



Global File System Storage Cluster 6 Aus dem Pool schöpfen

Mark Hlawatschek, Marc Grimme

Im Zeichen der Globalisierung müssen IT-Services rund um die Uhr verfügbar sein – Clustering ist hier oft das Mittel der Wahl. Red Hat bietet mit seinem kürzlich unter die GPL gestellten Global File System eine freie Lösung, die mehrere Linux-Server und ein angebundenes SAN zu einem Single System Image auf Dateisystembasis vereint.

Was die Welt des High Performance Computing erfolgreich vorgemacht hat, hält in letzter Zeit vermehrt Einzug in traditionelle IT-Bereiche. Statische Dienste lassen sich gut auf mehreren Servern parallel betreiben, ein Lastverteiler weist ihnen die Benutzeranfragen zu. Während man bei diesem Szenario die dazugehörigen Daten per Replikation auf alle Knoten verteilt, benötigen dynamische Dienste wie Datenbanken oder File-Services einen von allen Knoten aus les- und beschreibbaren Datenbestand. Ein traditioneller Ansatz hierfür ist die Nutzung eines File-Servers als NAS (Network Attached Storage). Auch wenn dieser für manche Anwendungen eine adäquate Lösung ist, stellt der NAS oft den Flaschenhals und einen Single Point of Failure (SPoF) des Gesamtsystems dar.

Um die genannten Schwierigkeiten zu umgehen, sollte jeder Server in dem Cluster direkten Zugriff auf die Speichergeräte haben und diese konkurrierend gleichermaßen lesen und beschreiben kön-

nen. Red Hats Global File System (GFS) kann als Clusterdateisystem das Herzstück der Lösung bilden. Es verbindet Linux-Server und ein angebundenes SAN (Storage Area Network) zu einem Single System Image (SSI) auf Dateisystembasis.

Einige GFS-Internas

GFS ist ein derzeit ausschließlich unter Linux laufendes Journaling-Dateisystem. Es wurde von Grund auf als 64-Bit-Cluster-Dateisystem entworfen und verschafft mehreren Servern mit Standard-Unix-Semantik parallelen Zugriff auf ein gemeinsames Speichergerät. Die GFS-Geschichte begann 1995 an der Universität von Minnesota unter der Leitung des damaligen Professors Matthew O'Keefe. Die Entwicklung von Software für Supercomputing-Cluster brachte das Problem mit sich, dass viele Knoten große Datenmengen generierten, die auf elegante Weise auf einen zentralen Speicherpool ge-

schrieben werden mussten. Um die Entwicklung von GFS weiter voranzutreiben, gründete O'Keefe 1997 die Firma Sistina, Inc. Bis zur Version 4.11 stand GFS unter der GPL. 2001 stellte Sistina das Lizenzmodell für Version 4.2 von GPL auf die Sistina Public License um. Mit Version 5 stellte Sistina die Open-Source-Lizenz vollständig ein, was eine starke Diskussion in der Entwicklergemeinschaft hervorrief. Um die offene Entwicklung weiter vorantreiben zu können, entstand auf Basis der letzten GPL-Version das Projekt „openGFS“. Im Juni 2004 übernahm Red Hat Sistina und stellte auf Grund ihrer Lizenzpolitik Version 6 wieder unter die GPL.

Jeder Clusterknoten bekommt ein eigenes Journal zugewiesen. Standardmäßig schreibt GFS Änderungen an den Dateisystemmetadaten zuerst in ein Journal und danach erst auf das Dateisystem. Dies gewährleistet bei Ausfall eines Knotens die Dateisystemkonsistenz. Optional lässt sich die Protokollierung auch auf Data Journaling erweitern.

GFS speichert seine Daten in so genannten „Dynamic Inodes“ (Dinodes). Diese belegen immer einen ganzen Dateisystemblock (Default: 4096 Bytes). Da in einem Cluster immer mehrere Server auf das Dateisystem zugreifen, würde die Zusammenlegung mehrerer Dinodes in einem Block zu



- Red Hats Global File System ist ein clusterfähiges Journaling-Dateisystem für Linux.
- Die Nutzung eines Datenpools erübrigt die in klassischen Ansätzen oft erforderliche Datenreplikation auf alle Knoten.
- Über das Shared-Root-Konzept lassen sich Cluster mit vollständiger zentraler Datenhaltung und Konfiguration realisieren.

mehr konkurrierenden Blockzugriffen führen. Zur Vermeidung unnötiger Platzverschwendung speichert GFS Nutzdaten zunächst im Dinode selbst – der Zugriff auf eine kleine Datei benötigt also nur einen Blockzugriff. Bei wachsender Größe nutzt GFS eine „Flat File“-Struktur. Alle Pointer in einem Dinode haben die gleiche Tiefe: Entweder existieren nur direkte, nur indirekte oder nur doppelt indirekte Pointer. Die Höhe des Baums wächst so weit wie für die Speicherung einer Datei erforderlich.

Für die Verzeichnisstruktur dient „Extendible Hashing“ (ExHash). Für jeden Dateinamen speichert GFS einen Multibit-Hash als Index in der Hash-Tabelle. Die zugehörigen Pointer in der Tabelle zeigen auf einen Blattknoten („Leaf Node“), wobei mehrere Zeiger jeden Blattknoten referenzieren können. Wird die Hash-Tabelle zu klein, um die Verzeichniseinträge zu speichern, verdoppelt GFS ihre Größe. Ist ein Blattknoten zu klein, spaltet er sich in zwei gleich große Blattknoten auf. Die eine Hälfte der Pointer der Hash-Tabelle zeigt auf den ersten Blattknoten, die andere auf den zweiten. Existieren genügend wenig Einträge, landen die Verzeichniseinträge wie bei Dateien im Dinode-Block. Durch das Abbild der Hash-Bits lässt sich ein Verzeichniseintrag schnell finden.

In der aktuellen Version 6 bietet GFS zusätzlich File Access Control Lists (ACLs), clusterübergreifende Quota-Unterstützung, Direct I/O und dynamische Online-Vergrößerung der Datenkapazität.

Aufbau eines Storage Cluster

In einer GFS-Storage-Cluster-Infrastruktur stellen ein oder mehrere eigenständige Speichergeräte den so genannten Datenpool dar und sind mit einem SAN über

einen oder mehrere Datenpfade verbunden. Die einzelnen Server des Clusters greifen so – ebenfalls über einen oder mehrere Datenpfade an das SAN angebunden – direkt auf die Speichergeräte zu.

Server im GFS Storage Cluster arbeiten unter Linux. Ein einfacher Volume Manager, der GFS Pool Layer, virtualisiert die Speichergeräte (*/dev/sda*) und stellt diese hardwareunabhängig unter */dev/pool/<wasauchimmer>* bereit. Dabei lassen sich mehrere Devices eines Pools per Striping oder linearer Verknüpfung zusammenfassen. Änderungen an der Poolkonfiguration sehen alle Clusterserver. Der Volume Manager sorgt für die Online-Erweiterbarkeit des Dateisystems. Pool-Devices bilden die Grundlage des GFS.

Da in einem GFS Storage Cluster viele Server auf dieselben physikalischen Datenblöcke zugreifen, gibt es eine Instanz zur Koordination der verteilten Zugriffe – den so genannten „Lock-Service“. Dieser gewährleistet die Datenkonsistenz des Dateisystems. GFS verfügte von Anfang an über eine modulare Locking-Schicht. Anfänglich tauschte es die Locking-Information über das SCSI-Protokoll aus (DLOCK, DMEP). Seit GFS Version 5 nutzt es den redundanten, IP-basierten Userspace-Dienst GULM (Global Universal Lock Manager), der auf allen GFS-Knoten laufen kann. Red Hat arbeitet momentan an der Integration seines Distributed Lock Managers (DLM), wobei die openGFS-Gruppe auf die Integration des openDLM setzt.

Jeder Server des Storage-Clusters besitzt ein Heartbeat-Lock, das er regelmäßig aktualisieren muss. Ist er dazu nicht in der Lage, erkennen ihn die anderen Server oder der Lock-Manager als defekt und entfernen ihn aus dem Cluster. Den Mechanismus, anfänglich STOMITH (Shoot the other machine in the head)

benannt, taufen die Entwickler inzwischen politisch korrekt in Multi-Fencing um. Das Wort „Multi“ steht für die Option, verschiedene Fencing-Mechanismen nacheinander auszuprobieren, bis der defekte Server tatsächlich aus dem Cluster entfernt ist. GFS unterstützt hier verschiedene Network Power Switches, Fibre Channel Switches, die HP-RIOE-Karte (Remote Insight Lights-Out Edition) sowie das eigene Protokoll GNBD (Global Network Block Device).

Skalierbarkeit und Datensicherung

Klassische auf einzelnen Servern laufende IT-Systeme lassen sich nur beschränkt skalieren, da man an Grenzen ihrer Leistungsfähigkeit und Finanzierbarkeit stößt. Auf einem Storage-Cluster parallel laufende Anwendungen lassen sich dagegen um einiges leichter erweitern. Bei Leistungengpässen integriert man neue Bausteine (Server, Storage) in das System, bis die gewünschte Leistung beziehungsweise Kapazität erreicht ist. Die gemeinsame Nutzung eines Speicherpools erspart nicht nur die mühsame synchrone Datenreplikation auf mehrere Server, sondern bietet darüber hinaus

auch elegante Skalierungsmöglichkeiten. Bei wachsendem Speicherbedarf vergrößert man einfach den Pool, der sofort allen Servern zur Verfügung steht.

Eine Datensicherung erfolgt typischerweise entweder über das LAN auf einen dedizierten Backupserver, „LAN-Free“, das heißt von den Applikationsservern direkt auf ein Datensicherungsgerät oder „Server-Less“ – von einem zentralen Speichergerät über einen so genannten Datamover auf das Backup-Device. Der GFS Storage Cluster unterscheidet nicht zwischen „LAN-Free“- und „Server-Less“-Backup. Da in einem Clusterfilesystem jeder angebundene Server gleichermaßen auf alle Daten zugreifen kann, lässt sich ein Server zu einem GFS Datamover machen. Dieser kann während des Betriebs eine Sicherung der Daten durchführen, ohne dass die Applikationsserver davon betroffen sind. Genauso lassen sich über APIs der Speichergeräte Snapshots oder Clones erstellen, die man auf dem GFS Datamover einhängen und sichern kann. GFS bietet hierfür einen so genannten Dateisystem-Freeze, um einen konsistenten Datenbestand zu gewährleisten. Das „Freeze“ bedeutet hierbei, dass GFS alle Dateisystemzugriffe beendet

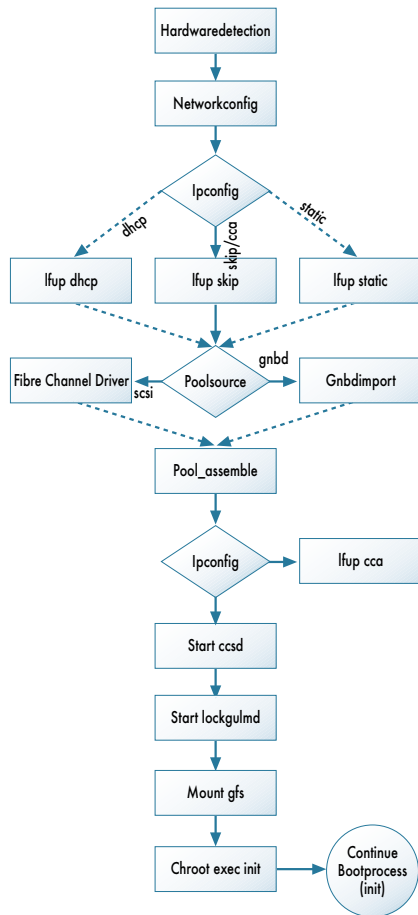
Listing 1

```
ls -l /cdsl.local /cluster /cluster/cdsl/ /cluster/cdsl/default/
lrwxrwxrwx 1 root root 22 Aug 5 15:56 /cdsl.local -> cluster/cdsl/@hostname

/cluster:
total 8
drwxr-xr-x 7 root root 3864 May 20 04:30 cdsl
drwxr-xr-x 3 root root 3864 May 19 03:29 shared

/cluster/cdsl/:
total 44
lrwxrwxrwx 1 root root 27 Aug 5 15:56 blade1-1 -> blade1-1.iptech.localdomain
drwxr-xr-x 6 root root 3864 May 20 03:23 blade1-1.iptech.localdomain
lrwxrwxrwx 1 root root 27 Aug 5 15:56 blade1-2 -> blade1-2.iptech.localdomain
drwxr-xr-x 6 root root 3864 May 19 23:45 blade1-2.iptech.localdomain
lrwxrwxrwx 1 root root 27 Aug 5 15:56 blade1-5 -> blade1-5.iptech.localdomain
drwxr-xr-x 6 root root 3864 May 19 23:45 blade1-5.iptech.localdomain
lrwxrwxrwx 1 root root 27 Aug 5 15:57 blade1-6 -> blade1-6.iptech.localdomain
drwxr-xr-x 6 root root 3864 May 19 09:03 blade1-6.iptech.localdomain
drwxr-xr-x 6 root root 3864 May 19 09:03 default
lrwxrwxrwx 1 root root 7 Aug 5 16:04 localhost -> default
lrwxrwxrwx 1 root root 7 Aug 5 16:04 localhost.localdomain -> default

/cluster/cdsl/default/:
total 528
drwxr-xr-x 24 root root 2048 May 20 01:20 dev
drwxr-xr-x 3 root root 3864 May 20 03:24 etc
drwxrwxrwt 28 root root 3864 May 19 03:17 tmp
drwxr-xr-x 25 root root 3864 May 19 03:21 var
```

Beim Starten über das Netz muss die Initial-RAM-Disk vor dem „normalen“ Boot-Prozess dafür sorgen, dass alle GFS-Voraussetzungen erfüllt sind.

Symbolic Links (CDSL) oder Context Dependent Path Names (CDPN) erfüllen diese Aufgabe. Ein gutes Beispiel für eine Datei, die unter Linux für jeden Knoten einzigartig sein muss, ist */etc/mtab* mit der Liste der aktuell eingehängten Dateisysteme. Für Shared-Root-Cluster ist es wichtig zu wissen, welche Dateien für jeden Knoten einzigartig sein müssen und welche nicht. Besteht der Cluster beispielsweise auch aus Knoten unterschiedlicher Hardwarearchitekturen, steigt die Komplexität noch zusätzlich. CDPN müssen sich also sowohl hinsichtlich der einzelnen Knoten als auch hinsichtlich der Architektur unterscheiden. GFS und andere

Cluster-Dateisysteme implementieren CDPNs als symbolische Links. Ein CDPN enthält eine Variable, die erst zur Zugriffszeit aufgelöst wird, der Kasten „CDPN-Variablen“ zeigt die von GFS unterstützten. Um die Übersicht nicht zu verlieren – es kann sich durchaus um viele Dateien handeln –, empfiehlt es sich, alle clusterknotenabhängigen Dateien in ein spezielles Verzeichnis zu legen. Listing 1 zeigt ein Beispiel mit nur einer Architektur.

init im Shared-Root-Cluster

Nicht weniger anspruchsvoll gestaltet sich der Boot-Prozess von Shared-Root-Clustern. Es soll an dieser Stelle nicht darauf eingegangen werden, wie man über das Netz einen Kernel mit initialer RAM-Disk (Initrd) bootet, sondern wie der Systemstart beim Einsatz von GFS viel häufiger vorkommenden Szenario von Speichernetzen auf Fibre-Channel-Basis funktioniert. Da Fibre Channel sowohl ein blockbasiertes als auch ein Netzprotokoll ist, erfolgt der Datenzugriff meistens via SCSI-III. Fibre-Channel-Controller, kurz FC-HBAs, verhalten sich dabei wie normale SCSI-Controller. Das heißt, die FC-HBAs sind bootfähig und lassen sich mit wenigen Handgriffen dazu bewegen, vom SAN zu starten. Zuvor sollte man eine Bootdisk im SAN anlegen. Dies erfolgt analog zu lokalen Bootdisks: Einen Bootloader wie Grub oder Lilo aufsetzen, den Kernel und das passende Initrd-Image darauf kopieren – fertig.

Der anspruchsvolle Part dieser Konfiguration ist wohl das Initrd-Image. Der Kernel lädt dieses vor dem Einhängen des Root-Dateisystems, hängt es als RAM-Disk im Speicher ein und führt die Datei */linuxrc* aus. Durch Kernelparameter lässt sich dieser Prozess relativ modu-

lar anpassen. Die Initrd hat nun die Aufgabe, alle Abhängigkeiten zum Mounten des Wurzeldateisystems vorzubereiten, und danach die Aufgabe des Root-Mount wieder an den Kernel zurückzugeben. Da es sich jedoch bei GFS um ein spezielles clusterfähiges Dateisystem handelt, muss in diesem Fall Initrd auch den Boot-Prozess in Gang setzen. Vorher sorgt das Initrd aber dafür, dass alle Voraussetzungen für die Funktion von GFS erfüllt sind. Abbildung 1 zeigt ein Schema dieses Prozesses. Funktioniert Initrd reibungslos, erkennt man schnell, ob das Shared-Root-Konzept richtig aufgesetzt ist.

Will man nun einen Server austauschen oder zum Cluster hinzufügen, funktioniert dies mit wenigen Handgriffen. Es sei jedoch erwähnt, dass man gerade bei Software-Updates sehr aufpassen muss: weder RPM- noch *.deb-Pakete sind clusterfähig. Die Folge sind Anpassungen, besonders bei Kernel-Updates.

Eine universell einsetzbare Schnittstelle für clusterweite Konfigurationsdaten bietet GFS mit dem Cluster Configuration Archive (CCA). Dieses ist ein Storage-Pool, der für GFS die Clusterkonfiguration enthält. Ein Server (*ccsd*) sorgt dafür, dass alle Knoten dieselben Konfigurationen „sehen“ und das Archiv konsistent bleibt. Es enthält Informationen über jeden Node genauso wie über den Aufbau des Clusters und die Fencing-Konfiguration. An sich erwartet GFS in diesem Archiv Dateien nach einer speziellen Syntax. Hält man sich an sie, kann man auch eigene Konfigurationsparameter in diesem Archiv ablegen; *ccs_read* bietet eine einheitliche Schnittstelle zum Zugriff darauf. Für Shared-Root-Cluster sind beispielsweise einige nicht von GFS spezifizierte Parameter notwendig – wie der Storage-Pool für das gemeinsame Root-Verzeichnis oder die IP-Konfiguration.

Listing 2 zeigt eine solche Konfiguration für einen Knoten. Der Parameter *com_ip_autoconfigure* steuert die automatische Netzkonfiguration, die wiederum der Abschnitt *ip_interfaces* definiert. Dies erspart gegebenenfalls den DHCP-Server. *com_sharedroot* nennt den Pool, auf dem das Shared-Root zu finden und einzuhängen ist, *com_role* definiert die Rolle des Knotens. Unterschiedliche Aufgaben von Servern lassen sich über Rollen definieren. Die Webserver starten dann die in ihrer Rolle definierten Services und die Mailserver alle Services der ihrigen. Dies sind Beispiele für Parameter, die GFS im Gegensatz zu *ip_interfaces*, *fence* et cetera zwar nicht fordert, die aber gerade bei Shared-Root-Clustern das Management stark erleichtern und den automatisierten Bootprozess der Knoten ermöglichen. (avr)

MARK HLAWATSCHEK

MARC GRIMME

sind Geschäftsführer der Atix GmbH in Unterschleißheim und beraten unter anderem im Bereich Linux-Enterprise-Storage-Lösungen.

Literatur

- [1] Kenneth W. Preslan, Andrew P. Barry, Jonathan E. Brassow, Grant M. Erickson, Erling Nygaard, Christopher J. Sabol, Steven R. Soltis, David C. Teigland and Matthew T. O’Keefe, A 64-bit, Shared Disk File System for Linux, 1999
- [2] Red Hat GFS 6.0 – Administrator’s Guide; www.redhat.com/docs/manuals/csgfs/admin-guide/
- [3] Kenneth W. Preslan; GFS – Evolution and changes between GFS 4.2 to today; sources.redhat.com/cluster/events/summit2004/kwp.gfs.pdf