Clustered Samba Challenges and Directions

SDC 2016
Santa Clara

Volker Lendecke

Samba Team / SerNet

2016-09-20

SerNet

- Founded 1996
- Offices in Göttingen and Berlin
- Topics: information security and data protection
- Specialized on Open Source Software
- Samba: Windows/Linux interoperability, clustering and private cloud
- SAMBA+: Samba for Enterprise Linux
- verinice.: Open Source ISMS Tool
- Firewalls and VPN solutions for mid-size and large corporations
- Old economy: no venture capital, no loans

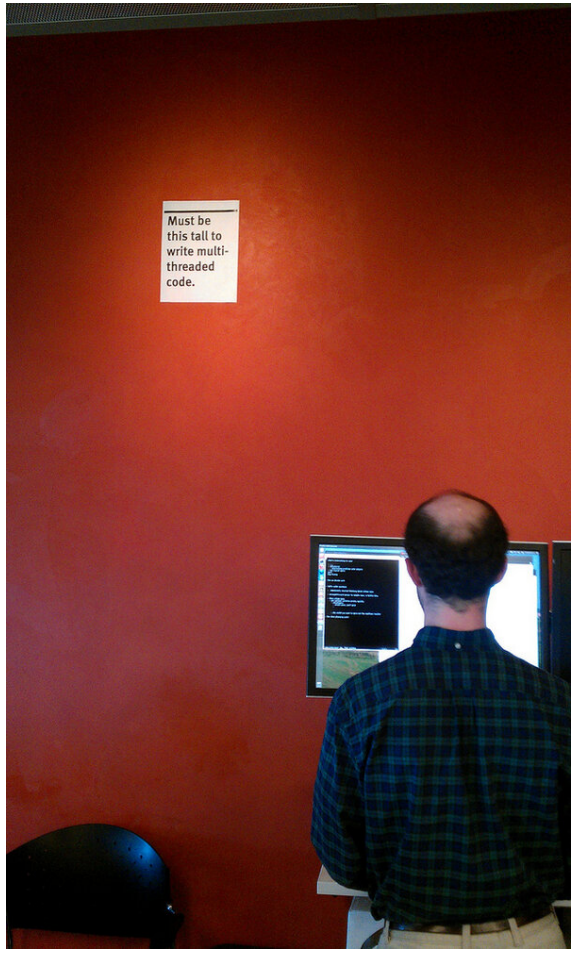
Overview

- Clustered Samba architecture
  - Samba as a classic Multi-Process daemon
  - Clustered TDB architecture
- Performance improvements
- Stability
  - File system slowness should not impact the cluster
- Database model changes

Samba architecture

- Traditional Unix architecture
- One listener process
  - Every client gets its own worker process
- Helper Threads for asynchronous I/O
  - Linux has no good general kernel-level aio
- Multi process single thread is vastly simpler than multi thread
  - Run-Down of structures is really hard
- Samba has to communicate: The oplock break
  - Process A needs to ask process B to release an oplock
- Architecture makes clustered SMB possible
  - Multi-process enforces IPC discipline
- Going more async: Notifyd, cleanupd

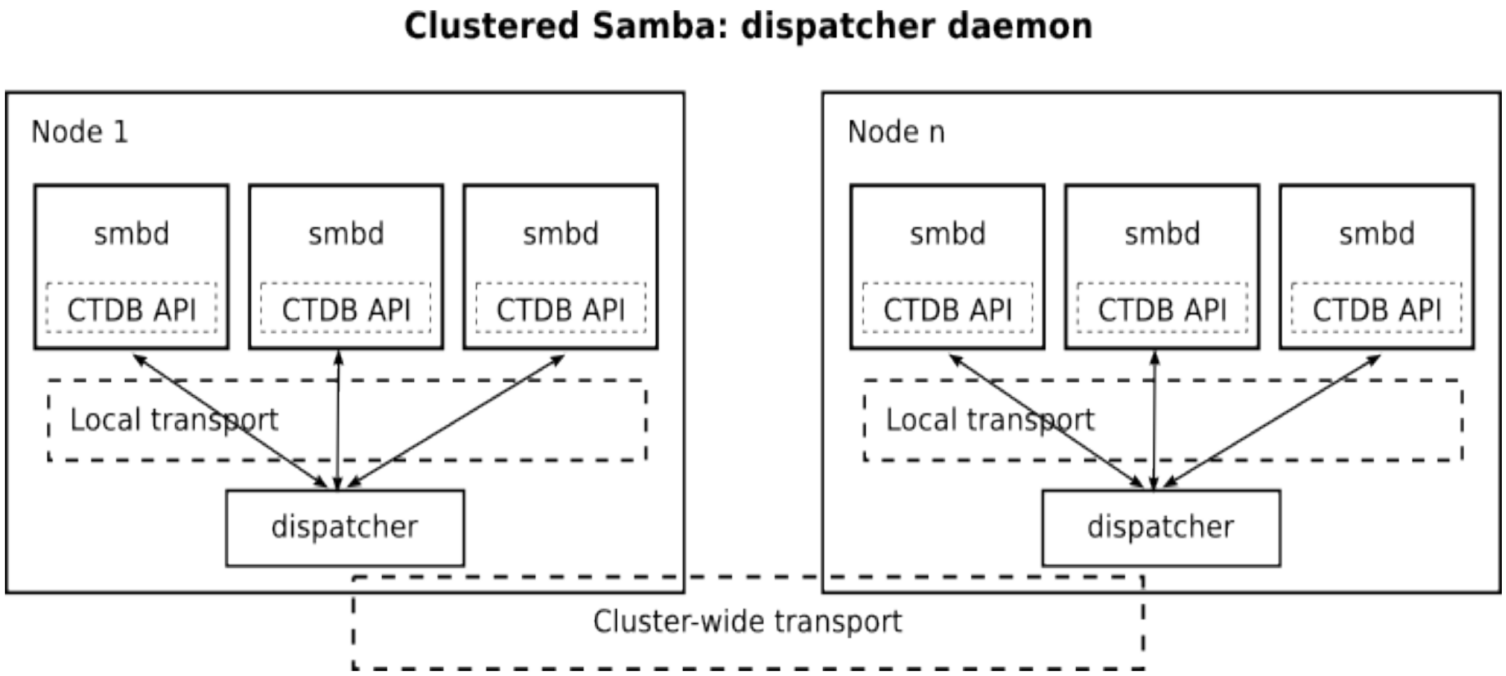
This Tall



locking.tdb

- Locking.tdb is Samba's central store for open file handle information
  - Share modes
  - Oplocks/Leases
  - File Disposition (delete-on-close)
- Most contended database in Samba
- Every open file handle ends up there
  - Recent scalability issue: phonebook.exe on Samba used by many thousands of clients
- tdb is a simple key/value multi-writer database
  - Uses mmap and shared mutexes
  - Well-tuned for many small write requests

ctdb


Clustered Samba: dispatcher daemon
Node 1
smbd
CTDB API
smbd
CTDB API
smbd
CTDB API
Local transport
dispatcher
Node n
smbd
CTDB API
smbd
CTDB API
smbd
CTDB API
Local transport
dispatcher
Cluster-wide transport

ctdb tasks

- Complete cluster manager
- Cluster Membership
- Service Monitoring
- Service IP management
- Ship tdb records back and forth (LMASTER/DMASTER roles)
- Replicate persistent database records everywhere (machine pwd)
- Samba messaging

ctdb performance improvements

- Remove tasks from ctdb
- Optional these days:
  - Service Monitoring
  - Service IP management
- Replace fork with vfork/exec for frequent tasks like the lock_helper
- Samba messaging
  - Every smbd connects to ctdb
  - Notifyd spreads file change notify
  - Simple, isolated task that a separate daemon will do
- Spread database management load
  - One LMASTER/DMASTER per database
  - Shard even further (hash per record key?)

locking.tdb scalability

- Every file open/close goes through locking.tdb
- One record per inode carries all share modes, leases, etc.
- Modern clients hold \\server\share directory handle open
  - Nonclustered Samba copes with it, although records get large
  - Clustered locking.tdb record becomes "hot", bouncing between nodes
- For the share root directory you might cheat
  - Assign per-node fake device number for \
  - No record bouncing, no cross-node share modes
- phonebook.exe still a problem
  - Split up locking.tdb into a per-node and a global component
- Only store the strictest share mode in ctdb
  - Keep individual handle's share modes local per node

Stability

- Open and close lock records in locking.tdb
- brlock.tdb locked simultaneously to locking.tdb record
  - Two records locked simultaneously – deadlock?
  - DBWRAP_LOCK_ORDER maintains lock ordering
- ctdb intercepts in locking
  - Previous deadlock now fixed
- Smbd does filesystem metadata operations while holding a lock
  - File systems need to take locks for those
  - In a cluster, there's too many locks to guarantee progress
  - Unlink can take ages (I've got a bug where unlink too 7 minutes)

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

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

Databases for persistent handles

- ctdb scales for independent workload, because it can loose data
  - When a node goes down, all files it holds are closed
  - locking.tdb records aren't valuable anymore
- Persistent file handles have different requirements
  - Node failover must retain data
  - Replicate persistent file information in the cluster
  - Something in between volatile (locking.tdb) and persistent (secrets.tdb) database model
- Where to replicate?
  - ctdb clusters are moderately sized so far, so a broadcast might be good enough initially

Questions?

vl@samba.org / vl@sernet.de