are done while holding the lock
    - Unlink can take ages

dbwrap

- tdb is a low-level API
  - Exposes the hash chain structure ("tdb_chainlock")
- Really, really tricky semantics around locking
- Not aware of talloc
- We wanted clustering, tdb does not cluster, so:
  - All problems in computer science can be solved by another level of indirection, except of course for the problem of too many indirections.
- Implement a wrapper around tdb with the really needed features
  - dbwrap_fetch_locked() being the heart of it

g_lock

- ctdb can not provide clusterwide locks
- For persistent databases, we need to protect replication
- Simulate fcntl locks in user space
- g_lock_lock creates a record with the locker's PID as the only content
  - There's code for shared locks, but that was never used
- First implementation: lock waiters were added in an array
- Unlock sent messages to all waiters for retry

dbwrap_watch

- g_lock was the third place where someone waits for record changes
  - Oplock breakers waited for break or close
  - SHARING_VIOLATION 1-sec delay (or 5x 200msec: Hi, Chris :-))
- dbwrap_record_watch_send abstracts that
- dbwrap_watchers.tdb holds all waiters for any record in any db
- With dbwrap_watch_db(), every store to a database will trigger watchers
- Watchers typically wait for:
  - Lease break ack by client's smbd
  - g_lock unlocked by lock holder

Monitoring processes

- Watching a record ist mostly waiting for someone to do something
- What happens if that "someone" dies hard?
- Arbitrary processes need to monitor each other
  - SIGCHLD only works for direct children
- With unix datagram messaging every process holds a lockfile
  - fcntl wait for the lockfile to be given up?
- tmond and stream based messaging solves monitoring local processes
  - g_lock in current master just polls
- dbwrap_record_watch_send grew a "blocker" argument
  - dbwrap_record_watch_recv indicates blocker crash: EOWNERDEAD

Finally, dbwrap_nolock

- Double locks (locking.tdb and brlock.tdb) are bad
  - Gave Amitay a bad time for parallel database recoveries
- Cluster file systems can block smbd completely in D for a looong time
  - The file is dead, the others on the hash chain too :-(
- With mutexes, we lost /proc/locks
  - Diagnosis for contended locks more difficult
- dbwrap backend based on g_lock
  - A locked record holds the lock owner in the data field
  - Lock waiters use dbwrap_record_watch_send
- With mutexes, the noncontended case should not be much slower
  - Lock contention is worse, but that's bad already

Implementation details

- dbwrap_nolock is not exactly lockless
- Critical region under the lock is very small and confined
  - No file system operations under the lock
- Always locks two tdbs very briefly: Locking.tdb and dbwrap_watch.tdb
- The critical region ops could be delegated to a finite state machine
  - Persistent file handles anyone?
- Open issues:
  - Performance of course
  - Scalability with thousands of waiters – watchersd (like notifyd?)
  - Watching processes on remote nodes
- Demo time :-)

Questions?

vl@samba.org / vl@sernet.de