Clustered NAS For Everyone
Clustering Samba With CTDB

Michael Adam <obnox@samba.org>
Samba Team / SerNet GmbH

April 15, 2009

Abstract

Clustered storage is a very popular topic. Together with the open source software CTDB, current vanilla Samba code allows for setting up a freely scaling clustered CIFS Server that shares a distributed file system–a feature that current Microsoft servers do not offer!

This paper explains the problems Samba faces when serving files from a clustered file system, describes the design principles of CTDB and demonstrates how Samba 3.3 together with CTDB as a lock-manager and failover daemon can be configured to be a robust, perfomant clustered CIFS server that scales well.

Contents

1 Introduction

As the numbers of bytes and users grow, storage space tends to become too small and too slow over time. Therefore, distributed storage comes in very handy. The storage space which is usually but not necessarily attached via a SAN, can be extended by adding disks or storage nodes, depending on the storage architecture. With a distributed file system, the same data can be accessed from an arbitrary number of nodes. By adding more cluster-nodes that use the common file system, the services offered by the cluster can in principle scale arbitrarily within the limits opposed by the hardware. This is common practice for instance with web- and database servers.

The topic of this paper is how the file system itself can be offered in a scaling manner via network protocols like CIFS and also NFS. That means the goal is to basically turning a SAN into a clustered, scaling NAS. Samba [1] together with the new open source software CTDB [3] achieves this goal for the CIFS protocol. But CTDB, while originally developed specfically as the cluster-enhancement software for Samba, has meanwhile been extended by a couple of general high availability and load balancing features that also makes other file services like NFS and FTP clusterable.

2 The File System – A Black Box

For the purposes of this paper, it is not important, exactly which distributed file system is used. Neither is it important how this file system is exactly constructed. The storage can be attached via fibre channel from a SAN, it can be imported via iSCSI, even consist of local hard disks replicated with drbd [12]. The only requirement for CTDB to be able operate is that all nodes that want to participate in the CTDB-Cluster have a common distributed file system mounted and that this file system supports POSIX-`fcntl`-locks.

2.1 Ping Pong

The simple program `ping_pong` [18] can be used to check whether a file system is suitable for use with CTDB. It tests coherence and performance of `fcntl` byte range locks on the file system as well as correctness and perfomance of concurrent write operations and the memory mapping mechanism mmap. The Samba wiki [19] has the details on ping-pong.

2.2 Special File Systems

The file system that has been most thoroughly tested with CTDB and Samba until now is probably IBM's proprietory *General Parallel File System* GPFS [13]. But there are suitable open source cluster file systems out there as well: Red Hat's *Global File System* GFS [14] has also been extensively tested. Other file systems that have been reported to work with CTDB are the GNU Cluster File System GlusterFS [15] and Sun's Lustre [16]. The Oracle Cluster File System OCFS2 [17] is *not* ready for CTDB yet, since support for POSIX-fcntl-locks is only just being worked on.

For the rest of this paper, the distributed file system will be treated as a black box.

3 Challenges For Clustered Samba

When serving files from a distributed storage, there are a number of things that need to be taken care of.

Firstly, one needs to realize that for several Samba daemons running on multiple cluster nodes, serving the same files from a common file system essentially implies that the various Samba instances act as the same CIFS server: The files must belong to the same users and for instance file locking must work across the nodes.

Operating as a single server, the Samba instances must share certain data:

- The user database has to be synchronized between the nodes, if a local user database is used at all. This is the case for a standalone server or a domain controller.
- For a member server in a Windows domain, the join information must be available to all nodes. This is basically the domain SID and the machine account password.
- The mapping tables that assoicate Unix user and group IDs to the Windows Users and Groups have to agree on all nodes.

On a lower level, certain metadata has to be available to all nodes in the cluster for proper file serving:

- The active SMB-sessions and share connections constitute the overall state of the distributed SMB server.
- A Samba server needs to offer various kinds of locks: Whole files are locked exclusively with *share modes* while *byte range locks* are used to grant exclusive access to portions of a file. It is very important for distributed Samba that these locks are honoured by the smbd processes on all the nodes. These Windows locks are mandatory, not advisory like the POSIX locks that the file system offers.

Finally, just like the Samba processes on a single server, the Samba daemons on the various nodes need to communicate with each other through a messaging mechanism. This is used for instance to notify a client that a file that had previously been locked has now become available.

Apart from the messaging that involves signals, all the data mentioned above are stored in Samba's internal TDB databases [2]. TDB is a small, fast Berkeley-DB-style database that additionally supports record locks and memory mapping. With respect to the nature of the data stored, Samba distinguishes two sorts of TDB databases: The first list of databases mentioned above is the *persistent* kind – these pieces of information need to survive restarts and usually stay around for long. The persistent data is read rather frequently but seldomly written. So write performance is not critical here. For these TDBs, data integrity is more important than performance. The second kind of databases, like the locking and session databases, contain short-lived data and need to be written and read very frequently. These databases could be called volatile or non-persistent TDBs but in CTDB speak, they are simply referred to as the *normal* TDBs. A good performance of the normal databases is critical for Samba's overall file server performance.

This is also the reason why the naive approach of storing these databases in the cluster file system does not work very well: The TDB write operations make heavy use of `fcntl` locks. And these are usually slow on cluster file systems, the more nodes, the slower. This leads to bad, even negative scaling when clustering Samba this way. But the declared goal is to achieve a scaling where the number of requests answered per second and the accumulated data throughput grow as linearly as possible.

4 The CTDB Project

This is where CTDB comes into play. CTDB can be considered as an add-on software to Samba. Apart from taking care of the inter-node-messaging of Samba, CTDB distributes the TDB databases across all cluster nodes.

4.1 History of the project

The first usable version of CTDB was released in April 2007. The preceeding work leading up to the creation of CTDB was started in 2006 by a group of developers around Volker Lendecke and Andrew Tridgell. Meanwhile, Ronnie Sahlberg has become the maintainer and principal author of the CTDB project. The official CTDB sources can be obtained from his CTDB-git-repository [7]. RPM-packages of CTDB and the latest official source tarball are available from [8].

4.2 Samba versions for use with CTDB

Until recently, only a specially patched version of Samba contained the cluster enhancements that allowed Samba to use CTDB for inter-node messaging and handling of the TDB databases. The first cluster version of Samba was based on Version 3.0.25. RPMs of this version can still be found on the web [9]. Version 3.2 of Samba that was released on July 1, 2008, contains cluster support that was still incomplete. So a branch `v3-2-ctdb` of the 3.2 series branch was created after the feature freeze of Samba 3.2. This branch is maintained in the repository [10] and contains all the cluster enhancements that were developed in the sequel. The Samba-Packages found under [8] are built from these sources. Samba 3.3 that was released in January 2009, contains full cluster support in the unmodified sources for the first time. The packages offered on `enterprisesamba.org` [11] are built with cluster support since April 2009.

5 Workings of CTDB

On each cluster node, a CTDB daemon `ctdb` is running. Samba, instead of writing direcly to its TDB databases, talks to its local `ctdbd`. The daemons negotiate the metadata for the TDBs over the network. But for the actual data write and read operations, they maintain local copies on fast local storage. According to the different requirements, CTDB treats the normal and the persistent database completely differently.

5.1 Persistent TDBs

For the persistent TDBs, each node always has a complete and up-to-date copy. So the read operations only involve fast local reads. When a node wants to write to a persisten TDB, it locks the whole database on the network with a transaction, performs its write and read operations within that transaction, and the transaction commit operation finally distributes the changes to all nodes and also writes them locally. This way data integrity and good read performance is guaranteed.

5.2 Normal TDBs

For normal TDBs, the key insight is that one node does not need to know all records of a database. Most of the time, it is sufficient when a node has up to date copies of the records that affect its own client connections. Even more importantly, when a node goes down, it is acceptable, yes even desirable to lose those data that are just about the client connections on that node.

Therefore, for a normal TDB, a node only has those records in its local TDB that it has already accessed. Data is not automatically propagated to other nodes and just transferred upon request. Only one node has the current, authoritative copy of a record, this is the record's *data master*. When a node wants to write or read a record, it first checks, whether it is the data master for this node. If so, it directly accesses the record in the local TDB. Otherwise, it first requests the current record data along with the data master role and then accesses the data locally.

Due to the fact that data is always read and written locally, a one node CTDB-cluster is essentially as fast as direct TDB access without CTDB intervention. The good scalability is further based in the fact that data is only transferred upon explicit request and not unconditionally. Performance measurements confirm this design in a very satisfactory manner: An `smbtorture-NBENCH` test with 32 clients scales as can be seen in table 1. These test results were presented by Andrew Tridgell and Ronnie Sahlberg at Linux Conf Australia 2008 [20].

nodes	**throughput**
1	109 MBytes/sec
2	210 MBytes/sec
3	278 MBytes/sec
4	308 MBytes/sec

Table 1: Performance figures

5.3 Recovery

What happens if a node dies or leaves the cluster? Since along with the node probably also the data master for some records has vanished, CTDB then performs a process called *recovery* to re-establish a proper state. The recovery is carried through by the node that holds the role of the *recovery master*. It collects the most recent copy of all records from the other nodes. In order to be able to determine what is the most recent copy, CTDB keeps track of data master role changes with a *record sequence number* (RSN) that it stores in the local TDB headers in addition to the standard TDB headers. At the end of the recovery, the record master is the data master of every record of every normal TDB.

5.4 The Recovery Lock

The recovery master is dermined by an election process. This process involves a lock file –the *recovery lock*– that is placed in the cluster file system. At the end of the election, the newly nominated recovery master holds an `fcntl`-lock on the recovery lock file. This lock is the *only* reason for the requirement that the cluster file system must support POSIX `fcntl` locks. While other election processes are imaginable that would not involve a lock on the shared file system, this mechanism has the enormous advantage that it prevents *split brain* for CTDB as long as the distributed file system stays intact!

6 Configuring CTDB

For proper operation, CTDB cluster requires at least two separate networks, one internal network, through which the CTDB daemons communicate, and one public network through which the cluster offers its services like Samba and NFS to the clients. The internal network can be the same as the network the cluster filesystem uses for communication. Figure 1 shows the basic setup of a three node CTDB cluster.

The central configuration file for CTDB is `/etc/sysconfig/ctdb`. It contains detailed comments about the available configuration parameters. The only variable that strictly *must* be set is the `CTDB_RECOVERY_LOCK`. It specifies the full path of the CTDB recovery lock file in the cluster file system. Furthermore, the administrator has to fill the file `/etc/ctdb/nodes` with the IP addresses of all the cluster nodes, that have statically been assigned to the nodes in the internal CTDB network. The location of the nodes file can be changed through the variable `CTDB_NODES` in `/etc/sysconfig/ctdb`. This file must be identical on all nodes. Here is an exampe of a nodes file:

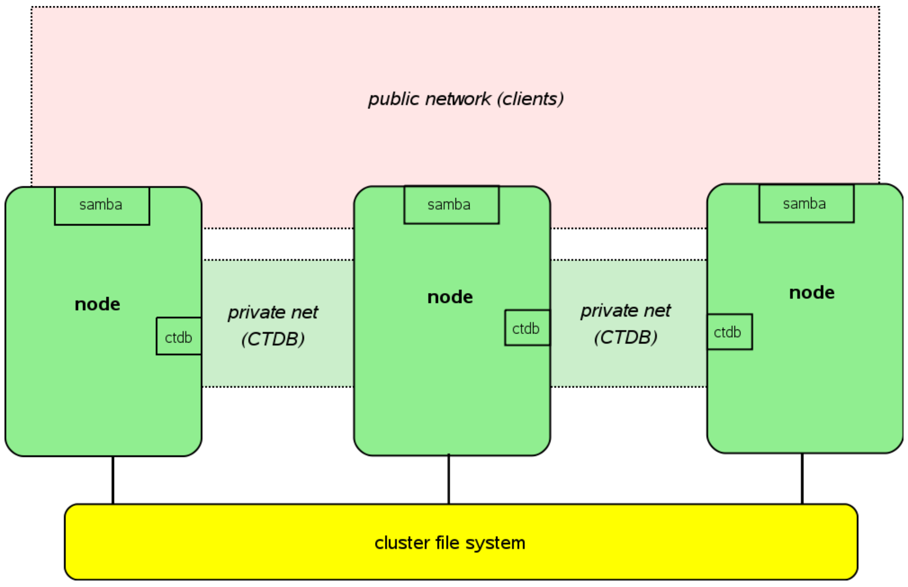


Figure 1: Basic structure of a CTDB cluster

```
10.0.0.10
10.0.0.11
10.0.0.12
```

It simply contains one IP address per line.

6.1 Public addresses

There are several modes of operation for the public network. It is possible to assign IP addresses satatically, and have CTDB stay out of business. This way, one dispenses all failover and high availability features of CTDB. A second mode is the LVS mode, in which the cluster nodes carry one common IP address and all client traffic is multiplexed by one single node, the LVS master.

In the third and most common mode, CTDB takes care of distributing a set of public IP addresses dynamically across the nodes. When a node goes down, CTDB moves the node's public IP addresses to other, healthy nodes. Together with a round robin DNS setup, this makes CTDB a loadbalancing and high availability solution for the services offered by the cluster.

To configure CTDB to handle the distribution of the public addresses, it has to be told the location of a public addresse file in the Variable `CTDB_PUBLIC_ADDRESSES` in the file `/etc/sysconfig/ctdb`. This is an example of a public addresse file:

```
192.168.0.100/24 eth0
192.168.0.101/24 eth0
192.168.0.102/24 eth0
10.11.12.13/16 eth1
10.11.12.14/16 eth1
10.11.12.15/16 eth1
```

The format is one address incuding netmask per line, with a network interface name. A default interface can be configured with the variable `CTDB_PUBLIC_INTERFACE` in the `sysconfig` file. A typical location for the public addresses file is

```
/etc/ctdb/public_addresses
```

The public addresses file does not have to agree on all nodes, it is used to specify the pool of addresses that a node can possibly hold.

6.2 IP Failover: Tickle ACKs

When a node goes leaves the cluster, CTDB moves its public IP addresse to other nodes that have the addresses listed in their public addresses pool. But now the clients connected to that node have to reconnect to the cluster. In order to reduce the nasty delays that come with these IP switches to a minimum, CTDB makes use of a clever trick called *tickle-ACK*. The dilemma is this: The client does not know that the IP he is connected to has moved, while the new CTDB node only knows the TCP connection has become invalid, but does not know the TCP sequence number. So the *new* CTDB node sends an invalid TCP packet with sequence and ACK number set to zero. This "tickles" the client to send a valid ACK packet back to the new node. Now CTDB can validly close the connection by sending a RST packet and force the client to re-establish the connection. This is especially useful for NFS clients.

6.3 CTDB manages ...

CTDB has evolved to a high availability solution for a couple of services offered by the cluster. CTDB can "manage" a service, which means that it takes care of starting and stopping the service and that CTDB monitors the runnig service. The following `sysconfig`-variables can be set to `yes` to have CTDB manage the corresponding services:

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

The management is performed by the event scripts stored under

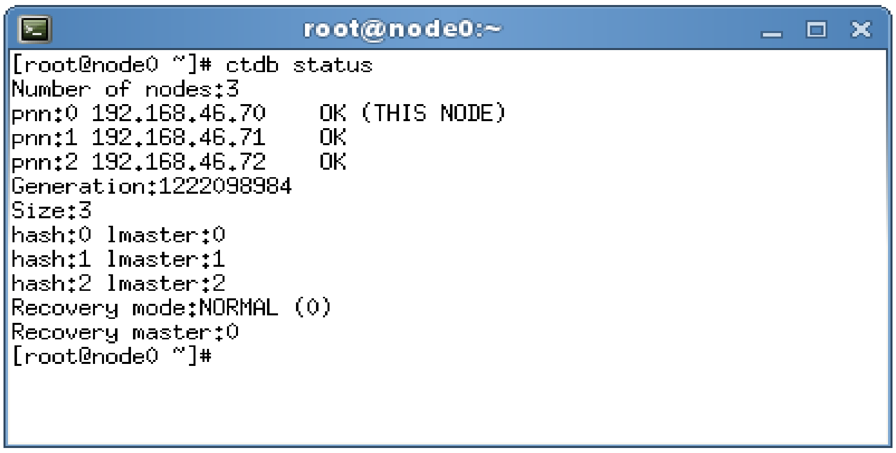
```
/etc/ctdb/events.d/
```

Note that when CTDB manages a service, this service should be removed from the system runlevels.

6.4 CTDB Toolbox

Apart from the CTDB daemon `ctdbd`, there are two commandline tools in the CTDB suite: The CTDB management tool `ctdb` and the script `onnode`.

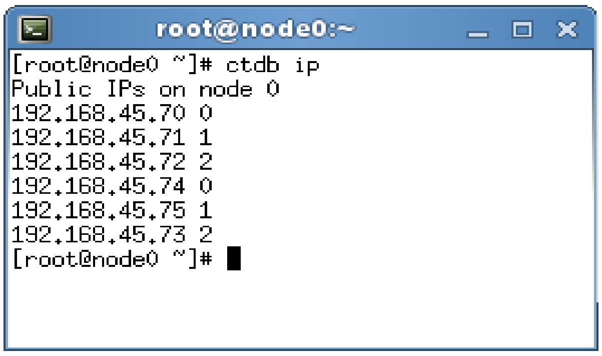
The program `ctdb` can be used to inspect and manage basically all aspects of the CTDB cluster. The most common commands are `ctdb status` (see figure 2) that



```
[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:1222098984
Size:3
hash:0 lmaster:0
hash:1 lmaster:1
hash:2 lmaster:2
Recovery mode:NORMAL (0)
Recovery master:0
[root@node0 ~]#
```

Figure 2: `ctdb status` on a three node cluster

prints an overview of the cluster's heath status and `ctdb ip` (see figure 3) which



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

Figure 3: Example `ctdb ip` output

prints the current distribution of the public IP addresse across the nodes.

The script `onnode` allows commands to be executed on all or on seleced nodes. `onnode` looks into the nodes file to get the nodes' IP addresses, but it does not require `ctdbd` to be running, since it uses `ssh` to connect to the remote nodes. This is extremely useful for distributing configuration files or software packages onto all nodes and for starting services, especially CTDB itself on all or selected nodes.

The manual pages contain all the details about `ctdb` and `onnode`.

7 Setting up Samba for Clustering

Once CTDB is set up properly, it is easy to configure Samba for clustering. Samba 3.3 that has been released on January 15, 2009, has full cluster support in the vanilla sources.

7.1 Building clustered Samba

To begin with, Samba must be compiled with cluster support. For this purpose, the configure parameter

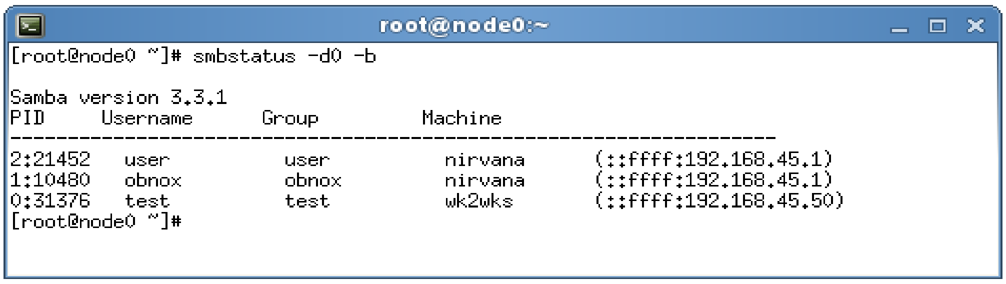
```
--with-cluster-support
```

needs to be specified. Further the list of shared modules that is passed to `configure` with `--with-shared-modules` should contain `idmap_tdb2`. The `tdb2` module is the cluster replacement for the `tdb` idmapping module. As mentioned above, precompiled Samba packages ready for cluster deployment can be downloaded from [11].

7.2 Configuration Options

Once this is done, there are a few important `smb.conf` options:

- The new option `clustering = yes` enables cluster support at runtime. This option makes samba talk to CTDB for its messaging and instead of direct TDB access for most of the TDB databases. Without this option, Samba behaves just like a Samba built without cluster support. With clustering, internal identification of the `smbd` processes is extendet to contain not only the PID but also the node number. This can for instance be seen in the output of the smbstatus command. Figure 4 shows an example.



```
[root@node0 ~]# smbstatus -d0 -b

Samba version 3.3.1
PID     Username      Group         Machine
-------------------------------------------------------------------
2:21452   user          user          nirvana      (::ffff:192.168.45.1)
1:10480   obnox         obnox         nirvana      (::ffff:192.168.45.1)
0:31376   test          test          wk2wks       (::ffff:192.168.45.50)
[root@node0 ~]#
```

Figure 4: Output of `smbstatus` in a cluster

- On some places on the Samba wiki [6] and the CTDB website [3] contain the information that the `private dir` should be put into the cluster file system. This information stems from the early days of CTDB when the support for persistent TDBs had not yet been added. Nowadays this is no longer necessary nor recommended.

- If a local password database is to be used and if it should be automatically distributed across the cluster, then the value of the parameter `passdb backend` should be changed from its default `smbpasswd` to `tdbsam`.

- The parameter `groupdb:backend = tdb` makes tdb propagate the group mappings across the cluster.
- The file identification code Samba uses internally when storing locking information is usually constructed from the device and inode number from the `stat()` system call. Since the device number is not a cluster invariant of a file, the `fileid` vfs module allows to change the algorithm used for creating the file ID. The two supported methods are `fsname` and `fsid`. The first, which is the default, constructs replaces the device number with a hash value of the file system name obtained from the `getmntent()` call, while the second uses the file system id from the `statfs()` call. The module is loaded by specifying `fileid` the list of `vfs objects` list globally or per share. The method of the file ID creation can be changed by setting `fileid:algorithm` to either `fsname` or `fsid`. See the `vfs_fileid` manpage for details.
- When using `CTDB_MANAGES_SAMBA`, it is extremely important not to change the network interfaces or addresses Samba is listening on by the parameters `interfaces` or `bind interfaces only`! This is because CTDB's monitoring of the samba services requires them to be listening on the wild card addresses `0.0.0.0` for IPv4 or `::` for IPv6.

7.3 Example

Here is an example `smb.conf` file for cluster deployment:

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

    idmap backend = tdb2

    groupdb backend = tdb
    idmap uid = 1000000-20000000
    idmap gid = 1000000-20000000

    fileid:algorithm = fsid

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

In this example, Samba acts as a member server in an Active Directory domain. This file should be distributed to all cluster nodes.

8 Registry Configuration

While this is already easy enough, the registry based configuration introduced in July 2008 with Samba 3.2, makes the management of a Samba cluster easier still. Registry configuration that can be used in addition to or as an alternative for the traditional `smb.conf` text file configuration, is stored in Samba's internal registry database in the key

```
HKLM\Software\Samba\smbconf
```

The data model of the registry is very well suited for storing Samba's configuration data which consists of sections built of parameters: A registry key consists of its name, a list of its subkeys, and a list of its values. A registry value consits of the value name and the value data. So configuration sections correspond to subkeys of the `smbconf` key and parameters in a section correspond to the values in the respective registry subkey.

The key feature of the registry configuration is the fact that Samba stores the registry data in the `registry.tdb`, which is hence automatically distributed across the cluster nodes.

8.1 Three Levels of Activation

Registry configuration can be activated in three levels:

- The first level is to only load share definitions from registry. This is triggered by setting

  ```
  registry shares = yes
  ```

  in the global section of `smb.conf`.

- Global options can be included from registry by specifying

  ```
  include = registry
  ```

  in the global section of `smb.conf`. In this case global options from registy are mixed with the global options from the text file just like when including a text file. Registry shares are implicitly activated by this mode.

- A registry-only configuration can be triggered by putting

  ```
  config backend = registry
  ```

  into the global section of `smb.conf`. This ignores all text configuration and reads global and share configuration exclusively from registry.

Note that the text configuration file `smb.conf` is still the initial configuration source in all cases. For the clustering scenario, a registry only configuration is not possible, since Samba has to be told for a start that registry is to be accessed not locally but through CTDB. Hence the minimal `smb.conf` file that has to be present on all cluster nodes looks like this:

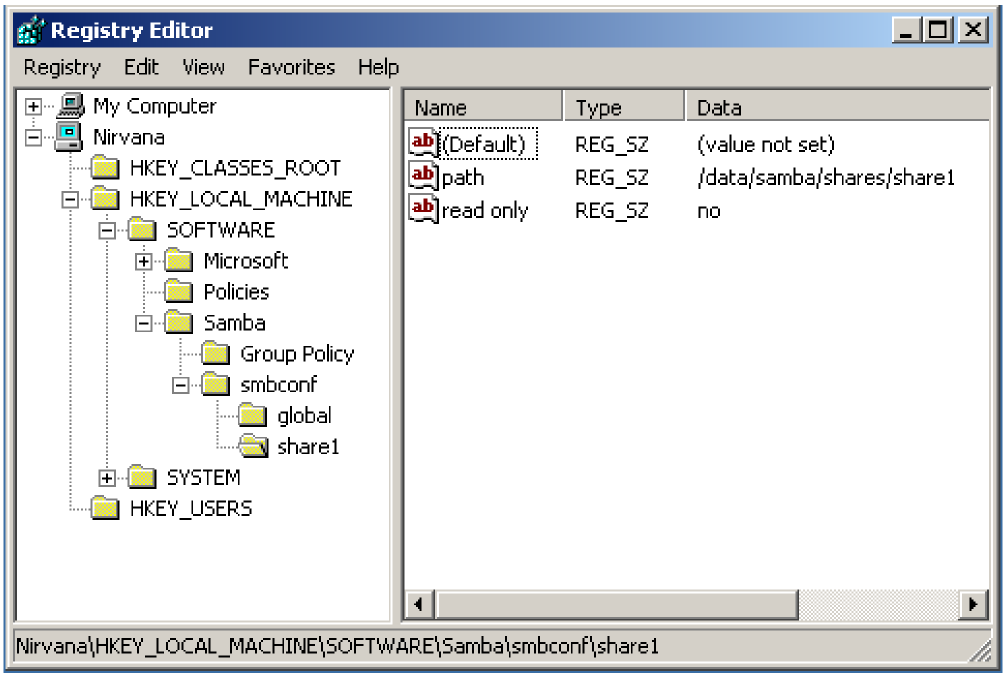
```
[global]
    clustering = yes
    include = registry
```

All other configuration options can be put into registry and therefore after this initial text configuration, the whole cluster can be configured in one workstep.

8.2 Accessing Registry Configuration

Now that registry configuration has been activated, how does one fill the registry with configuration data? There are several possible ways:

- The windows administrator can use the remote feature of the windows registry editor `regedit.exe` (see figure 5).



Name	Type	Data
(Default)	REG_SZ	(value not set)
path	REG_SZ	/data/samba/shares/share1
read only	REG_SZ	no

Nirvana\HKEY_LOCAL_MACHINE\SOFTWARE\Samba\smbconf\share1

Figure 5: Configuring Samba with `regedit.exe`

- From the unix command line, the two commands `net registry` and `net rpc registry` offer basically the same functionality as `regedit.exe` locally and over the `WINREG` rpc interface, respectively.

- But the most comfortable way is to use the new dedicated tool `net conf` that offers an interface specific to registry configuration. Table 2 shows the list of subcommands offered by net conf. For the details on net conf, see the `net` manual page and the commandline help for `net conf`.

The details on registry configuration system can be obtained from [21].

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

Table 2: The `net conf` subcommands.

9 Conclusion

Given a clustered file system that supports POSIX `fcntl` byte range locks, vanilla Samba 3.3 and CTDB allow for easily setting up a clustered CIFS server that scales very well, thanks to the design of CTDB. Hence, with an open source cluster file system like GFS or GlusterFS, one can implement a clustered NAS using only open source software: Linux, CTDB, and Samba. Samba's registry configuration system greatly eases the adminstration of the Samba/CTDB cluster.

References

[1] Samba - the Open Source CIFS server for UNIX. http://www.samba.org/

[2] TDB - Samba's trivial database. http://tdb.samba.org/

[3] CTDB - the clustered tdb project. http://ctdb.samba.org/

[4] Samba Wiki: *CTDB project*: http://wiki.samba.org/index.php/CTDB_Project

[5] Samba Wiki: *Samba & Clustering*: http://wiki.samba.org/index.php/Samba_%26_Clustering

[6] Samba Wiki: *CTDB Setup*: http://wiki.samba.org/index.php/CTDB_Setup

[7] CTDB, the official git repository, git://git.samba.org/sahlberg/ctdb.git

[8] CTDB, packages and sources, http://ctdb.samba.org/packages/

[9] Old RPM packages of CTDB and Samba 3.0.25-ctdb, http://ctdb.samba.org/packages/ibm/SOFS

[10] The Clustered Samba git Repository, git://git.samba.org/obnox/samba-ctdb.git

[11] Samba Packages for Enterprise Linux, a services offered by SerNet GmbH, http://www.enterprisesamba.org/

[12] DRBD, the Distributed Replicated Block Device, http://www.drbd.org/

[13] IBM, General Parallel File System (GPFS), http://www-03.ibm.com/systems/clusters/software/gpfs/index.html

[14] Red Hat, Global File System (GFS), http://www.redhat.com/gfs/

[15] GNU Cluster File System (GlusterFS), http://www.gluster.org/

[16] Lustre File System, http://www.lustre.org/

[17] Oracle Cluster File System (OCFS2), http://oss.oracle.com/projects/ocfs2/

[18] ping_pong.c, http://junkcode.samba.org/ftp/unpacked/junkcode/ping_pong.c

[19] Sama Wiki: *Ping Pong*, http://wiki.samba.org/index.php/Ping_pong

[20] Andrew Tridgell und Ronnie Sahlberg, *Clustered Samba*, Talk at Linux Conf Australia 2008, http://mirror.linux.org.au/pub/linux.conf.au/2008/slides/178-tridge-ctdb.pdf

[21] Michael Adam, *Samba's New Registry Based Configuration System*, Uptimes No. 2/2008, pp. 105–124, ISBN 978-3-86541-300-0 (pdf: http://samba.org/~obnox/presentations/linux-kongress-2008/lk2008-obnox.pdf)

License

This paper is a slightly extended version of a paper by the author, submitted to the NLUUG spring conference 2009, "Filesystems and Storage", http://www.nluug.nl/activiteiten/events/vj09/index-en.html.