Clustering Samba With CTDB
A Tutorial At sambaXP 2010

Michael Adam

obnox@samba.org

SerNet / Samba Team

2010-05-05

Outline

1. Cluster Challenges
   - Introduction
   - Challenges For Samba
2. CTDB
   - The CTDB Project
   - CTDB Design
   - Setting Up CTDB
3. Clustered Samba
   - Getting Sources and Binaries
   - Clustered File Systems
   - Samba Configuration
   - CTDB manages...
   - Registry Configuration

Ideas

- quite common: clustered web servers and database servers...
- idea: share a cluster file system as a network service (NFS/CIFS)
- i.e. turn your SAN into a *clustered* NAS
- ⇒ we want to cluster Samba/nfs in an all-active fashion
- with CTDB, we *can* cluster Samba (and nfs, and ...)

SerNet

SAMBA

Starting Points

- Samba daemons on cluster nodes need to act as *one* CIFS server:
  - consistent view of file ownership
  - windows file lock coherence
- hence we need IPC of Samba daemons between nodes
- furthermode share some persistent data

SerNet

SAMBA

Challenges For Samba

- IPC: messaging (`messages.tdb` and signals)
- IPC: share volatile session data:
  - SMB sessions (`sessionid.tdb`)
  - share connections (`connections.tdb`)
  - share modes (`locking.tdb`)
  - byte range locks (`brlock.tdb`)
- share certain persistent data:
  - user database (`passdb.tdb`)
  - domain join information (`secrets.tdb`)
  - id mapping tables (`winbindd_idmap.tdb`)
  - registry (`registry.tdb`)

TDBs

- most problems are about distributing TDBs in the cluster
- TDB: small fast Berkeley-DB-style database with record locks and memory mapping
- volatile ("normal") TDBs:
  - read and written very frequently
  - not all data must be known to every node (or `smbd` process) at each point in time
  - R/W performance critical for overall fileserver performance
  - especially important for the Windows locks
- persistent TDBs:
  - read frequently
  - written rather rarely
  - data consistency very important

TDBs And Clustering

- TDB R/W performance critical for Samba performance
- TDB R/W operations: excessive use of POSIX `fcntl` byte range locks
- `fcntl` locks are usually slow on cluster file systems
- the more nodes, the slower...
- ⇒ naive approach of putting TDBs on cluster storage works in principle but scales *very badly*
- Usual clustered data bases are also too slow.
- A more clever approach is needed.

SerNet
SAMBA

Goals

- Cluster Samba So That:
  - One node is not slower than an unclustered Samba server.
  - $n+1$ nodes should be faster than n nodes.
- This in requires a clustered TDB implementation ...
- ... and messaging solution.
- This is what CTDB provides.

SerNet
SAMBA

The CTDB Project

- started in 2006
- first prototype in `vl-messaging` SVN branch
- Volker Lendecke, Andrew Tridgell, ...
- first usable version of CTDB: April 2007
- meanwhile: Ronnie Sahlberg project maintainer
- `git://git.samba.org/sahlberg/ctdb.git`
- `http://ctdb.samba.org/packages/` (RPMs, Sources)

SerNet

The CTDB Project - Relases

- to be honest: There is no real release process.
- version number and changelog in `packaging/RPM/ctdb.spec.in`
- version in the master branch is incremented more or less frequently
- some versions stabilize in extra branches: 1.0.69, 1.0.82, 1.0.108, 1.0.112, ...
- Hint: packagers better check with developers for advice on versions!

SerNet

The CTDB Project - Community

- #ctdb channel on freenode
- samba-technical mailing list
- feedback and contributions by packagers
- increasing development activity, number of developers

SerNet
SAMBA

CTDB Design - Warning

A Word Of Warning

- Client connections are *not* spread over multiple cluster nodes.
- I.e., each single client connection (CIFS, nfs, ...) is serverd by one node just as a non-clustered file server would server the connection.
- Hence a single connection is not faster than on a non-clustered file server, but the sum should (possibly) be faster.
- In case of failover, connections are not migrated: clients need to reconnect.

SerNet
SAMBA

CTDB Design – General

- one daemon `ctdbd` on each node (and temporary forks)
- `smbd` talks to local ctdbd for messaging and TDB access
- `ctdbd` handles metadata of TDBs via the network
- `ctdbd` keeps local TDB copy (LTDB) for fast data reads/writes
- the actual record read and write ops are directly to the LTDB
- normal and persistent TDBs are handled differently
- HA and cluster management features: monitor and fail over/back IP addresses and Samba, NFS and other services

CTDB Design – normal TDBs

- one node does not need to know all records all the time:
- the records related to connections to a node are node specific
- when a node goes down:
- ⇒ we may, even *should* lose records specific to that node
- a node only has those records in its LTDB that is has already accessed

CTDB Design - Record Roles

- nodes can carry certain roles with respect to a record:
- DMASTER (data master):
  - has the current, authoritative copy of a record
  - moves around as nodes write to the record
- LMASTER (location master):
  - knows the location of a record's DMASTER
  - is fixed (calculated by record hash)
  - LMASTER roles distributed across active nodes
- R/W operation to a record:
  - check if we are DMASTER
  - if not, request DMASTER role and current copy of record over network (via LMASTER)
  - read/write locally

Recovery

- what happens if a node goes down?
- data master for some records will be lost
- one node – the *recovery master* – performs *recovery*
- recovery master collects most recent copy of all records from all nodes
- additional TDB header *record sequence number* determines recentness
- at the end, the recovery master is data master for all records

Recovery Election / Recovery Lock

- recovery master is determined by an election process
- if the cluster file system supports POSIX `fcntl` byte range locks, then CTDB can use it for split brain prevention:
- election process can involve one file on shared storage: the *recovery lock* file
- nodes compete with POSIX `fcntl` byte range locks
- finally, the newly elected recovery master holds lock on the recovery lock file
- ⇒ CTDB has no split brain (other than the file system)

Performance Figures

By Andrew Tridgell and Ronnie Sahlberg, Linux Conf Australia 2009
GPFS file system

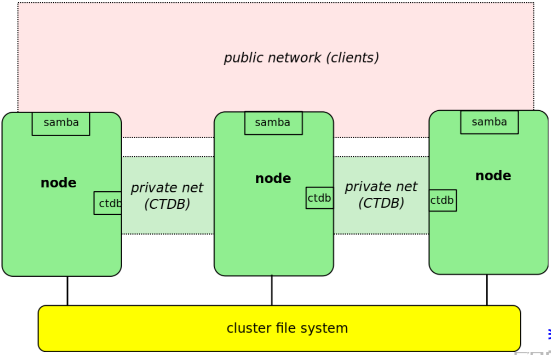
32 client smbtorture NBENCH test

- 1 node: 109 MBytes/sec
- 2 nodes: 210 MBytes/sec
- 3 nodes: 278 MBytes/sec
- 4 nodes: 308 MBytes/sec

CTDB Design – persistent TDBs

- each node always has *complete* copy in LTDB
- reads operations directly from LTDB
- write operations:
  - lock entire DB in a global lock
  - perform R/W ops in memory (prepare a marshall buffer)
  - at commit distribute changes to other nodes and write to LTDB in a local transaction
  - finally drop global lock
- ⇒ data integrity and good read performance guaranteed

CTDB - Basic Setup



CTDB - Configuration

- central file: /etc/sysconfig/ctdb
- debian based: /etc/default/ctdb
- set CTDB_RECOVERY_LOCK for split brain prevention
- fill /etc/ctdb/nodes with internal node addresses

example /etc/ctdb/nodes

```
10.11.12.10
10.11.12.11
10.11.12.12
```

- same file on all nodes!

CTDB - Public Addresses

- set CTDB_PUBLIC_ADDRESSES in /etc/sysconfig/ctdb
- typical value /etc/ctdb/public_addresses

example /etc/ctdb/public_addresses

```
172.16.17.10/24 eth2
172.16.17.11/24 eth2
172.16.17.12/24 eth2
172.16.17.13/24 eth2
172.16.17.14/24 eth2
172.16.17.15/24 eth2
```

- need *not* be the same on all nodes
- need not even be present on all nodes (management node...)

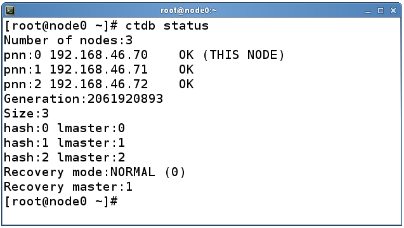
IP Failover

- healthy nodes get IP addresses from their public pool
- when a node goes down: public IPs are moved to other nodes
- CTDB distributes the public IPs equally among healthy nodes
- with round robin DNS ⇒ HA and load balancing
- speed up client reconnects with *tickle ACKs*:
  - client does not yet know the IP has moved
  - new node does not have a valid TCP connection to client
  - new node sends illegal TCP ACK packet to the client (seqnum 0)
  - client sends back correct ACK packet to the *new* node
  - new node sends back a RST packet to the client
  - client re-establishes connection to the new node

CTDB Toolbox

- `ctdb` – control `ctdbd`
- `onnode` – execute programs on selected nodes

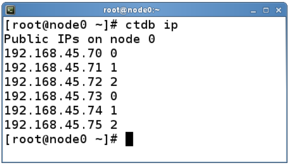
ctdb status



```
[root@node0 ~]# ctdb status
Number of nodes:3
pnn:0 192.168.46.70     OK (THIS NODE)
pnn:1 192.168.46.71     OK
pnn:2 192.168.46.72     OK
Generation:2061920893
Size:3
hash:0 lmaster:0
hash:1 lmaster:1
hash:2 lmaster:2
Recovery mode:NORMAL (0)
Recovery master:1
[root@node0 ~]#
```

SerNet
SAMBA

ctdb ip



```
[root@node0 ~]# ctdb ip
Public IPs on node 0
192.168.45.70 0
192.168.45.71 1
192.168.45.72 2
192.168.45.73 0
192.168.45.74 1
192.168.45.75 2
[root@node0 ~]#
```

SerNet
SAMBA

Let's start setting up a "real" cluster.

SerNet
SAMBA

Getting A Clustered Samba

- in vanilla Samba code since Samba 3.3 (January 2009)
- transaction rewrite in 3.5.2 (March 2010)
- precompiled packages from `http://www.enterprisesamba.org/`
- clustered Samba repository:
  `git://git.samba.org/obnox/samba-ctdb.git`
  branches: `v3-4-ctdb` and `v3-2-ctdb`
- configure `--with-cluster-support`
- add `idmap_tdb2` to `--with-shared-modules`
- verify that `gpfs.so` is built for GPFS usage

SerNet
SAMBA

Clustered File System - Requirements

- file system: black box
- storage: fibre channel, iSCSI, drbd, ...
- simulatneous writes from all nodes
- good to have: coherent POSIX `fcntl` byte range lock support use `ping_pong` test to verify

Special File Systems

- General Parallel File System GPFS (IBM): OK
- Global File System GFS(2) (Red Hat): OK
- GNU Cluster File System GlusterFS: OK
- Lustre (Sun): OK
- Oracle Cluster File System OCFS(2): OK
- Ceph: ?

Samba Configuration

identical configuration on all nodes

- `clustering = yes`
- `passdb backen = tdbsam`
- `groupdb:backend = tdb`
- `vfs objects = fileid`
  `fileid:algorithm = fsid / fsname`
- `idmap backend = tdb2`
- no need to change `private dir`

example smb.conf

```
[global]
    clustering = yes
    netbios name = smbcluster
    workgroup = mydomain
    security = ads
    passdb backend = tdbsam

    groupdb:backend = tdb

    idmap backend = tdb2
    idmap uid = 1000000-2000000
    idmap gid = 1000000-2000000

    fileid:algorithm = fsname

[share]
    path = /cluster_storage/share
    writeable = yes
    vfs objects = fileid
```

Let's configure Samba on our cluster!

CTDB manages ...

- CTDB can manage several services
- i.e. start, stop, monitor them
- controlled by sysconfig variables `CTDB_MANAGES_`*`SERVICE`*
- management performed by scripts in `/etc/ctdb/events.d`
- managed services should be removed from the runlevels
- NOTE: if `CTDB_MANAGES_SAMBA`, do *not* set `interfaces` or `bind interfaces only`

CTDB manages ...

- `CTDB_MANAGES_SAMBA`
- `CTDB_MANAGES_WINBIND`
- `CTDB_MANAGES_NFS`
- `CTDB_MANAGES_VSFTPD`
- `CTDB_MANAGES_HTTPD`

Registry Configuration

- store config in Samba's registry
- `HKLM\Software\Samba\smbconf`
- subkey ⇔ section
- value ⇔ parameter
- stored in `registry.tdb` ⇒ distributed across cluster by CTDB
- means of easily managing the whole Samba cluster

Activation of Registry Configuration

- `registry shares = yes`
- `include = registry`
- `config backend = registry`

smb.conf for cluster usage

```
[global]
    clustering = yes
    include = registry
```

net conf

manage the whole Samba cluster with one command

```
net conf list          Dump the complete configuration in smb.conf format.
net conf listshares    List the share names.
net conf import        Import configuration from file in smb.conf format.
net conf drop          Delete the complete configuration.
net conf showshare     Show the definition of a share.
net conf addshare      Create a new share.
net conf delshare      Delete a share.
net conf setparm       Store a parameter.
net conf getparm       Retrieve the value of a parameter.
net conf delparm       Delete a parameter.
net conf getincludes   Show the includes of a share definition.
net conf setincludes   Set includes for a share.
net conf delincludes   Delete includes from a share definition.
```

Let's experiment more with our cluster! ...

Thank you very much!