Systemmanagement

Storage und verteilte Dateisysteme

Eingereicht von: **Ingo Ebel**
Computer Science and Media
4. Semester
ingo.ebel@ingoebel.de
Matrikel-Nr.

Veranstaltung von: **Prof. Walter Kriha**

Kurzfassung

Die vorliegende Arbeit beschäftigt sich mit einem Mailserver-Setup, das auf einen gemeinsamen Storage zugreifen soll. Dazu werden die Clusterdateisysteme Ceph und OCFS2 getestet und ihr Einsatz auf Debian GNU/Linux beschrieben.

Schlagwörter: Dateisystem, Clusterfilesystem, verteiltes Dateisystem, Ceph, DRBD, iSCSI, OCFS2, Linux, Debian

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Um von mehreren Servern auf einen gemeinsam Storage zuzugreifen, wird ein verteiltes Dateisystem (Clusterfilesystem) benötigt. Packt man noch Virtualisierung dazu, hat man das, was heute unter dem Thema „Cloud“ in aller Munde ist. Dieser Storage ist also die solide Grundlage für eine Cloud-Infrastruktur. Aus diesem Grund wollte ich mir dieses Thema in „System Management“ genauer ansehen, außerdem sollte ein Setup für ein Mailsystem entstehen. Hierzu verschaffte ich mir einen Überblick über SAN[1]- und NAS-Systeme[2] und den dahinterstehenden Technologien. Da sich nicht jeder ein SAN leisten kann und Ethernet praktisch überall verfügbar ist hat auch iSCSI, also SCSI über Ethernet, immer größere Bedeutung bekommen. Daher wollte ich mir auch iSCSI näher anschauen. Anschließend wurden zwei Clusterdateisysteme ausprobiert und der Prototyp für das neue Mailserversetup geschaffen.

1.1 Interessante Erkenntnisse und Tools

Während dieser Arbeit habe ich zwei Kleinigkeiten entdeckt, die ich hier explizit erwähnen will. Sie machen das Arbeiten mit einer größeren Anzahl von Servern oder virtuellen Maschinen viel einfacher. Zum einen ist das ein Tool bzw. eine Kategorie von Tools, nämlich Clustershells. Diese loggen sich per SSH auf vielen Servern ein und man kann über eine Shell mehrere Server gleichzeitig steuern. Gerade wenn auf vielen Servern jeweils die gleichen Arbeiten (Updates, Einspielen von Software, Einrichten der Config Datei) zu machen sind, ist dies eine enorme Erleichterung. Diese Tools gibt es u. a. für Linux (clusterssh[3], pssh[4]) oder Mac OS X (csshX[5]). Das zweite ist eine kleine Einstellung für ssh: der Forward Agent, der es erlaubt SSH-Keys quasi über Server Grenzen hinweg zu benutzen. Loggt man sich auf Server A ein, und von dort auf Server B, kann dies auch mit dem SSH-Key geschehen, der nur auf dem eigenen Rechner liegt. Der Key wird also

1 Storage Area Network
2 Network Attached Storage
3 siehe http://sourceforge.net/projects/clusterssh/
4 http://www.theether.org/pssh/
5 http://code.google.com/p/csshx/

weitergegeben; ähnlich einem Token. Eine schöne bebilderte Beschreibung findet sich unter http://unixwiz.net/techtips/ssh-agent-forwarding.html. Um den Forward Agent zu benutzen, kann man folge Zeile in seine .ssh/config aufnehmen:

```
Host *
        ForwardAgent      yes
```

2 Storage und Dateisysteme

Die Liste der verteilten Dateisysteme ist lang. Die deutsche Wikipedia listet alleine 29 Netzwerk- und Cluster-Dateisysteme[6]. Davon sind folgende meiner Meinung nach am interessantesten:

- Ceph, objektbasiertes paralleles Cluster-Dateisystem für Linux
- GFS, Global File System von Red Hat
- Lustre, objektbasiertes Cluster-Dateisystem für Linux
- OCFS2, Oracle Cluster File System Version 2
- XtreemFS, objektbasiertes verteiltes Dateisystem für Linux, Mac OS X und Windows

Alle diese Dateisysteme sind POSIX-konform und können unter Linux benutzt werden. Einige stehen auch für Mac OS X zur Verfügung. Ich verwende auf den Servern Debian GNU/Linux; die Dateisysteme müssen also auf Linux-Servern laufen. Debian wird in der Version Squeeze (Debian 6.0) eingesetzt. Die Dateisysteme lassen sich unter anderem grob in zwei Kategorie einteilen: ob sie selbst fehler-tolerant sind, d. h. Redundanz besitzen, oder nicht. Ceph, Lustre und XtreemFS gehören zu der Gruppe der fehlertoleranten Systeme, während GFS und OCFS2 dies nicht bieten[13]. GFS und OCFS gehören eher zur alten Garde und sind vom Featureumfang sehr identisch. Die neueren Dateisysteme sind alle objektbasiert. Im Folgenden habe ich mir zwei der Dateisysteme genauer angesehen, den Anfang dabei macht Ceph.

2.1 Ceph

Durch seine Objektbasierung und Fehlertolerantheit ähnelt es dem Google File System. Ceph wurde von Sage Weil für seine Doktorarbeit erschaffen [14] und besteht aus drei Komponenten: dem Cluster Monitor (MON), dem Metadaten Server (MDS) und den

6 siehe http://de.wikipedia.org/wiki/Liste_von_Dateisystemen

Abbildung 2.1: Logo von Ceph

Object Storage Nodes (OSDs). Alle diese Komponenten können verteilt werden; es kann mehrere von ihnen geben, die wiederum über verschiedene Server verteilt sind. Manche Komponenten können aber auch auf einem Server gemeinsam laufen. Die OSDs speichern ihre Daten wiederum auf lokalen Dateisystemen, als Standard wird btrfs[7] genommen, jedoch sind auch andere Dateisysteme möglich. Zusätzlich bietet ceph ein zu Amazon S3-kompatibles REST-Interface an. So können Applikationen, die für S3 geschrieben wurden, in der „Privaten-Cloud“ auch Ceph nutzen.

Die erste Anlaufstelle für Installationen und Informationen rund um Ceph ist das Wiki unter http://ceph.newdream.net/wiki/. Dort gibt es auch die Anleitung für eine Installation unter Debian. Für Debian und Ubuntu werden auch extra Repositorys gepflegt, in denen die benötigten Pakete zum Aufsetzten des Clusters vorhanden sind. Der Linux Kernel enthält seit 2.6.34 bereits den Ceph Client und so ist ein Patchen des Linux Kernels nicht nötig. In jeder Kernel Version werden natürlich Änderungen und Verbesserungen am Ceph Kernel Module hinzugefügt, daher benutzte ich Debian Squeeze nicht mit dem mitgebrachten Kernel, sondern habe diesen erst mal aktualisiert. Die entsprechenden Änderungen an der /etc/apt/source.list vorrausgesetzt, kann mit

```
aptitude install -t experimental linux-image-2.6.37-trunk-amd64
```

der Kernel 2.6.37 installiert werden. Nach einem Neustart der Virtuellen Maschine (VM) mussten die Ceph Tools installieren werden:

```
echo deb http://ceph.newdream.net/debian/ squeeze main >> /etc/apt/
    source.list
wget http://newdream.net/~sage/pubkey.asc -q -O- | apt-key add -
aptitude update
aptitude install bzip2
aptitude install ceph
```

Dann musste das Kernelmodule mit modprobe ceph geladen werden. Mein Testsetup besteht aus drei Virtuellen Maschinen (node0 alias cloud1, node1 alias cloud2 und node2

7 Das B-tree file system ist das Next Generation Dateisystem für Linux und soll ext4 ablösen. Es hat einen ähnlichen Featureumfang wie das von SUN entwickelte ZFS.

alias cloud3), das heißt alle eben genannten Schritte müssen auf allen Server ausgeführt werden.

Damit das Clustertool auf alle Server zugreifen kann, sollten die Nodes in die /etc/hosts eingetragen werden; außerdem muss ein SSH-Key ohne Passphrase erzeugt werden und auf allen Nodes akzeptiert sein.

```
root@cloud1:~# cat /etc/hosts
127.0.0.1       localhost
46.4.156.77     cloud1.jit-creatives.de cloud1 node0
46.4.156.78     cloud2.jit-creatives.de cloud2 node1
46.4.156.79     cloud3.jit-creatives.de cloud3 node2

ssh-keygen
ssh-copy-id -i .ssh/id_rsa.pub root@node0
ssh-copy-id -i .ssh/id_rsa.pub root@node1
ssh-copy-id -i .ssh/id_rsa.pub root@node2
```

Da dies nur ein Testsetup ist, wurde hier kein großer Sicherheitsmaßstab angelegt. In Produktivumgebungen sollte der Zugriff per Key eingeschränkt, und nur bestimmte Shell Befehle zugelassen werden. Anschließen kann es an das Design des Clusters gehen. Je nachdem wieviel Hardware zur Verfügung steht, ergeben sich unterschiedliche Möglichkeiten die OSD, MON und MDS aufzuteilen. Um Redundanz zu erhalten, sollten es mindestens 2 MDS sein und 1 oder 3 MON, dazu soviele Datenspeicher (OSD) wie möglich[11]. Der Einfachheit halber wurden hier 3 MONs, 3 MDS und 3 ODS verwendet. Das bedeutet auf jeder VM läuft genau ein MON, ein MDS und ein OSD. Diese Informationen werden in die /etc/ceph/ceph.conf geschrieben. Diese muss auf allen Cluster-Nodes vorhanden und identisch sein.

Listing 2.1: ceph.conf

```
[global]
        pid file = /var/run/ceph/$name.pid
        auth supported = cephx
        debug ms = 1
[mon]
        mon data = /data/mon$id
[mon0]
        host = node0
        mon addr = 46.4.156.77:6789
```

```
[mon1]
        host = node1
        mon addr = 46.4.156.78:6789
[mon2]
    host = node2
        mon addr = 46.4.156.79:6789
[mds]
        keyring = /data/keyring.$name
[mds0]
        host = node0
[mds1]
        host = node1
[mds2]
          host = node2
[osd]
   sudo = true
   osd data = /data/osd$id
   btrfs devs = /dev/vdb1
   osd journal = /cephjournal/journal
   osd journal size = 512
   osd keyring = /data/keyring.$name
   keyring = /data/keyring.$name
[osd0]
        host = node0
[osd1]
        host = node1
[osd2]
        host = node2
```

Damit sind die Vorbereitungen abgeschlossen. Jetzt wird der Cluster initialisiert.

```
mkcephfs -c /etc/ceph/ceph.conf --allhosts --mkbtrfs -k /etc/ceph/
    keyring.bin
```

Damit wird auf allen Cluster-Nodes der Datenspeicher mit btrsfs formatiert. Danach startet man mit /etc/init.d/ceph -a start auf allen Nodes die jeweiligen installieren Teile des Clusters. Im Falle des Testsetups also mon0-3, ods0-3 und mds0-3 siehe Abbildung 2.2.

```
root@cloud1:~# /etc/init.d/ceph -a start
=== mon.0 ===
Starting Ceph mon0 on node0...already running
=== mon.1 ===
Starting Ceph mon1 on node1...
 ** WARNING: Ceph is still under heavy development, and is only suitable for **
 **          testing and review.  Do not trust it with important data.       **
starting mon.1 rank 1 at 46.4.156.78:6789/0 mon_data /data/mon1 fsid 0780adee-016a-313e-07b3-d59b56945a81
=== mon.2 ===
Starting Ceph mon2 on node2...
 ** WARNING: Ceph is still under heavy development, and is only suitable for **
 **          testing and review.  Do not trust it with important data.       **
starting mon.2 rank 2 at 46.4.156.79:6789/0 mon_data /data/mon2 fsid 0780adee-016a-313e-07b3-d59b56945a81
=== mds.0 ===
Starting Ceph mds0 on node0...
 ** WARNING: Ceph is still under heavy development, and is only suitable for **
 **          testing and review.  Do not trust it with important data.       **
2011-04-13 15:06:06.345628 7f9a98be1720 starting mds.0 at 0.0.0.0:6800/1085
=== mds.1 ===
Starting Ceph mds1 on node1...
 ** WARNING: Ceph is still under heavy development, and is only suitable for **
 **          testing and review.  Do not trust it with important data.       **
2011-04-13 15:06:10.852942 7f1677908720 starting mds.1 at 0.0.0.0:6800/1580
=== mds.2 ===
Starting Ceph mds2 on node2...
 ** WARNING: Ceph is still under heavy development, and is only suitable for **
 **          testing and review.  Do not trust it with important data.       **
2011-04-13 15:06:15.092520 7f079fa91720 starting mds.2 at 0.0.0.0:6800/1513
=== osd.0 ===
Mounting Btrfs on node0:/data/osd0
Scanning for Btrfs filesystems
failed to read /dev/sr0
Starting Ceph osd0 on node0...
 ** WARNING: Ceph is still under heavy development, and is only suitable for **
 **          testing and review.  Do not trust it with important data.       **
2011-04-13 15:06:37.412101 7f2322d01720 starting osd0 at 0.0.0.0:6801/1257 osd_data /data/osd0 /cephjournal/journal
=== osd.1 ===
Mounting Btrfs on node1:/data/osd1
Scanning for Btrfs filesystems
failed to read /dev/sr0
Starting Ceph osd1 on node1...
 ** WARNING: Ceph is still under heavy development, and is only suitable for **
 **          testing and review.  Do not trust it with important data.       **
2011-04-13 15:06:46.462629 7f84dd562720 starting osd1 at 0.0.0.0:6801/1675 osd_data /data/osd1 /cephjournal/journal
=== osd.2 ===
Mounting Btrfs on node2:/data/osd2
Scanning for Btrfs filesystems
failed to read /dev/sr0
Starting Ceph osd2 on node2...
 ** WARNING: Ceph is still under heavy development, and is only suitable for **
 **          testing and review.  Do not trust it with important data.       **
2011-04-13 15:06:55.325613 7f5383c06720 starting osd2 at 0.0.0.0:6801/1607 osd_data /data/osd2 /cephjournal/journal
```

Abbildung 2.2: Ausgabe beim Starten aller Ceph-Bestandteile ausgehend von cloud1

Das Setup benutzt schon einen Schlüssel, damit nicht jeder das Clusterdateisystem mounten kann (auth supported = cephx). Damit andere Clients das Clusterdateisystem mounten können, müssen sie den Schlüssel besitzen, welcher mit

```
cauthtool --print-key /etc/ceph/keyring.bin > /etc/ceph/secret
```

exportiert wird.

Nun kann das Cluster-Dateisystem gemountet werden:

```
mount -t ceph -o name=admin,secretfile=/etc/ceph/secret node0:/ /mnt/
```

Dieser Mount-Befehl könnte jetzt von jedem Linux mit Kernel >=2.6.34 ausgeführt werden, vorausgesetzt die Datei mit dem Secret ist vorhanden und node0 ist in der hosts-Datei eingetragen. Statt node0 kann natürlich auch eine IP-Adresse verwendet werden.

2.1.1 Tests

Leider ließ sich das aufgesetzte System nicht richtig testen. Schon sequenzielle Schreibversuche mittels dd führten nach kurzer Zeit zu einem kompletten Stillstand der Virtuellen Maschine mit Kernelpanics. Dieses Problem ließ sich nicht in den Griff bekommen; trotz

einiger Versuche die Configs noch mal anzupassen. Das Schreiben kleiner Textdateien lief, aber per dd brachte ich das System zum Abstürzten. Das kann natürlich mit dem eingesetzten Setup zusammen hängen, da nur Virtuelle Maschinen auf einem physischen Server zur Verfügung standen; dennoch darf das System nicht komplett abstürzen, wenn es mal größere Dateien schreiben muss. Auch ein weiterer Test, bei dem eine Virtuelle Maschine auf einen anderen physischen Server gelegt wurde, bracht keine Verbesserung.

Damit war klar: Ceph ist noch nicht reif für den dauerhaften Einsatz, auch wenn es sehr interessante Features hat. Daraufhin schaute ich mir ein Clusterdateisystem an, welches schon länger im Markt ist, und fand mit OCFS den richtigen Kandidaten.

2.2 iSCSI, DRBD und OCFS

Die klassischen Verteilen Dateisysteme arbeiten doch etwas anders als Ceph. Es wird ein großer Storage (Massenspeicher) freigegeben, auf welchen mehrere (Virtuelle-)Maschinen zugreifen. Bei Ceph dagegen kann jeder Node Daten bzw. einen Teil seiner Festplatte beisteuern und auch direkt auf den Pool zugreifen.

Klassischerweise werden oft teure SAN-Systeme eingesetzt, wie etwa von EMC, Cisco, HP oder NetApp. Diese sind oft über Fibre Channel oder neuerdings aber auch über iSCSI angebunden. Trotzdem bilden sie einen Single-Point-of-Failure wenn man nicht gleich mehrere SAN Systeme kauft. Außerdem sind sie Hersteller-proprietär, also wenn man von einem SAN Hersteller Hardware gekauft hat, muss man immer bei diesem kaufen, oder alles komplett neu von einem anderen Hersteller beziehen. Billiger kann es daher sein, sein eigenes SAN zu bauen, damit es von anderen VMs angesprochen werden kann. Dabei wird iSCSI genommen. Für die Redundanz sorgt DRBD[8] und für den gleichzeitigen Zugriff von verschiedenen VMs sorgt ein klassisches, verteiltes Dateisystem. Ich habe hier OCFS2 genommen. OCFS kommt von Oracle und ist schon seit mehreren Jahren im Einsatz. Die Wahl fiel nach einer kleinen Umfrage von mir auf Twitter und Co auf OCFS. Der andere Kandidat war GFS2 von Red Hat. Beide Systeme haben Vor- und Nachteile. So ist GFS2 etwas älter und unterstützt außerdem ACLs und Quotas[9]. OCFS lässt sich jedoch leichter aufsetzten.

2.2.1 Setup 1

Die erste Idee war ein Hochverfügbarkeitssetup mit iSCSI, DRBD und OCFS, welches Abbildung 2.3 zeigt. Um ein solches Setup zu realisieren, müsste man jedoch eine Clustersoftware einsetzten, die auch einen IP-Failover unterstützt [7]. Das war mir erst mal

8 Das Distributed Replicated Block Device stellt quasi ein RAID 1 über Netzwerk (IP) sicher.

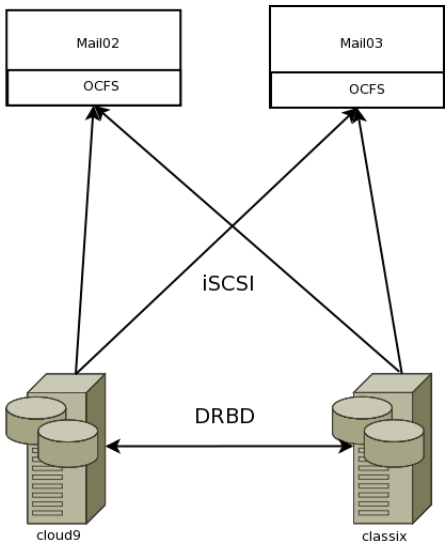

Abbildung 2.3: gedachtes Mailserver Setup

zu groß und ich wollte mich zunächst mit den Basics beschäftigen und nicht gleich eine Clusterverwaltungssoftware aufsetzten, welche dann auch viele Einrichtungsschritte automatisch und für den Nutzer nicht sichtbar vornimmt. Daher wurde die Setup-Idee umgewandelt (siehe Abbildung 2.4) und enthält kein Überkreuz-iSCSI-Mounting mehr. Das wäre zwar theoretisch per Hand möglich, spielt aber beim Setup keine Rolle. Zunächst wurden alle nötigen Pakete auf den Servern (cloud9 + classix) installiert.

```
aptitude install drbd8-utils ocfs2-tools iscsitarget
    iscsitarget-dkms
```

Schwierigkeiten gab es bei dem iSCSI Module unter Debian Squeeze. Wie schon in Abschnitt 2.1 erwähnt, nutzte ich Debian Squeeze nicht mit den Standardkernel 2.6.32, der schon etwas angestaubt ist, sondern den 2.6.37 aus Debian Unstable. Auch weil DRBD erst seit 2.6.33 offiziell im Kernel enthalten ist [12]. Leider musste dafür das iSCSI Modul neu kompiliert werden. Zwar bringt Debian Squeeze einen Automatismus (iscsitarget-dkms) dafür mit, aber dieser schlug zusammen mit dem neueren Kernel fehl. Erst die svn Version vom iSCSI Target[9] brachte Abhilfe und ließ das Kernel Modul dann richtig kompilieren. Auch bei DRBD hat sich im Kernel einiges getan, so dass beim Starten von DRBD die Meldung kam, dass Kernel und Userland nicht zusammen passen. Die Meldung könnte man ignorieren, da keine Major Version geändert wurde, aber um das bestmögliche Ergebnis zu haben, sollte auch das Userland richtig passen.

9 siehe http://sourceforge.net/projects/iscsitarget/develop

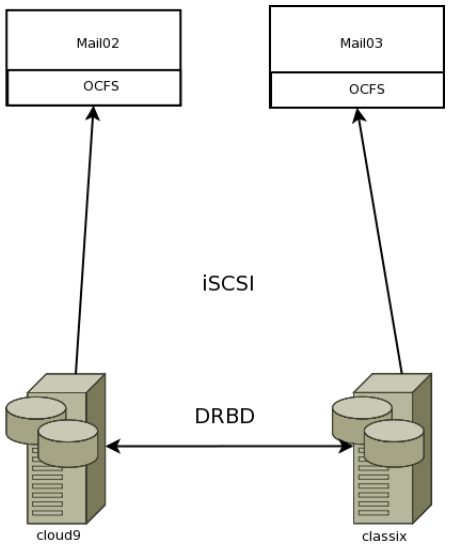

Abbildung 2.4: Test-Setup 1

Daher musste ein neues drbd-utils Debianpaket erzeugt werden. Eine Anleitung findet sich unter [3].

DRBD Konfiguration

Da DRBD die Basis bildet, muss dieses zuerst eingerichtet werden. Die Konfiguration erfolgt über die Datei /etc/drbd/drbd.conf bzw. deren includes unterhalb von /etc/drbd.d/. Die Einrichtung erfolgt nach der Anleitung im DRBD Users Guide [4]. Die Resource heißt hier r0 und wird in /etc/drbd.d/r0.res konfiguriert. Da ein Clusterdateisystem aufgesetzt wird, soll DRBD im Master-Masterbetrieb laufen. In der DRBD Sprache heißt das Primary-Primary. Listing 2.2 zeigt die fertige Resourcen-Datei; diese muss auf beiden DRBD Nodes (cloud9, classix) gleich sein. Dann kann DRBD gestartet werden und mit „drbdadm – –overwrite-data-of-peer primary r0“ auf einem Node die initiale Synchronisation gestartet werden. Auf welchem Node ist egal, da noch keine Daten vorhanden sind.

Listing 2.2: /etc/drbd.d/r0.res

```
resource r0 {
        startup {
                become-primary-on both;
        }
        handlers {
```

```
                split-brain "/usr/lib/drbd/notify-split-brain.
                    sh root";
        }
        net {
                allow-two-primaries;
                after-sb-0pri discard-zero-changes;
                after-sb-1pri consensus;
                            #discard-secondary;
                after-sb-2pri disconnect;
        }
        on cloud9 {
                device     /dev/drbd0;
                disk       /dev/sdb1;
                address    78.46.70.7:7789;
                meta-disk internal;          }
        on drbd01 {
                device /dev/drbd0;
                disk     /dev/mapper/drbd01-drbd;
                address 78.47.9.21:7789;
                meta-disk internal;
        }
}
```

Die Synchronisation dauert einige Zeit und ist abhängig von der Netzwerkgeschwindigkeit und der Größe des DRBD Volumes. In diesem Beispiel sind es ca. 400 GB. Nachdem die Synchronisation abgeschlossen ist (siehe Listing 2.3) kann auf dem zweiten Node, der bisher nur secondary ist, „drbdadm primary r0“ ausgeführt werden. Damit ist der Master-Master-Betrieb fertig konfiguriert.

Listing 2.3: Ausgabe von /proc/drbd - Synchronisation ist beendet

```
cloud9:/etc/drbd.d# cat /proc/drbd
version: 8.3.10 (api:88/proto:86-96)
srcversion: 24353649F58F784AA5458AB
0: cs:Connected ro:Primary/Secondary ds:UpToDate/UpToDate C r-----
ns:0 nr:0 dw:0 dr:208 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:0
```

iSCSI Konfiguration

Die nötige Software ist auf den Servern bereits installiert (das iSCSI Enterprise Target, kurz IET). In der iSCSI Sprache heißen diese Server Targets. Die Systeme, die die Targets einbinden, also in diesem Beispiel hier mail02 und mail03, heißen Initiators. Es gibt mehrere verschiedene iSCSI Implementierungen für Linux: sowohl als Target als auch als Initiator. Ab Kernel 2.3.38 ist ein neues Module namens LIO dabei. Da in diesem Beispiel noch Kernel 2.6.37 verwendet wird, ist LIO keine Option.

Auf den Targets wird die Datei /etc/iet/ietd.conf angepasst:

```
Target iqn.2011-09.de.jit-creatives:iscsi.drbd1
        Lun 0 Path=/dev/drbd0,Type=fileio
```

In dieser Datei können viele Parameter gesetzt werden; unter anderem können auch User und Passwörter vergeben werden, mit denen sich ein Initiator identifizieren muss. Für das Testsetup wurde darauf verzichtet und nur die einfachste Konfiguration gewählt. Anschließend muss in der /etc/iet/targets.allow festgelegt werden, welche IP Adressen auf welche Targets zugreifen dürfen. Für den Testzeitraum wurde ALL ALL angeben. Das heißt dass alle IP-Adressen auf alle Targets zugreifen dürfen die auf dem Server existieren. Das sollte auf keinen Fall als dauerhafte Einstellung bestehen! Die IPs oder IP-Netze können hier auch genau definiert werden z. B. 78.46.0.0/16. Mit /etc/init.d/iscsitarget restart werden die neuen Konfigurationen eingelesen.

Nun werden die Initiators konfiguriert. Dazu wird open-scsi auf mail02 und mail03 installiert. Um sich mit einem iSCSI Target zu verbinden, muss erst mal die Liste aller Targets geholt werden[10]:

```
root@mail02:~# iscsiadm -m discovery -t sendtargets -p
    78.46.70.7:3260
78.47.9.17:3260,1 iqn.2011-09.de.jit-creatives:iscsi.drbd1
78.46.70.7:3260,1 iqn.2011-09.de.jit-creatives:iscsi.drbd1
78.47.12.193:3260,1 iqn.2011-09.de.jit-creatives:iscsi.drbd1
```

Der Server meldet die Targets zurück. Da der hier gefragte Server über mehrere Networkinterfaces mit dem Internet verbunden ist, meldet er drei Einträge zurück, die alle das gleich Target beschreiben, was sich auch an der ID erkennen lässt. Nun kann kann sich der Initiator mit dem ausgewählten Target verbinden.

```
root@mail02:~# iscsiadm -m node -T iqn.2011-09.de.jit-creatives
    :iscsi.drbd1 -p 78.47.70.7:3260 --login
```

10 für mail03 sind die nachfolgenden Befehle mit der IP Adresse von classix durchzuführen

```
Logging in to [iface: default, target: iqn.2011-09.de.jit-
    creatives:iscsi.drbd1, portal: 78.47.70.7,3260]
Login to [iface: default, target: iqn.2011-09.de.jit-creatives:
    iscsi.drbd1, portal: 78.47.70.7,3260]: successful
```

Wenn ein „successful“ zurück kommt, kann mit dmesg überprüft werden, ob die neue Festplatte, die jetzt übers Netz zur Verfügung steht, auch richtig erkannt wurden:

```
[492712.049972] scsi 5:0:0:0: Attached scsi generic sg1 type 0
[492712.113683] sd 5:0:0:0: [sda] 781233032 512-byte logical
    blocks: (399 GB/372 GiB)
[492712.115309] sd 5:0:0:0: [sda] Write Protect is off
[492712.115311] sd 5:0:0:0: [sda] Mode Sense: 77 00 00 08
[492712.118025] sd 5:0:0:0: [sda] Write cache: disabled, read
    cache: enabled, doesn't support DPO or FUA
[492712.124331] sda:
[492712.134239] sd 5:0:0:0: [sda] Attached SCSI disk
```

Über /dev/sda lässt sich nun die neue knapp 400 GB große Festplatte ansprechen. Die VM hat ihr Root auf /dev/vda (virtuelle Disk), daher heißt die neue Festplatte sda. Sollte es bereits ein sda geben, würde sdb gewählt werden usw. Mittels cfdisk oder fdisk wird auf der neuen Festplatte eine neue Partition /dev/sda1 angelegt.

OCFS2 Konfiguration

Nachdem nun auf den beiden VMs mail02/mail03 iSCSI funktioniert, ist der letzte Schritt die OCFS Konfiguration. Dazu werden auf beiden VMs die ocfs2-tools installiert sowie auf mail02 auch die ocfs2console. Die ocfs2console ist ein einfaches GUI Tool, mit dem der Cluster verwaltet wird. Damit wird die cluster.conf erstellt (siehe Listing 2.4).

Listing 2.4: /etc/ocfs/cluster.conf

```
node:
        name = mail02
        cluster = ocfs2
        number = 0
        ip_address = 78.47.9.22
        ip_port = 7777
node:
```

```
        name = mail03
        cluster = ocfs2
        number = 1
        ip_address = 46.4.156.93
        ip_port = 7777
cluster:
        name = ocfs2
        node_count = 2
```

Es ist auch möglich dieses per Hand zu tun, aber gerade für Anfänger auch gut, dass es ein entsprechendes Tool gibt. Die cluster.conf muss auch auf die andere Node, also mail03, kopiert werden. Nun wird auf mail02 das OCFS2 Dateisystem initialisiert, wobei -N 2 angibt, dass es 2 OCFS Nodes gibt :

Listing 2.5: Anlegen des OCFS2 Dateisystems

```
root@mail02:~# mkfs.ocfs2 -N 2 /dev/sda1
mkfs.ocfs2 1.4.4
Cluster stack: classic
o2cb Label:
Features: sparse backup-super unwritten inline-data strict-journal-super
Block size: 4096 (12 bits)
Cluster size: 4096 (12 bits)
Volume size: 399987105792 (97653102 clusters) (97653102 blocks)
Cluster groups: 3028 (tail covers 14190 clusters, rest cover 32256 clusters
    )
Extent allocator size: 201326592 (48 groups)
Journal size: 268435456
Node slots: 2
Creating bitmaps: done
Initializing superblock: done
Writing system files: done
Writing superblock: done
Writing backup superblock: 5 block(s)
Formatting Journals: done
Growing extent allocator: done
Formatting slot map: done
Writing lost+found: done
mkfs.ocfs2 successful
```

Damit der Cluster korrekt funktioniert, muss noch der Dienst gestartet werden, der sich um das Locking kümmert: der Distributed lock manager (DLM), Er wird zum ersten mal über /etc/init.d/o2cb enable aktiviert. Anschließen sieht man den extra Mountpoint

/dlm. Nun kann das neue Clusterdateisystem gemoutet werden.

```
root@mail02:~# mount /dev/sda1 /ocfs
root@mail02:~# mount
/dev/vda1 on / type ext3 (rw,errors=remount-ro)
tmpfs on /lib/init/rw type tmpfs (rw,nosuid,mode=0755)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
udev on /dev type tmpfs (rw,mode=0755)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode
    =620)
configfs on /sys/kernel/config type configfs (rw)
ocfs2_dlmfs on /dlm type ocfs2_dlmfs (rw)
/dev/sda1 on /ocfs type ocfs2 (rw,_netdev,heartbeat=local)
```

Der DLM muss auch auf mail03 gestartet werden; dann kann auch hier das OCFS gemoutet werden. Sollten Probleme auftreten, gibt es gute TroubleShooting-Seite unter [10].

Probleme

Leider ließ sich das gewünschte Setup, wie in Abbildung 2.3 zu sehen, nicht umsetzten. Zwar ließ sich das Clusterdateisystem auf beiden Nodes mounten und es ließen sich auch Daten schreiben, aber das führte jedes mal zu einem Split-Brain. Die beiden OCFS Clusternodes haben sich nicht gegenseitig gefunden, der DLM konnte daher nicht korrekt arbeiten. Eigentlich hätte es eine Ausgabe geben müssen, wie im Listing 2.7 zu sehen. Jede Node dachte jedoch sie wäre alleine im Cluster. Mehrere weitere Einstellungen, auch um Split-Brains des DRBDs automatisch zu beheben, brachten keine Lösung. Trotzdem wurde das primary-primary Setup dadurch verfeinert, siehe Listing 2.6. Für den Handler outdate-peer ist es nötig, dass beide DRBD Server sich per SSH direkt erreichen können; also wiederum mit SSH-Key ohne Passwortschutz, wie schon unter Abschnitt 2.1 beschrieben. Damit outdate-peer funktioniert und dieses Script „Shoot The Other Node In The Head“ (STONITH) aufführen kann, muss die Disk und das Clusterdateisystem fencing unterstützen. Dies wird hier mit fencing resource-and-stonith festgelegt. Sollten die Automatismen nicht funktionieren, kann ein Split-Brain natürlich auch per Hand beseitigt werden, indem der Administrator festlegt, welcher Node der „Überlebende“ ist [4].

Listing 2.6: common-Teil von /etc/drbd.d/global_common.conf

```
common {
        protocol C;
        handlers {
                pri-on-incon-degr "/usr/lib/drbd/notify-pri-on-incon-degr.
                    sh; /usr/lib/drbd/notify-emergency-reboot.sh; echo b >
                    /proc/sysrq-trigger ; reboot -f";
                pri-lost-after-sb "/usr/lib/drbd/notify-pri-lost-after-sb.
                    sh; /usr/lib/drbd/notify-emergency-reboot.sh; echo b >
                    /proc/sysrq-trigger ; reboot -f";
                local-io-error "/usr/lib/drbd/notify-io-error.sh; /usr/lib/
                    drbd/notify-emergency-shutdown.sh; echo o > /proc/sysrq
                    -trigger ; halt -f";
                outdate-peer "/usr/lib/drbd/outdate-peer.sh";
        }
        startup {
                wfc-timeout 0;
        }
        disk {
                 fencing resource-and-stonith;
        }
        net {
                max-epoch-size 8000;
                cram-hmac-alg sha1;
        }
        syncer {
                rate 50M;
                 verify-alg sha1;
        }
}
```

DRBD über 2x iSCSI und OCFS2 funktioniert also nicht. Dabei ist das Zusammenspiel zwischen iSCSI und OCFS wohl Schuld daran[11]. Was jedoch möglich ist, ist, dass beide Mailserver das iSCSI-Target von cloud9 mounten, siehe Abbildung 2.5, dann funktioniert auch OCFS ohne Probleme und man kann OCFS mounten und von beiden Systemen auf den Cluster zugreifen und Daten schreiben, natürlich auch gleichzeitig. Dann ist jedoch der Vorteil der DRBD Synchronisation weg, außer man lässt diesen im Master-Slave und schaltet im Fehlerfall per Hand um.

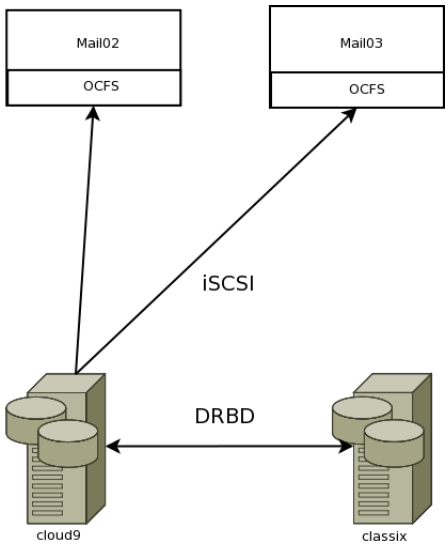

Abbildung 2.5: Setup mit iSCSI aber OCFS Mount nur über Server cloud9

Abbildung 2.6: Setup 2 nur mit DRDB und OCFS

```
version: 8.3.10 (api:88/proto:86-96)
srcversion: 24353649F58F784AA5458AB
 0: cs:Connected ro:Primary/Primary ds:UpToDate/UpToDate C r-----
    ns:390616516 nr:0 dw:0 dr:390616724 al:0 bm:23840 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:0
```

Abbildung 2.7: Ausgabe von cat /proc/drbd nachdem beide zum Master erhoben worden sind

2.2.2 Setup 2

Da Setup 1 nicht wie gewünscht funktionierte, wurde das Setup leicht abgewandelt und der iSCSI Teil wurde rausgenommen. Die DRBD Synchronisation wurde nur zwischen den 2 Mailservern vorgenommen und dort das OCFS gemountet, siehe Abbildung 2.6. Das DRBD wurde als Master-Master neu eingerichtet und das Setup angepasst. Die grundlegenden DRBD Einstellungen blieben gleich. Ausgetauscht wurde in der r0.res nur der 2. Server. Das Ziel ist jetzt direkt mail03, also die Virtuelle Maschine auf classix. Wiederum sollte sichergestellt sein, dass sich beide Hosts gegenseitig erreichen und per SSH verständigen können, damit der Split-Brain wargenommen und ggf. behoben werden kann. Nach der initialen Synchronisation wurde auf beiden Systemen das DRBD mit „drbdadm primary r0“ zum Master erhoben.

Ob alles funktioniert hat, lässt sich mit einem cat /proc/drbd überprüfen, siehe Ausgabe in Abbildung 2.7.

Im ersten Setup wurde in dem drbd blockdevice eine Partition erzeugt und darin das ocfs dateisystem gelegt. Um diese Verhaltensweise beizubehalten muss mit kpartx -a -v /dev/drbd0 der Linux Devicemapper dazu gebracht werden, die Partition als Device anzuzeigen. Ohne diesen Befehl wäre nur der Zugriff auf /dev/drbd0 möglich. Hier könnte direkt ein neues OCFS erstellt werden, aber ich wollte auf das schon existierende zugreifen. Der Devicemapper legt /dev/mapper/drbd0p1 an, also die erste Partition in drbd0. Nachdem die /etc/ocfs/cluster.conf angepasst und auf beide Nodes verteilt wurde, muss auf beiden Hosts nun ocfs gestartet werden. Das geschieht mit /etc/init.d/o2cb enable bzw. /etc/init.d/o2cb restart falls ocfs schon einmal benutzt wurde. Nun konnte das Clusterdateisystem auf beiden Nodes gemountet werden:

```
mount /dev/mapper/drbd0p1 /ocfs/
```

Ob auf beiden das Mounten richtig funktioniert hat, lässt sich mit der Ausgabe von dmesg überprüfen:

11 siehe http://www.gossamer-threads.com/lists/linuxha/users/61060

Listing 2.7: Ausgabe von dmesg / Kernel Logmeldungen

```
[77419.209849] o2net: connected to node cloud9 (num 0) at 78.46.70.7:7777
[77422.660956] ocfs2_dlm: Node 0 joins domain 9
    A2985A18A04469B825F476C9B4D2A4E
[77422.660959] ocfs2_dlm: Nodes in domain ("9
    A2985A18A04469B825F476C9B4D2A4E"): 0 1
```

Die letzte Zeile zeigt an, dass beide Nodes verbunden sind. Auch das ocfs mit seinen Testdaten ist noch vorhanden.

```
root@mail03:/ocfs# ls -al
insgesamt 1530636
drwxr-xr-x  5 root root       3896 12. Jan 11:53 .
drwxr-xr-x 25 root root       4096 13. Jan 09:43 ..
drwxr-xr-x  2 root root       3896 22. Nov 15:41 lost+found
-rw-r--r--  1 root root  160711168 12. Jan 09:06 test_12.01.img
drwxr-xr-x  2 root root       3896 11. Jan 18:56 test2.txt
drwxr-xr-x  2 root root       3896 12. Jan 11:53 test3
-rw-r--r--  1 root root 1348133376 12. Jan 11:57 test3.img
drwxr-xr-x  2 root root       3896 11. Jan 18:29 test_dir
-rw-r--r--  1 root root   58511872 11. Jan 18:22 test.img
-rw-r--r--  1 root root          0 11. Jan 18:22 test.txt
```

Das Setup ist fertig. Von beiden Nodes aus kann auf das Clusterdateisystem geschrieben werden. Nun kann es ausgiebig ans Testen gehen.

2.2.3 Tests

Um Filesysteme zu testen, gibt es einige Tools. Angefangen von klassischen Unix tools wie dd, iostat[12] und hdparm bis hin zu extra Benchmark-Tools wie tiobench[13], seekwatcher[14], bonnie++[15] und iozone[16]. Die meisten Tools, die ich gefunden habe, eigenen sich aber eher für den Test von lokalen Dateisystemen. Iozone bildet hier die Ausnahme und besitzt auch Schalter, die extra für NFS eingebaut wurden, daher wählte ich für die Tests iozone aus.

In den installierten Setups wurde auf Virtualisierung mit KVM gesetzt. Das bedeutet, dass viele virtuelle Maschinen auf ein oder zwei im Server verbaute Festplatten zugreifen.

12 bei debian im Paket sysstat enthalten
13 siehe http://linuxperf.sourceforge.net/tiobench/tiobench.php
14 siehe http://oss.oracle.com/~mason/seekwatcher/
15 siehe http://www.coker.com.au/bonnie++/
16 siehe http://www.iozone.org/

Eine Begrenzung ist also das I/O für den Festplattenzugriff. Bei den verteilten Dateisystemen spielt außerdem noch die Netzwerkanbindung eine große Rolle. Sind die Systeme im gleichen Rechnerschrank mit Gigabit angebunden? Oder ist es sogar ein SAN per Fibrekabel? Das Testsetup hatte nur eine 100 MBit/s Anbindung. Die Server standen in benachbarten Rechenzentren beim Hoster Hetzner. Es kann also angenommen werden, dass die 100 MBit/s auch ausgenutzt werden. Einer der physischen Server hatte keine weiteren Aufgaben und keine zusätzliche Last. Der zweite physische Server hatte mehrere Virtuelle Maschinen am Laufen, die auch alle Last und natürlich Festplattenzugriffe produzierten. Die einzelnen Tests wurden auf dem Server durchgeführt, der keine weiteren Aufgaben hatte, um sowenig Fehler wie möglich in den Messungen zu haben. Dieser Server hat folgende Merkmale:

- Prozessor: AMD Athlon 64 X2 6000+ (Dual Core)
- 2x 750GB Western Digital WD7502ABYS-0
- 6 GB DDR2 RAM
- Network Device: Realtek RTL8111
- Disk Scheduler: CFQ

Getestet wird das Setup 2. Daher spielt DRBD im Test eine Rolle iSCSI jedoch nicht.

Listing 2.8: sequentielles Lesen von Festplatte und DRBD

```
cloud9:~# hdparm -tT /dev/drbd0
/dev/drbd0:
Timing cached reads:   2148 MB in  2.00 seconds = 1074.79 MB/sec
Timing buffered disk reads: 272 MB in  3.02 seconds =  90.11 MB/sec

cloud9:/media/sdb2# hdparm -tT /dev/sda
/dev/sda:
Timing cached reads:   2318 MB in  2.00 seconds = 1159.89 MB/sec
Timing buffered disk reads: 290 MB in  3.00 seconds =  96.57 MB/sec
```

Der erste Test nutzt das Tool hdparm, welches zum Manipulieren von scsi/sata devices unter Linux benutzt werden kann und Lesegeschwindigkeiten von blockdevices messen kann. Die Ausgabe des Tools hdparm zeigte keinen großen Unterschied zwischen dem sequentiellen Lesen von der lokalen Festplatte sda und dem vom Blockdevice drbd0. Danach wurde das Tool dd eingesetzt, mit welchem auch Lese- und Schreibtests durchgeführt werden können. Dabei kam auch das Dateisystem zum Tragen. Per dd wurde

jeweils 1 GiB[17] (1024 * 1024 KB) auf die jeweiligen Dateisysteme geschrieben. Gelesen wurde dabei von /dev/zero, also wurden Null-Bytes generiert und geschrieben.

Listing 2.9: 1 GiB Nullen auf ext3 schreiben

```
cloud9:/# dd if=/dev/zero of=/data/1GB_null.img bs=1024k count=1024
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 3.98996 s, 269 MB/s
```

Listing 2.10: 1 GiB Nullen auf ocfs schreiben

```
cloud9:/# dd if=/dev/zero of=/ocfs/1GB_null.img bs=1024k count=1024
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 45.2451 s, 23.7 MB/s
```

Listing 2.11: 1 GiB Nullen in RAM schreiben

```
cloud9:/lib/init/rw# dd if=/dev/zero of=1GB_null.img bs=1024k count
    =1024
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 1.43542 s, 748 MB/s
```

Gerade beim Schreiben ist der Unterschied zwischen der lokalen ext3-Partition und dem OCFS deutlich zu sehen. Zum Vergleich ist auch der Wert für das Schreiben in den RAM angegeben. Die lokale Partition ist mehr als 10 Mal schneller. Das liegt jedoch nicht am Dateisystem OCFS an sich, sondern am Setup mit DRBD darunter. Erst wenn das DRBD den zu schreibenden Block auf beiden Servern geschrieben hat, nimmt es den nächsten entgegen; hier ist also auch die Latenz des Netzwerks einberechnet. Dafür ist eine Schreibgeschwindigkeit von 23,7 MB/s sogar als hoch anzusehen. Denn die 100 Mbit/s, die am Server anliegen, übertragen nur 12,5 MB/s. Hier sind also verschiede Caching Verfahren vom Dateisystem und Betriebsystem am Werk. Dies zeigt sich auch, wenn direkt auf Blockdevice, also direkt auf die Festplatte, geschrieben wird, statt auf das Dateisystem ext3.

```
cloud9:/# dd if=/dev/zero of=/dev/sdb2 bs=1024k count=1024
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 17.3905 s, 61.7 MB/s
```

17 GiB bedeutet gibibytes, GB steht für Gigabyte d. h. 1073741824 bytes = 1 GiB = 1,074 GB

Tests in MiB/sec	ext3	xfs	ocfs	ocfs (drbd inaktiv)
Initial write	564	1533	66	263
Rewrite	1189	1657	63	403
Read	3647	2912	1483	1324
Re-read	2277	3508	1427	1414
Reverse Read	3314	3133	1403	1277
Stride read	3097	2680	1440	1282
Random read	3754	2709	1433	1326
Mixed workload	3325	2421	1014	906
Random write	1421	1649	102	393
Pwrite	580	1537	42	240
Pread	2654	2627	1326	1326

Tabelle 2.1: Benchmark Vergleich von OCFS2, ext3 und xfs

Das ext3-Dateisystem ist immer noch mehr als 4 mal schneller als die reine Festplatte. Das variiert natürlich von Dateisystem zu Dateisystem, hängt von der Größe der Blöcke ab und von der internen Struktur der Dateisysteme, aber eben auch von den verschiedenen Caches wie dem Buffer Cache von Linux selbst [5].

Nach den Standardtools, die schon mal eine grobe Orientierung gegeben haben, wurde jetzt das expliziert für Filesystem Benchmark geschriebene Tool iozone verwendet. Dazu wird iozone in der Version 3.308 64bit verwendet. Getestet wurde mit iozone -R -l 5 -u 5 -r 4k -s 200m, das heißt, es wurden jeweils 5 Prozesse/Threads gestartet, die in 4k Blöcken jeweils 200 MB lesen und schreiben. 4K Blöcke eignet sich hier gut, da sowohl OCFS (siehe Listing 2.5 beim Anlegen des OCFS) als auch ext3 und xfs intern mit 4k Blöcken arbeiten. Die Option -R liefert am Ende eine Ausgabe, die in einer Tabellenkalkulation zur weiteren Überarbeitung genutzt werden kann [8, 1, 2]. Tabelle 2.1 zeigt die mit iozone gemessenen Werte; einmal für das OCFS, welches über DRBD synchronisiert wird, einmal OCFS bei dem die DRBD Synchronisation bewusst unterbrochen wurde[18], ein lokales ext3-Filesystem und eine lokales xfs-Filesystem. Die Werte sind für die jeweils 5 gestartet Prozesse kummuliert, daher sind sie wesentlich größer als beim Test mit dd. Die iozone-Tests wurde für alle Dateisysteme 3 Mal gefahren. Alle Tests lieferten ähnliche Ergebnisse, daher sind hier die Daten des ersten Tests verwendet worden. Iozone liefert die Ergebnisse in KB/sec, genauer gesagt sind es KiB/sec (1 KiB/sec ist gleich 1024 bytes/s). Die gemessenen Werte wurden in MiB/s (1 MiB/s = 1024 KiB/sec[19]) umgewandelt und dabei entsprechend gerundet. Ext3 ist in allen Punkten schneller als ocfs2.

18 d. h. mail03 war zu dem Zeitpunkt aus. Es konnte keine Synchronisation statt finden.
19 siehe auch http://en.wikipedia.org/wiki/Data_rate_units

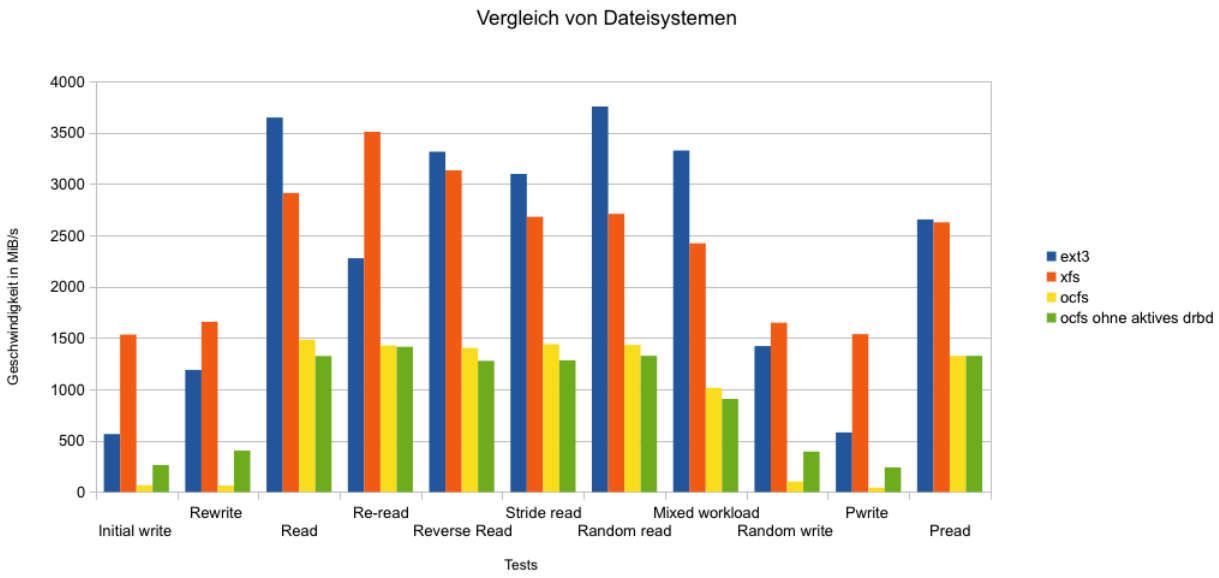

Abbildung 2.8: Vergleich der getesteten Dateisysteme beim Lesen und Schreiben von 200 MB in 4K Blöcken

Das liegt im erwarteten Rahmen und zeigte sich schon bei den Messungen mit dd. Ext3 ist beim Schreiben ca. 13 Mal schneller und beim Lesen ca. 2-3 Mal schneller. Zum Vergleich, wie sich das darunter liegende DRBD auswirkt, wurde auch ein Test gemacht, bei dem mail03 ausgeschaltet war; damit kann die DRBD Synchronisation nicht funktionieren und ocfs2 ist nicht vom langsamen Schreiben über Netzwerk abhängig. Der Effekt zeigt sich beim Schreiben deutlich - so ist das ocfs nun 4 Mal schneller als vorher. Beim Lesen hat das Unterbrechen der DRBD Sychronisation erwartungsgemäß keinen Effekt, wie sich auch in Abbildung 2.8 zeigt. Da Ceph leider nicht getestet wurde, lässt sich nichts darüber aussagen, ob OCFS nun ein besonders schnelles Clusterdateisystem ist oder nicht. Im Gegensatz zu den lokalen Dateisystemen ext3 und xfs schneidet es jedoch schlechter ab, wobei das beim Schreiben besonders deutlich ist.

2.2.4 Mailserver

Das Testsetup soll für einen Mailserver gedacht sein. Genauer gesagt für einen Mail Transfer Agent (MTA)[20]. Ich setzte dafür Postfix ein. Beide vorgestellten Setups wären möglich. Das zweite wird jetzt einen längerem Test unterzogen, ob die DRBD Master-

20 MTAs nehmen Mails an und leiten diese weiter. Siehe http://de.wikipedia.org/wiki/Mail_Transfer_Agent um Mails per IMAP oder POP3 abholen zu können wird ein Mail Retrieval Agent wie Cyrus oder Dovecot benötigt

Master Replikation auch über längere Zeit sauber läuft. Das weltweite Mailsystem ist bestens dafür geeignet, da es schon von Haus aus Loadbalancing mitbringt und zwar über DNS. Gibt man im DNS Server einfach mehrere MX Records an, werden aussendende Mailserver zuerst den mit der niederen Priorität versuchen, sollte dieser nicht erreichbar sein oder temporär zuviel Last haben, wird der 2. MX Record genommen. Solange also nicht beide Mailserver ausfallen kommt die Mail an. Getestet wurde mit der Domain savar.de, die 2 MX Records hat.

```
~/ dig mx savar.de
; <<>> DiG 9.6-ESV-R4-P3 <<>> mx savar.de
;; global options: +cmd ;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 51476
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;savar.de.                      IN      MX

;; ANSWER SECTION:
savar.de.               10400   IN      MX      5 mail03.jit-creatives.de.
savar.de.               10400   IN      MX      0 mail02.jit-creatives.de.
```

Dazu wird auf den Mailserver /var auf das OCFS gelegt. Ankommende Mails werden ihn ihren Ordner unterhalb von /var/spool einsortiert. Getestet wird erst mal nur das Empfangen von Mails. Ein Hochverfügbarkeitssetup für das Abholen von Mails (via POP3 oder IMAP) benötigt eine noch größere Konfiguration. Hierfür ist wieder Clustersoftware nötig, die auch einen IP Failover unterstützt, oder es wäre ein IMAP Loadbalancer nötig, der jedoch natürlich wieder ein Single-Point-of-Failure wäre, wenn er nicht redundant ausgelegt ist.

3 Fazit und Lessons Learned

Ein eigenes kleines Storage mit verteiltem Dateisystem aufzuziehen, ist doch schwieriger, als ich das auf den ersten Blick vermutet hätte. Ich kann verstehen, warum sich Unternehmen lieber ein SAN kaufen und sich an einen Hersteller binden, den man aber dann auch für alles verantwortlich machen kann und der Support leisten muss. Das Setup muss wohl überlegt sein und es gibt viele Entscheidungen zu treffen. Es gibt viele Möglichkeiten in der Open Source und Linux-Welt etwas zusammen zu stecken, aber es muss am Ende eben auch auf Dauer funktionieren. Beim Einrichten auf dem Server gibt es viel zu lernen und auch einige Stolpersteine. Beim ersten Versuch klappt es meistens gar nicht, gerade wenn viele verschiedene Dienste wie iSCSI, DRBD und OCFS auch noch richtig zusammen arbeiten sollen. Für kleinere Unternehmen wiederum wird es sich aber doch lohnen, sich die Technologien anzuschauen und wenn der eigene Systemadministrator sich erst mal das Know-how angeeignet hat, läuft auch ein OCFS wie geölt.

Das Clusterfilesystem Ceph ist noch jung, zu jung für einen produktiven Einsatz. Der Ansatz jedoch ist sehr viel versprechend: Verschiedene Nodes, Redundanz und viele Festplatten werden zu einem großen Image zusammen geführt. Auch dass Ceph schon im Linux Kernel enthalten ist, ist ein guter Ausgangspunkt. Wie man am Beispiel von Virtualisierungslösungen Xen und KVM gesehen hat, kann eine Technologie auch schnell an Bedeutung verlieren, wenn sie nicht im Linux-Kernel ist, aber das ihr Hauptseinsatzort ist[21]. Es bleibt zu hoffen, dass die Entwicklung von Ceph schnell voran schreitet. Bei meinen Tests ist das System jedoch sehr instabil gewesen und lies sich nicht über längere Zeit verwenden, was mich stark enttäuscht hat.

Da Ceph noch nicht ausgereift ist, mussten die alten Hasen ran. Für den Prototyp des Mailservers habe ich OCFS genommen. Im Zusammenspiel mit DRDB ist das eine Technologie, die vielfach erprobt ist und damit nahezu rocksolid ist. Was Ceph mit seiner Redundanz schon eingebaut hat, muss bei OCFS2 allerdings nachgerüstet werden; mit DRBD oder einer anderen Technik. Nur so hat man auch die Ausfallsicherheit und kann von Hochverfügbarkeit sprechen. Das Testsystem kann noch um weitere Techniken

21 XEN und KVM sind Virtualisierungstechnologien. Während XEN älter ist und lange der Standard war verhindert eine Streit mit den Kernelentwicklern lange die Aufnahme. KVM wurde früh in den Linux-Kernel integriert und hat seit dem einen kometenhaften Aufstieg erlebt.

erweitert werden um z. B. einen automatischen Failover zu erreichen, bei dem dann im Fehlerfall auch die IP des Mailservers geändert wird. Dazu sind dann Hochverfügbarkeitstechnologien wie Pacemaker und corosync nötig[7]. Diese Erweiterungen werde ich mir auch anschauen und ggf. in einem weiteren Testsetup austesten. Für den Prototyp war es erst mal nur wichtig, dass die Mails jederzeit ankommen, auf einen Speicher geschrieben werden, und dies auch dann, wenn einer der Server ausfällt. Außerdem wollte ich die dahinterstehenden Technologien und Ideen selbst austesten und aufsetzten. Der nächste Schritt mit IP Failover ist dann die Kür. Der Failover von IPs ist aber bei größeren Hostern wie Hetzner entweder nicht möglich oder muss extra bezahlt werden. Für komplizierte Setups bleibt wohl nur die Möglichkeit sich einige Server in ein Rechenzentrum zu stellen, also Serverhousing[22] statt Serverhosting.

Aber nicht nur Ceph ist ein interessantes Clusterdateisystem; auch das in Berlin unter der GPL entwickelte und von der EU gesponserte xtreemfs[6] macht einen guten Eindruck. Es ist ein replizierendes, verteiltes Dateisystem und kann von Linux, Windows und Mac OS angesprochen werden. Für die Administration wird Java benötigt, außerdem ist es unter Linux nur über FUSE[23] angebunden und daher potentiell nicht so schnell wie Ceph. Auch lustrefs ist ein interessanter Kandidat. Das mittlerweile zu Oracle gehörende Cluterfilesystem wird auf mehren Supercomputern eingesetzt. Leider hat Oracle nach dem Kauf von SUN nicht das nötige Interesse das unter GPL stehende System groß zu vermarkten. Es läuft eigentlich nur auf Oracle Hard- und Software. Die Dokumentationen, wie man es auf Debian zum Laufen zu bekommen kann, sind veraltet[24]. Daher konnte ich dieses Dateisystem ebenfalls nicht testen. Es gibt mittlerweile einige Unternehmen, die lustre weiterentwickeln wollen[15]. Auch lustre ist jedoch nicht direkt im Linux Kernel enthalten Um es einsetzen zu können muss man den Linux Kernel entsprechen patchen. Diese Patches müssen u.U. für jede Linux-Kernel-Version angepasst werden. Das ist ganz klar der Vorteil von Ceph, da es ein Kernelmodul benutzt und aktiv im Linux Kernel weiterentwickelt wird. Sollten xtreemfs und/oder lustre nicht in den Linux-Kernel kommen, wird langfristig Ceph das Rennen machen und ein Standard für moderne Clusterfilesysteme in der Linux-Welt werden.

22 siehe http://de.wikipedia.org/wiki/Serverhousing
23 Filesystem in Userspace
24 siehe http://wiki.debian.org/Lustre

Literaturverzeichnis

[1] Nitin Agrawal, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Generating Realistic Impressions for File-System Benchmarking. *ACM Transactions on Storage*, 5(4), December 2009.

[2] Don Capps. Iozone filesystem benchmark. `http://www.iozone.org/docs/IOzone_msword_98.pdf`.

[3] e100. Update drbd userland to 8.3.10 to match kernel in 1.9, 2011. `http://forum.proxmox.com/threads/7059-Update-DRBD-userland-to-8-3-10-to-match-kernel-in-1-9`.

[4] Florian Haas, Philipp Reisner, and Lars Ellenberg. The drbd user's guide. `http://www.drbd.org/users-guide-8.3/users-guide.html`.

[5] M. Tim Jones. Anatomy of the linux file system. `http://www.ibm.com/developerworks/linux/library/l-linux-filesystem/`.

[6] Björn Kolbeck and Jan Stender. xtreemfs - the team. `http://www.xtreemfs.org/about.php`.

[7] Oliver Liebel. *Linux Hochverfügbarkeit: Einsatzszenarien und Praxislösungen*. Galileo Computing. Galileo Press GmbH, 2010.

[8] Ramesh Natarajan. How to measure linux filesystem i/o performance with iozone, 2008. `http://www.cyberciti.biz/tips/linux-filesystem-benchmarking-with-iozone.html`.

[9] Udo Seidel. Server-Traube - GFS2 und OCFS2, zwei Cluster-Dateisysteme im Linux-Kernel. `http://www.linux-magazin.de/Online-Artikel/GFS2-und-OCFS2-zwei-Cluster-Dateisysteme-im-Linux-Kernel`.

[10] Thorsten Staerk. Set up an ocfs2 cluster filesystem. `http://linux.dell.com/wiki/index.php/Set_up_an_OCFS2_cluster_filesystem`.

[11] Ceph Team. Designing a cluster. `http://ceph.newdream.net/wiki/Designing_a_cluster`.

[12] Wikipedia. Drbd. `http://de.wikipedia.org/wiki/DRBD`.

[13] Wikipedia. List of file systems. `http://en.wikipedia.org/wiki/List_of_file_systems` abgerufen 08.06.2011.

[14] Wikipedia. Ceph — wikipedia, the free encyclopedia. 2011. `//en.wikipedia.org/w/index.php?title=Ceph&oldid=454151415`.

[15] Wikipedia. Lustre (file system) — wikipedia, the free encyclopedia, 2012. `http://en.wikipedia.org/w/index.php?title=Lustre_(file_system)&oldid=468865976`.

Lizenz