Status of Clustered CIFS using Samba

Volker Lendecke

vl@samba.org

SerNet / Samba Team

2010-09-15

Outline

Who am I?

- Co-founder SerNet - Service Network GmbH
- Free Software as a successful business model
- Network Security for the industry and the public sector
- Samba-Support/Development in Germany
- For almost 20 years concerned with Free Software
- First patches to Samba in 1994
- Consultant for industry in IT questions
- Co-founder emlix GmbH (Embedded Systems)

SerNet

- SLA based support for more than 650 customers
- network security for industrial and public customers
- firewalls, VPN, certificates, audits
- based on open standards wherever possible
- Support for many OS: Linux, Cisco IOS, Windows etc.
- Compliant with BSI Grundschutz and ISO 27001 and other international regulations

SerNet and Samba

- technological leadership of SerNet worldwide
- involved in almost every big European Samba project
- 5 out of 6 European developers work for SerNet
- SerNet distributes up-to-date Samba packages
- samba eXPerience
- The international Samba conference
- $>$ 150 developers & users from $>$ 15 countries

Clustering CIFS

- The easy part: Active/Passive clusters
- The holy grail: Share the same clustered file system via different Samba nodes, and scale linearly
- NFS is relatively easy: No locking around
- GFS/GPFS do it for Posix semantics
- Samba needs to fake SMB semantics, which are particularly hard to get right and fast

SerNet

SAMBA

CTDB

- None of the existing lock managers provided the semantics needed for CIFS clustering
- Samba requires locks with associated data, a big share of Samba is to implement the correct locking
- Many lock managers are much too slow
- Ctdb is the clustered tdb lock manager
- ... and also does IP failover, service start/stop, monitoring, TCP tickle acks, etc

Posix vs CIFS semantics

- Unix historically (VERY early) did not have proper IPC
- Applications had to cope with very few atomic operations like creating a file
- Locking until today is very rudimentary
- SMB/CIFS was designed for compatibility
- Existing single tasking applications needed to be protected against each other
- Share Modes provide exclusive access to open files

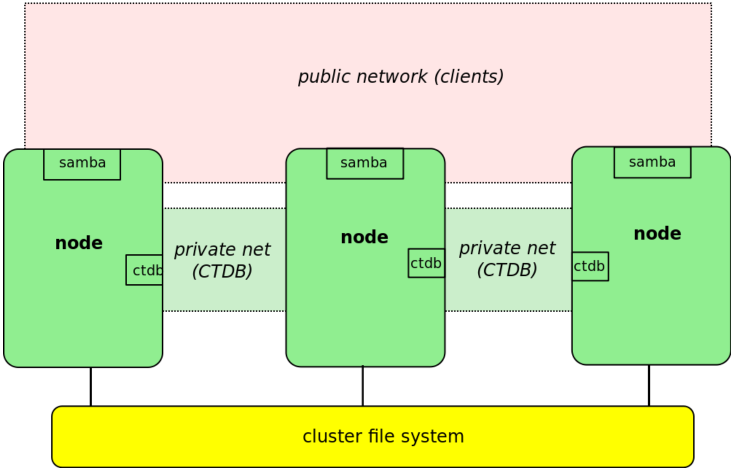
Share Modes

- On Posix, opening a file is an isolated operation that no other cluster node needs to know about
- Easy to get fast in a cluster file system
- Modifying a directory (create, rename, unlink) requires exclusive access, this gets slow
- For CIFS, every file open needs to be communicated
- Does another node have a conflicting file open mode?

The ctdb trick

- Samba uses a trivial database to hold file open information
- Ctdb is clustered tdb
- Traditional clustered databases provide very high consistency guarantees $\Rightarrow$ several orders of magnitude too slow for Samba
- CTDB can lose data!
- If a node dies, the files held by that node are closed by definition

CTDB architecture



Other CTDB tasks

- Complete HA solution
- Cluster membership detection via a byte range lock on a shared file
- IP Address takeover
- Service control
    - Start, stop, monitoring of services
- TCP tickle acks
    - Necessary for fast recovery of Windows clients, they might be stuck in a state where they wait for data from a dead node
- Why did we not use a classic HA solution like Linux-HA or others?
    - Not invented here?
    - CTDB as a HA solution was not planned, it just “happened”.
    - TCP tickle acks were not readily available (are they today?)

Persistent databases

- Not all tdb files are as highly dynamic as locking db's are.
- secrets.tdb: Domain membership
- registry.tdb: Holds Samba version of registry
- Persistent tdb's are handled differently: Everyone has an up-to-date copy, writes are broadcast

Registry configuration

- Parsing and writing smb.conf files with GUI tools is awkward at best
- Samba 3 has to implement a registry, clients expect to find certain keys to determine the server type
- Registry data model matches exactly smb.conf format, it was designed as a .ini file replacement
- HKLM/Software/Samba/smbconf
- Enabled only if `config backend = registry` is enabled in the smb.conf text file

Challenges

- File system assumptions
- Persistent tdb transactions
- CLEAR_IF_FIRST is gone
- Scalability of `brlock.tdb`
- Long-running system calls

File system assumptions

- Samba/ctdb does not use the underlying file system for its own operations
    - ctdb used to depend on an fnctl lock on a file stored in shared storage for split brain detection
    - Changing the reclockfile requires a global ctdb restart
    - The filesystem holding the reclockfile can not be unmounted
    - Can run without the central reclockfile, provided hooks into the cluster exist for split brain detection
- Tridge's ping-pong test checks fcntl lock coherence
    - With -rw command line switch it checks that writes are properly propagated
    - To safely store CIFS data on clustered Samba, -rw must pass

Persistent tdb transactions

- Persistent tdb's are mostly read, writes need to, well, persist
- Tdb can do transactions using proper fsync calls
- Changes to clustered tdbs need to be written everywhere
- Fail and retry turned out to be not successful
- Global lock required, which ctdb does not provide
- New tdb `g_lock.tdb` provides semantics similar to fcntl cluster-wide
- Transaction writes are broadcast under a lock
- Inconsistencies are cleaned up by cluster recovery: Last writer wins

CLEAR_IF_FIRST is gone

- `locking.tdb` and others store per-file / per-process information
  - Representation of locked files, with the PID of the smbd holding locks
- Wrapping PIDs (16-bit PID space still common) make files locked forever when a smbd crashes
- CLEAR_IF_FIRST is a tdb mechanism to wipe databases at startup
- At smbd startup all PID-based information is gone
- With ctdb around also accessing the tdb, restarting smbd does not wipe tdbs
- New tdb `serverid.tdb` holds a random 64-bit ID per active process

Scalability of `brlock.tdb`

- Posix byte-range locks are advisory, CIFS expects mandatory locks
  - Every read/write request needs to access `brlock.tdb`
- Popular use case: Read a 10GB file from multiple clients
- Right now the way ctdb is written, we play ping-pong with a `brlock.tdb` record
  - Scalability of reading 10GB is less than optimal
- Plan: "level II oplocks" on ctdb records
  - Multiple nodes hold a r/o copy of the `brlock.tdb` record holding the locks for a file
  - Changes to that (i.e. Locking&X calls) call back that cache

Long-running system calls

- Cluster file systems can be *very* slow sometimes
    - Node failures will trigger recoveries, cleanups, etc.
    - Posix API is synchronous. Open, unlink etc can take ages.
- Samba does Posix calls while holding a lock on `locking.tdb`
- Nodes being (possibly partly) stuck can block regular ctdb operations
    - ctdb walks tdb files for cleanup and recoveries after node failures
    - smbd locking them can cause nice deadlocks $\Rightarrow$ nodes get unhappy
- Samba could be changed to never do Posix calls under tdb locks
    - Performance penalty?
    - Fcntl lock cleanup semantics is lost
- Windows clients time out after 30 seconds (or so...)
    - Async echo handler helps, but even a fully multi-threaded server will will eventually make Windows clients unhappy

Questions?

Thank you very much!
vl@samba.org

SerNet
SAMBA