CTDB remix

II: Designing the Reality

Martin Schwenke <martin@meltin.net>

Samba Team
IBM (Australia Development Laboratory, Linux Technology Center)

SambaXP 2017

Overview

- Dreaming the Fantasy
- Designing the Reality
  - Cluster management
  - Service management
  - IP failover
    - Connection tracking
    - Failover daemon
  - CTDB daemon

Cluster management

Cluster management

- Cluster membership currently tightly integrated into `ctdbd`
- . . . due to transport/connectivity code
- Cluster leadership tightly integrated into CTDB `recoverd`
- New daemon with cluster leadership and (basic) membership
- Replaceable with 3rd party subsystem (e.g. etcd)?
- `ctdbd` needs to decide active nodes (e.g. `ban`, `stop`)
- New *LOST* state for known nodes that aren't in the cluster
- Need cluster-manager-specific glue in `ctdbd`

Cluster management — daemon

`ctdb_clusterd`

- `ctdb_cluster <action>`

  leave support `ctdb ban`, `ctdb stop` — shutdown?

  join all good, as you were . . .

Cluster management — daemon

`ctdb_clusterd` — notifications

Tricky integration bits...

- cluster-node-list: all configured/possible nodes
- cluster-member-list: current cluster members
- cluster-master: which node is the leader?

Service management

Service management

- Currently have `ctdb_eventd` and event scripts
- Subtract IP failover handling to leave services
- Replaceable with 3rd party subsystem (e.g. Pacemaker)?

Service management — daemon

ctdb_serviced

- `ctdb_serviced [ -e <event-script-dir> ] \`
  `[ -n <notify-script-dir> ]`

- `ctdb_service <action>`

monitor-disable	node is "unstable" (e.g. failover underway)
monitor-enable	all good, as you were . . .
reconfigure	maybe restart services? (e.g. IPs changed)
shutdown	bye!

Service management — daemon

`ctdb_serviced` — events

startup	starts services
shutdown	stops services
monitor	checks service health
reconfigure	in response to `ctdb_service reconfigure`

Service management — daemon

`ctdb_serviced` — expected event scripts

10.failover	a service, like any other. . .
20.system	existing system health checks: disk/memory/swap
49.winbind	existing winbind management
50.samba	existing `smbd`/`nmbd` management
60.nfs	existing NFS management
. . .	. . .

Service management — daemon

`ctdb_serviced` — notifications

Tricky integration bits...

service-available e.g. trigger IP failover

service-unavailable e.g. trigger IP failover

- Main `ctdbd` does not need to know about healthy/unhealthy
- `ctdb status` can still collate overall status

Service management — daemon

`ctdb_serviced` — miscellany

- When a node is inactive, `ctdb_serviced` is shut down

IP failover

IP failover

Currently CTDB supports

- Public IP addresses
- Linux Virtual Server (LVS)

and includes

- Connection tracking
- Generic routing
- Policy routing
- NAT gateway

IP failover

Observations

- LVS is currently shoehorned into public IP addresses
- Policy routing is an extension of public IP addresses
- Connection tracking is an extension of public IP addresses
- Public IP addresses are currently only supported on Linux!

IP failover

`ctdb_failoverd`

- New daemon to handle IP failover in CTDB
- IP failover "services" based on event scripts
- Node-to-node communication using "tunnel" protocol
- Replicated database for cluster-wide service state(s)
- However, `ctdb_failoverd` itself is (probably) stateless
- Connection tracking integrated or separate daemon?
- Lift LVS (and other IP failover services?) to 1st class
- Replaceable with 3rd party subsystem (e.g. Pacemaker)?

IP failover — connection tracking

Currently split between...

smbd	Hey, ctdbd! I have this new client!
ctdbd	Hey other nodes, here are some connections!
NFS	`ctdb addtickle`
Event scripts	`ctdb gettickles`, `ctdb_killtcp`

- Connection tracking can be decoupled from `smbd` and `ctdbd`
- ...without major structure!
- So, let's pick the low-hanging fruit first...

IP failover — connection tracking

Factoring out connection tracking

- `ctdb_conntrackd [ -i <commit-interval> ] \`
  `[ -c <connection-helper> ] \`
  `[ -r <reset-helper> ] \`
  `[ -s <ctdbd-socket> ]`
- `ctdb_conntrack <action>`

set-addresses	reads list of "IP-address" to monitor
reset-server	reads list of "IP-address interface" to reset
reset-client	reads list of "IP-address interface" to reset
shutdown	bye!

IP failover — connection tracking

ctdb_conntrackd -i <commit-interval>

- `ctdb_conntrackd` uses new "replicated" CTDB database
- Assume not fast enough to handle 5000 connections/second
- Specify interval between flushing connections to DB
- Even current Samba "tickle" replication is fire-and-forget!

IP failover — connection tracking

ctdb_conntrackd -c <connection-helper>

- Default Linux helper provided
- Can be replaced for testing...

`conntrack_libnetfilter_helper`

Output:

```
C 10.61.2.167:445 10.61.2.225:53452
D 10.61.2.167:445 10.61.2.225:53452
```

- BYO helper?
- Could even hook into Samba, `ss(8)` like current code

IP failover — connection tracking

ctdb_conntrackd -r <reset-helper>

- Default Linux helper provided
- Can be replaced for testing. . .

`conntrack_reset <action>`

server *interface*
- reads list of "TCP-connection" to reset
- replaces current `ctdb_killtcp`
- "needs" *interface* for packet sniffing

client
- reads list of "TCP-connection" to reset
- replaces tickle code in `ctdbd`

IP failover — connection tracking

ctdb_conntrack reset-server

`ctdb_conntrackd` does:

1. Group specified server IP addresses by interface
2. Enable internal "hold" state: do not process disconnects
3. For each interface:
   1. Get connections for IP addresses on interface
   2. `$CONNTRACK_RESET_HELPER server <interface>`
4. Disable internal "hold" state

IP failover — connection tracking

ctdb_conntrack reset-client

`ctdb_conntrackd` does:

1. Get connections for specified server IP addresses
2. Delete connections from database
3. N times (default=3):
   1. Send gratuitous ARP for each IP address
   2. `$CONNTRACK_RESET_HELPER client`

IP failover — daemon

ctdb_failoverd

- `ctdb_failoverd [ -e <event-script-dir> ] \`
  `[ -n <notify-script-dir> ] \`
  `[ -s <ctdbd-socket> ]`
- `ctdb_failover <action>`

reload	reloads configuration
failover	initiates an IP failover
shutdown	bye!

IP failover — daemon

`ctdb_failoverd` — basic events

startup	starts processes, initialises TDB(s) from configuration
shutdown	stops processes, clears node config from TDB(s)
monitor	checks processes, IP addresses are as expected
reload	reloads configuration

IP failover — daemon

ctdb_failoverd — expected event scripts

10.pubip	public IP address handling, policy routing
20.lvs	Linux Virtual Server support
30.static_routes	existing simple static route management
40.natgw	existing NAT gateway support

IP failover — daemon

`ctdb_failoverd` — failover events

Synchronised across cluster:

IP failover — daemon

`ctdb_failoverd` — failover events

Synchronised across cluster:

- calculate determine changes to be made

IP failover — daemon

`ctdb_failoverd` — failover events

Synchronised across cluster:

calculate determine changes to be made

release for public IPs: reset server end of connections, release unwanted addresses

IP failover — daemon

`ctdb_failoverd` — failover events

Synchronised across cluster:

calculate determine changes to be made

release for public IPs: reset server end of connections, release unwanted addresses

take for public IPs: take any newly required addresses, send gratuitous ARPs, tickle client end of connections

IP failover — daemon

`ctdb_failoverd` — failover events

Synchronised across cluster:

calculate determine changes to be made

release for public IPs: reset server end of connections, release unwanted addresses

take for public IPs: take any newly required addresses, send gratuitous ARPs, tickle client end of connections

finalise final tweaks, routing changes, ...

IP failover — daemon

`ctdb_failoverd` — failover events

Synchronised across cluster:

- **calculate** determine changes to be made
- **release** for public IPs: reset server end of connections, release unwanted addresses
- **take** for public IPs: take any newly required addresses, send gratuitous ARPs, tickle client end of connections
- **finalise** final tweaks, routing changes, ...

Most failover services will only need "finalise" and maybe "calculate"

IP failover — daemon

`ctdb_failoverd` — failover events

Synchronised across cluster:

- **calculate** determine changes to be made,
  (+ additional "master" step)
- **release** for public IPs: reset server end of connections, release unwanted addresses
- **take** for public IPs: take any newly required addresses, send gratuitous ARPs, tickle client end of connections
- **finalise** final tweaks, routing changes, ...

Most failover services will only need "finalise" and maybe "calculate"

IP failover — daemon

ctdb_failoverd — 10.pubip example

- startup/reload: Load IP address configuration into ctdb_failover.tdb

IP failover — daemon

ctdb_failoverd — 10.pubip example

- startup/reload: Load IP address configuration into ctdb_failover.tdb

1. calculate master: (IP-state, node-states) → IP-state'
   IP layout(s) stored in ctdb_failover.tdb

IP failover — daemon

ctdb_failoverd — 10.pubip example

- startup/reload: Load IP address configuration into ctdb_failover.tdb

1. calculate master: (IP-state, node-states) → IP-state' IP layout(s) stored in ctdb_failover.tdb
2. calculate: each node determines IPs to release/take/move, written to local file

IP failover — daemon

ctdb_failoverd — 10.pubip example

- startup/reload: Load IP address configuration into ctdb_failover.tdb

1. calculate master: (IP-state, node-states) → IP-state' IP layout(s) stored in ctdb_failover.tdb
2. calculate: each node determines IPs to release/take/move, written to local file
3. release: each node releases held but unwanted IPs ... after killing server end of connections

IP failover — daemon

ctdb_failoverd — 10.pubip example

- startup/reload: Load IP address configuration into ctdb_failover.tdb

1. calculate master: (IP-state, node-states) → IP-state'
   IP layout(s) stored in ctdb_failover.tdb
2. calculate: each node determines IPs to release/take/move, written to local file
3. release: each node releases held but unwanted IPs
   . . . after killing server end of connections
4. take: each node takes wanted but unheld IPs
   . . . moves IPs between interfaces as needed
   . . . sends gratuitous ARPs, tickles client end of connections

IP failover — daemon

ctdb_failoverd — 10.pubip example

- startup/reload: Load IP address configuration into ctdb_failover.tdb

1. calculate master: (IP-state, node-states) → IP-state'
   IP layout(s) stored in ctdb_failover.tdb
2. calculate: each node determines IPs to release/take/move, written to local file
3. release: each node releases held but unwanted IPs
   . . . after killing server end of connections
4. take: each node takes wanted but unheld IPs
   . . . moves IPs between interfaces as needed
   . . . sends gratuitous ARPs, tickles client end of connections
5. finalise: fix policy routes, ...

IP failover — daemon

`ctdb_failoverd` — notifications

Now for the tricky integration bits...

failover-begin	
ip-release-pre	NFS Ganesha grace...
ip-release-post	
ip-take-pre	
ip-take-post	
ip-layout-changed	e.g. `ctdb_service reconfigure`, replaces `ipreallocated` event
failover-end	

IP failover — daemon

`ctdb_failoverd` — miscellany

- On `shutdown` event, all IPs are released
- When a node is inactive, `ctdb_failoverd` is shut down
- Inactive nodes do not take part in failover
- `ctdb ip` replaced by helper that queries `ctdb_failover.tdb`

CTDB daemon

CTDB daemon

- What remains?
- Node transport/coordination, databases
- Eventually separate out database daemon(s)
- `ctdbd` handles startup/shutdown
- . . . and node inactive/active transitions

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM.
- IBM is a registered trademark of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Other company, product, and service names may be trademarks or service marks of others.

Questions?