Clustered Samba Scalability Improvements

SNIA SDC 2018
Santa Clara

Volker Lendecke

Samba Team / SerNet

2018-09-26

Samba architecture

- For every client Samba forks a new process
- Distinct memory spaces in every process
- MS-SMB2 and MS-FSA suggest a lot of shared tables
  - Lists of clients, tree connects, open files
- Samba can't use any of those data structures directly
- Samba shares data structures via shared key/value stores
  - TDB is a memory-mapped hash table
  - Protection via fcntl locks or shared mutexes
- TDB provides a clean separation layer

SMB history

- SMB semantics date back to DOS single-user OS
  - Every application by definition had exclusive file access
- SHARE.EXE maintained illusion by blocking concurrent access
- Network-aware applications could explicitly permit sharing
  - Different modes of access permitted on a per-open basis
- Posix opens only have to read metadata
  - Permissions, file location etc
- Inherent scalability problem through share modes
  - SMB opens need to examine all other opens

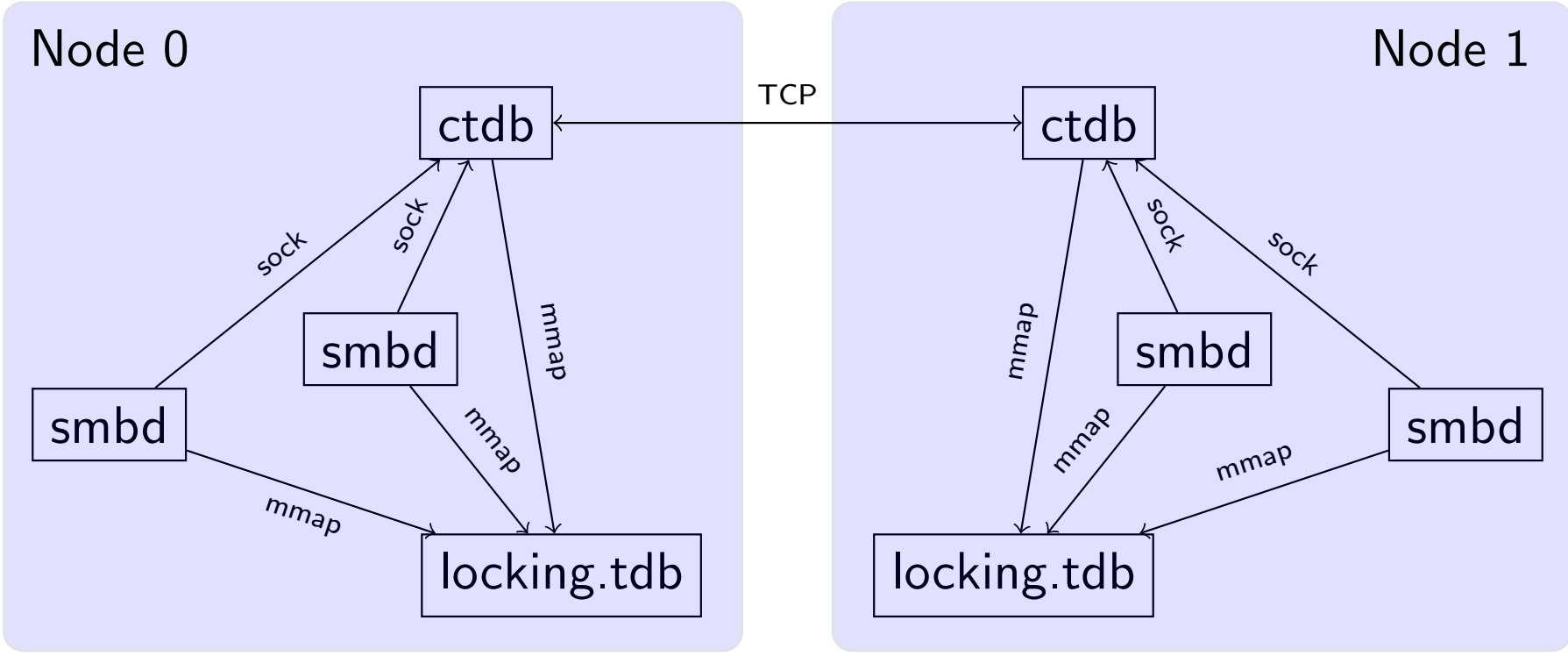
SMB share modes

- Every open call requests access permissions
  - READ, WRITE or DELETE (among others)
- Every open call allows other permissions
  - Concurrent READ, WRITE or DELETE permitted
- First come, first serve
- All open handles are entered in a central table
- struct share_mode_entry:
  - uint32_t share_mode
  - uint32_t access_mask
- Samba stores an array of those per inode in locking.tdb

Clustered TDB ctdb

- ctdb extends tdb files beyond a single machine
- ctdbd is a daemon to move records around
  - smbd requesting a record gets a local copy
  - ctdb maintains the most recent record location
- locking.tdb can be lossy
  - Share mode state valid only for open file handles
  - A crashed node's file handles are closed by definition
- ctdb record access is like NUMA with extreme node distance

ctdb Architecture



locking.tdb scalability

- Samba's design scales nicely on homedir workloads
  - Mostly exclusive access to many files
- Heavily contended files are a problem
- Customer use case:
  - 15,000 users accessing the same exe file simultaneously
- Every open call needs to check 15.000 share mode entries
- Nonclustered Samba handles this nicely
  - However, nonlinear processing time per open

Optimizations

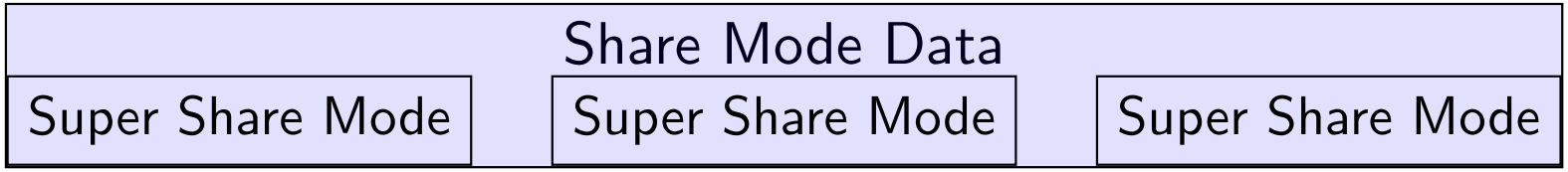
- Don't walk the whole list on every open
- Maintain a central "most restrictive share mode"
- share_mode:
  - Store the most restrictive share mode handed out
- access_mask:
  - Store the superset of all current access masks
- Lazy update:
  - More restrictive open request: update central record
  - When closing, don't update: We'd have to check all other entries
- At open time: Just check the central record
  - Only at conflict time, walk the whole list
- This optimizes the massive non-conflict case

Per-Node share mode lists

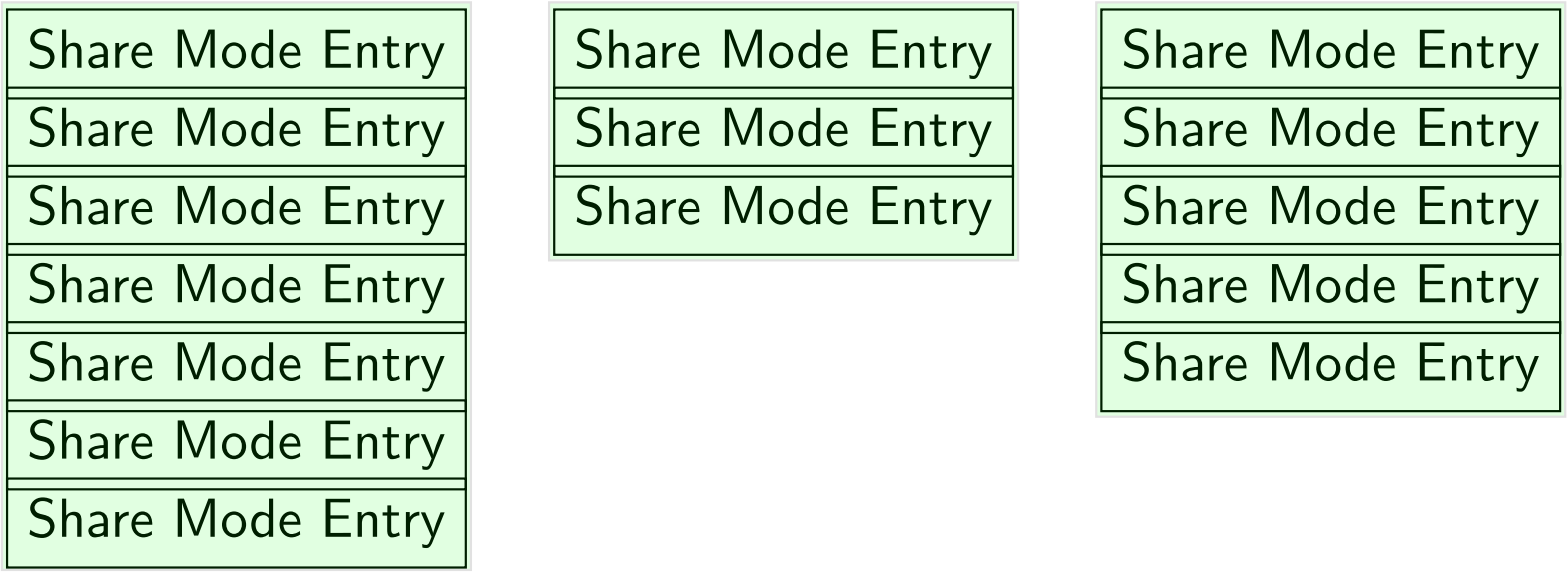
- Bouncing 15k share mode entries per open
  - ctdb melts down under this load
- One central share mode entry per node
- Every node maintains its own central entry
- Central record remains small
- The list of share entries is maintained separately
- Two databases: locking.tdb and share_entries.tdb

Multiple Nodes

locking.tdb



share_entries.tdb



But what about leases?

- Looking at open_files.idl there's two arrays
- Share modes can be split into share_entries.tdb
- Every share entry corresponds to a lease entry
  - lease entries are shared
- Where to store the lease entries?
- Leases can be used across different TCP connections
  - Lease information is stored in leases.tdb
  - leases.tdb indexed by client guid and lease key
- All lease information from locking.tdb moved to leases.tdb
  - leases.tdb needs serious micro-optimization

Current Status

- Remove leases from locking.tdb: Done
- Central share_mode_union: Work in progress
- Multiple share mode arrays: To be done
- Open problem: Cleanup of share mode data
  - Lazy close keeps share_mode_unions around
  - When a share mode array drops to 0, look at the others?

Questions?

vl@samba.org / vl@sernet.de

http://www.sambaxp.org/