Ausfallsichere Systeme mit KVM-Cluster

Fangnetz

Den Ausfall wichtiger Computersysteme kann sich heute keine Firma mehr erlauben. Hochverfügbare Cluster-Umgebungen gehören somit zum Standard einer sicheren Systemlandschaft. Der Einsatz von Virtualisierungs-Technologien hilft kostbare Ressourcen sparen. Thorsten Scherf

Beim Einsatz von Clustersystemen ist zwischen mehreren Varianten zu unterscheiden. Auf der einen Seite existieren die so genannten High-Availability-Cluster (HA), sie sorgen für eine möglichst hohe Verfügbarkeit eines bestimmten Dienstes. Fällt ein System aus, wandert der darauf laufende Dienst einfach auf einen anderen Clusterknoten. Ein Benutzer bekommt von dem Ausfall eines Systems somit überhaupt nichts mit.

Auf der anderen Seite sorgen so genannte Load Balancer für einen Lastenausgleich zwischen den verschiedenen Systemen. Üblicherweise leiten hier redundante Frontend-Systeme die eingehenden Anfragen an mehrere Backend-Systeme weiter. Je nachdem, welcher Algorithmus zum Einsatz kommt, verteilen die im Vordergrund laufenden Systeme ankommende Anfragen beispielsweise im Round-Robin-Verfahren an die eigentlichen Server im Hintergrund.

Es gibt auch Verfahren, bei denen der Admin anhand der Load der Systeme entscheidet, wohin eine Anfrage wandert. Fällt eines dieser Systeme aus, entfernen die Frontend-Router es aus ihren Routingtabellen. So ist sichergestellt, dass keine Pakete im Nirwana landen.

Schließlich existieren auch noch die so genannten High-Performance-Cluster (HPC), bei ihnen kommt es auf möglichst große Rechenleistung an. Um diese zu erreichen, unterteilt der Admin eine anstehende Aufgabe einfach in mehrere Happen und verteilt sie dann auf die einzelnen Systeme im Cluster. Somit

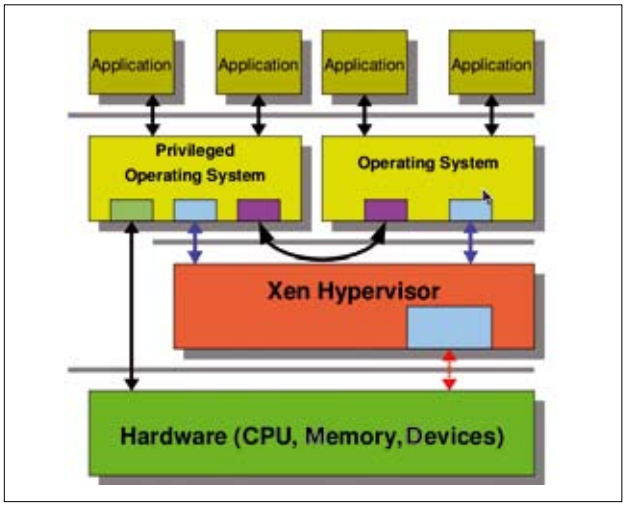

Abbildung 1: Bei Xen bauen alle virtuellen Maschinen auf einem privilegierten Managementsystem auf. Dabei ist dieses System selbst bereits auch schon virtualisiert.

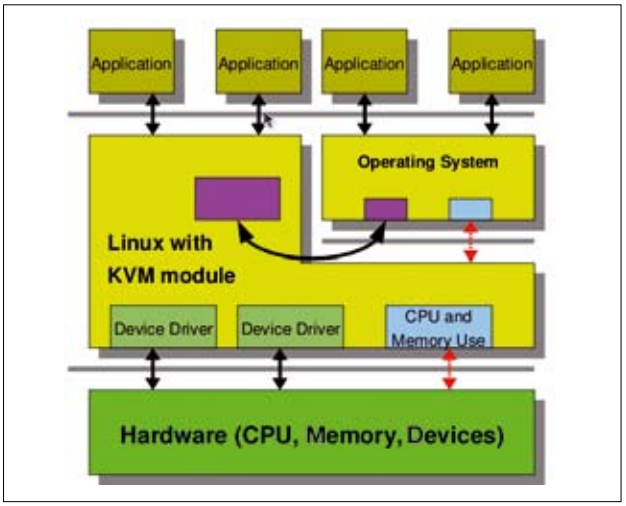

Abbildung 2: KVM ist durch ein einzelnes Kernelmodul implementiert. Alle virtuellen Maschinen laufen somit als reguläre Prozesse auf einem herkömmlichen Linux-Kernel.

stehen sämtliche Ressourcen (CPU, Speicher, Disks) allen Knoten des Clusters zur Verfügung. Solche Systeme kommen immer dann zum Einsatz, wenn es um extrem komplexe Berechnungen geht, beispielsweise in der Automobilindustrie, wo Crash-Simulationen einen immensen Rechenaufwand erfordern.

Hochverfügbar

In diesem Artikel geht es um den Aufbau eines High-Availability-Clusters (HA). Wie beschrieben kümmert dieser sich um die Ausfallsicherheit von Diensten, etwa eines Web- oder Mailservers. Was vielen gar nicht bekannt ist: Auch virtuelle Computer-Instanzen zählen zu den Clusterdiensten. Da mittlerweile die Virtualisierung von Computersystemen auch im x86-Umfeld einen sehr ausgereiften Stand erreicht hat, kommt diese auch hier zum Einsatz. Statt einzelne Dienste hochverfügbar anzubieten, kümmert sich der Clustermanager um die Bereitstellung und Überwachung von kompletten virtuelle Maschinen-Instanzen.

Das Ganze sieht dann so aus, dass die physischen Hostsysteme über eine bestimmte Anzahl von virtuellen Maschinen verfügen. Diese liegen in einem von allen Hostsystemen zugänglichen, gemeinsam genutzen Speicherbereich. Das kann beispielsweise ein I-SCSI- oder Fiber-Channel-Storage sein. Zu Testzwecken ist auch eine NFS-Freigabe möglich, wobei aber die Performance leidet.

Der Clustermanager prüft in regelmäßigen Abständen, ob die einzelnen Hostsysteme noch verfügbar sind. Hierzu greift er auf unterschiedliche Methoden zurück. So ist jeder Clusterknoten dazu angehalten, in regelmäßigen Abständen so genannte Heartbeat-Meldungen über das Netzwerk zu versenden. Alternativ existieren auch Implementierungen, in denen die Systeme Statusmeldungen auf einem Datenträger hinterlegen oder so genannte Heuristiken ausführen müssen, beispielsweise das Anpingen eines zentralen Routers.

Bleiben die Lebenszeichen der einzelnen Clusterknoten aus, kümmert sich der Clustermanager um entsprechende Maßnahmen – beispielsweise den Neustart eines ausgefallenen Systems. Dieser Vorgang ist auch unter dem Namen Fencing, früher als STONITH (Shoot The Other Node In The Head) bekannt.

Dies ist notwendig, damit ein Clusterknoten nach einer Unterbrechung nicht fälschlicherweise wieder auf seine Ressourcen zugreifen kann, nachdem ein anderer Knoten diese bereits übernommen hat. Klassisches Beispiel ist ein Datenspeicher, der als Cluster-Ressource definiert und einem Knoten exklusiv zugeteilt ist. Fällt dieser Knoten aus, übernimmt ein anderer den Datenspeicher und führt I/O-Operationen auf ihm aus.

Erwacht der vermeintlich defekte Knoten wieder, glaubt er immer noch exklusiven Zugriff auf den Datenspeicher zu haben. Es käme also dazu, dass zwei Knoten gleichzeitig auf dieselben Daten zugreifen – eine Beschädigung der Daten wäre die Folge. Um dies zu verhindern, führen Fencing-Methoden einen Neustart der Maschine durch. Üblicherweise geschieht dies über fernschaltbare Stromsteckdosen oder auch über integrierte Management-Boards, beispielsweise HP-ILO-Karten oder IBM-Managementsysteme.

Ein Clustermanager kann virtuelle Systeme auch online migrieren. Das ist besonders dann hilfreich, wenn der Admin an den bestehenden Servern Wartungsarbeiten durchführen möchte. In einem solchen Fall wandern alle virtuellen Systeme eines Clusterknotens auf ein anderes System im Clusterverbund. Der Benutzer bekommt von der Migration der virtuellen Systeme nichts mit und kann ganz normal weiterarbeiten.

Hypervisor-Wahl

Um virtuelle Maschinen als Clusterservice zu betreiben, sind einige Vorarbeiten notwendig. Zunächst muss der Admin den passenden Hypervisor aussuchen, den Vermittler zwischen virtuellen Maschinen und der tatsächlich vorhandenen Hardware. Neben diversen kleineren Lösungen bietet Linux mit Xen (Abbildung 1, [1]) und KVM (Abbildung 2, [2]) zwei quelloffene Lösungen, die sich in der Vergangenheit bewährt haben.

Xen bietet den Vorteil, dass bei seinem Einsatz nicht zwingend besondere Hardware notwendig ist, um virtuelle Maschi-

nen-Instanzen unter Linux zu betreiben. Der Grund hierfür liegt in den modifizierten Treibern, die im Xen-Kernel zum Einsatz kommen. Diese sorgen dafür, dass die virtuellen Maschinen nicht direkt auf die Hardware zugreifen, sondern mit dem Hypervisior kommunizieren und dieser sich dann um die Vermittlung der diversen Anfragen an die tatsächliche Hardware kümmert.

Würden die virtuellen Maschinen direkt auf die Hardware zugreifen, könnte das zu einem Fehler führen, da diese in einer virtualisierten Umgebung, also in einem nicht privilegierten Modus ablaufen. Aus diesem Grund ist kein direkten Zugriff auf die Hardware möglich.

Privileg für Hypervisor

Der Hypervisor selbst läuft im privilegierten Modus. Anders sieht es aus, wenn nicht-quelloffene Systeme unter Xen zu virtualisieren sind. Da hier keine Modifikation der Treiber möglich ist, ist der Einsatz spezieller CPU-Architekturen (VT-X/AMD-V) nicht zu vermeiden. Die bei Xen eingesetzte Technologie ist allerdings nicht mehr State of the Art, was sich unter anderem auch dadurch bestätigt, dass der Code für den Hypervisor es trotz seiner langen Anlaufzeit nicht in den offiziellen Linux-Kernel geschafft hat. Distributoren, die Xen in ihrem Portfolio haben, müssen also mindestens immer zwei unterschiedliche Kernelvarianten herausbringen.

KVM ist ein Hypervisor der dritten Generation und mittlerweile der De-facto-Standard bei quelloffenen Virtualisierungslösungen im Linux-Umfeld. Dass KVM sehr schnell die Aufnahme in den offiziellen Linux-Kernel geschafft hat, liegt nicht nur an dessen ausgezeichneter Codequalität, sondern auch daran, dass KVM alle aktuellen Virtualisierungstechnologien, beispielsweise IOMMU (sicheres PCI-Pass-Through), KSM (Shared Memory Management) und Virt IO (paravirtualisierte IO-Treiber) unterstützt.

So ausgestattet laufen virtuelle Betriebssystem-Instanzen mit fast nativer Performance. Dafür benötigt KVM jedoch aktuelle CPU-Generationen mit Virtualisierungs-Funktion (VT-X/AMD-V). Die virtuellen Maschinen in diesem Artikel laufen alle unter dem KVM-Hypervisor. Aktuelle Enterprise-Lösungen zur Virtualisierung von Servern und Desktops [3] basieren ebenfalls auf KVM.

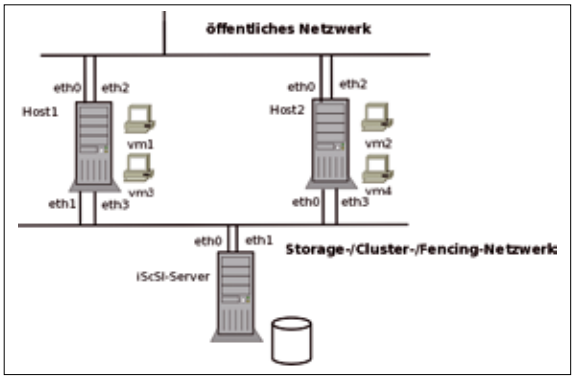

Abbildung 3: Jeder KVM-Host verwaltet eine gewisse Anzahl von virtuellen Systemen. Deren Überwachung ist Aufgabe des Clustermanagers.

Setup und Software

Das hier vorgestellte Beispiel arbeitet mit den beiden physischen Knoten »host1« und »host2«. Auf beiden kommt als Betriebssystem Red Hat Enterprise Linux 5 (RHEL5) zum Einsatz, jedoch funktionieren alle Beispiele auch auf aktuellen Fedora-Systemen. Auf beiden Hosts ist jeweils eine virtuelle KVM-System-Instanz (»vm1«, »vm2«) installiert. Diese Systeme laufen unter Fedora, der Aufbau funktioniert aber auch mit jedem anderen Betriebssystem, das der KVM-Hypervisor unterstützt [4].

Das Testlabor verzichtete auf die Installation weiterer virtueller Maschinen, um das Setup nicht unnötig komplex zu gestalten. Je nach Hardware-Ausstattung kann ein physischer Host natürlich eine Vielzahl virtueller Instanzen aufnehmen. Als Backend für die virtuellen Maschinen kommen Cluster-fähige logische Laufwerke (CLVM) zum Einsatz – diese sind auf den Clusterhosts statisch einzubinden. Ein I-SCSI-Server (Internet Small Computer System Interface) dient als Storage-Backend. Dies ist eine kostengünstige Alternative zu herkömmlichen SAN-Lösungen mit Fibre Channel, die bei heutigen Netzwerkbandbreiten hohe Performance gewährleistet.

Als Clustermanager kommt die Red-Hat-eigene Cluster Suite (RHCS) zum Einsatz. Sie enthält alle notwendigen Komponenten zum Betrieb eines HA-Clusters. Neben dem auf Open AIS (Application Interface Specification) basierenden Clustermanager »cman« kümmert sich das Cluster Configuration System »ccs« um eine einheitliche Konfiguration der einzelnen Knoten. Fencing-Agenten sind für ausgefallene Knoten zuständig, der Ressource-Manager »rgmanager« für die eigentlichen HA-Services, in diesem Fall die virtuellen Maschinen-Instanzen. Zum Einbinden eines gemeinsam genutzen Speichers kommt der bereits angesprochene Cluster Logical Volume Manager (CLVM) zum Einsatz.

Architektur

Die beiden physischen Clusterknoten in diesem Artikel-Beispiel verfügen über jeweils vier Netzwerkkarten »eth0« bis »eth3«. Die erste (»eth0«) und dritte (»eth2«) Netzwerkkarte sind dabei zu einem virtuellen Bond-Device »bond0« zusammengeschaltet. Dieses Device ist über eine Bridge »br0« mit einem öffentlichen Netzwerk verbunden. Über dieses Netzwerk ist beispielsweise der Zugriff auf einzelne Applikationen innerhalb der virtuellen Maschinen möglich.

Ein zweites Bond-Device »bond1« besteht aus der zweiten (»eth1«) und vierten (»eth3«) Netzwerkkarte. Dieses Bond-Device gehört zum privaten Clusternetzwerk. Über dieses Netz läuft die komplette Cluster-Kommunikation der einzelnen Knoten. Hierzu zählen auch die Kommunikation der einzelnen Fencing-Agents sowie der Netzwerkverkehr zum

Listing 1: Locking-Level 3 bei CLVM

```
01 # lvmconf --enable-cluster
02 # grep locking_type /etc/lvm/lvm.conf
03     locking_type = 3
```

Listing 2: I-SCSI-Server einrichten

```
01 # chkconfig tgtd on; service tgtd start
02 # tgtadm --lld iscsi --op new --mode target --tid 1
   -T iqn.2008-01.com.example:storage.iscsi.disk1
03 # tgtadm --lld iscsi --op new --mode logicalunit
   --tid 1 --lun 1 -b /dev/sda
04 # tgtadm --lld iscsi --op bind --mode target --tid 1
   -I ALL
```

I-SCSI-Server. Das Zusammenschalten der einzelnen Netzwerkkarten zu einem virtuellen Bond-Device ist deshalb notwendig, damit nicht die Netzwerkkarten den Single Point of Failure (SPoF) darstellen. Was bringt es sonst, sämtliche Systeme hochverfügbar zu gestalten, wenn die Systeme beim Ausfall einer Netzwerkkarte nicht mehr zu erreichen sind.

Unter Linux gibt es dafür mit dem Kernel-Bonding-Treiber eine bewährte Lösung. Hierbei bilden mehrere Netzwerkkarten ein neues, virtuelles Device. Je nach Bonding-Modus fließen die Daten im Round-Robin-Verfahren oder nach dem Master-Slave-Prinzip über die einzelnen Karten. Für Zugriffe auf den I-SCSI-Server steht alternativ die Multipathing-Funktion des Linux-Kernel-Device-Mappers bereit. Ähnlich wie beim Bonding führen auch hier mehrere Pfade zum Ziel. Fällt ein Pfad aus, ist der Zugriff auf die Daten immer noch über die verbleibenden Pfade möglich.

Als Backend-Speicher für die virtuelle Maschinen-Instanz kommen entweder Imagedateien oder Blockdevices auf einem gemeinsam genutzen Datenspeicher in Frage. Blockdevices in Form von logischen Laufwerken, wie sie der Cluster Logical Volume Manager (CLVM) zur Verfügung stellt, sind hier die erste Wahl. Im Vergleich zu Imagedateien oder Raw-Devices haben logische Laufwerke den Vorteil, dass sie ohne den zusätzlichen Overhead eines Cluster-Dateisystems auskommen. Außerdem steht beim Einsatz von CLVM auch dessen Snapshot-Funktion zur Verfügung. Dies ist vor allem zum Sichern virtueller Maschinen interessant.

Die Konfiguration des CLVM ist nahezu identisch mit der von Einzelsystemen. Ein Unterschied besteht in der Art und Weise, wie der CLVM mit Lock-Informationen für Dateizugriffe umgeht. Die Synchronisation dieser Informationen zwischen den einzelnen Clusterknoten ist Aufgabe des CLVM. Somit ist jedes System über entsprechende Datenspeicher-Zugriffe der anderen Systeme informiert. Zur Konfiguration ist auf allen Knoten des Clusters das Paket »lvm2-cluster« zu installieren. Über den Aufruf »lvmconf« aktiviert der Admin dann die entsprechende Locking-Bibliothek (Listing 1). Beim Einsatz des CLVM ist das Cluster-weite Locking (Type 3) zu aktivieren. Die Standardeinstellung des Logical Volume Manager (LVM) ist das lokale File-Locking (Type 1).

Konfiguration

Bevor es an die Konfiguration der virtuellen Systeme geht, ist im ersten Schritt das I-SCSI-System einzurichten. Bei I-SCSI handelt es sich um ein in RFC 3720 definiertes Verfahren, das herkömmliche SCSI-Befehle über ein TCP/IP-Netzwerk überträgt. Der klassische SCSI-Controller heißt hier Initiator, das Speichermedium Target. Der Vorteil von I-SCSI ist, dass sich große Speichernetzwerke mit einer bereits vorhanden Infrastruktur betreiben lassen.

Als Netzwerkkarten bieten sich eventuell TCP/IP-Offload-Karten an, um die CPUs der eingesetzten Rechner zu schonen. Alternativ gibt es auch dedizierte I-SCSI-Hardware, die jedoch nicht gerade billig ist, sodass für erste Tests durchaus vorhandene Hardware zum Einsatz kommen kann. Für das Server-Setup gibt es in Linux das Paket »scsi-target-utils«, das üblicherweise in den Repositories der meisten Distributoren enthalten ist. Nach dessen Installation muss der Admin auf dem I-SCSI-Server ein entsprechendes Blockdevice erzeugen, das dann über das I-SCSI-Protokoll den Clients zur Verfügung steht. Bei dem Device sollte es sich im besten Fall

SICHERHEIT IST KEINE FRAGE DES VERTRAUENS!

Die Sicherheit der Hochverfügbarkeit für Linux-Systeme ist vielmehr eine Frage der Kompetenz. Vor allem dann, wenn man mit der führenden Open-Source-Blockreplikationstechnologie DRBD kompromisslos auf Nummer sicher gehen will. Fragt sich, wer außer den Erfindern von DRBD auch im Support bestmögliche Unterstützung bieten kann? Wohl niemand.

CLUSTER-SUPPORT VON LINBIT BEDEUTET KOMPETENZTRANSFER.
Ein Supportvertrag mit LINBIT, den Entwicklern von DRBD und weltweit führenden Spezialisten für Hochverfügbarkeit, schenkt Ihnen und Ihrem Unternehmen die absolute Gewissheit, bestmöglich für Ihr Business vorgesorgt zu haben. Holen Sie sich Know-how, aus dem LINBIT kein Geheimnis macht, aus erster Hand ins Haus. 24x7-Verfügbarkeit und vieles mehr.
Beruhigend zu wissen:
Ab € 1.950,- sind Sie sicher dabei!

LINBIT
YOUR WAY TO HIGH AVAILABILITY

T: +43 (0)1/817 82 92-0
info@linbit.com | www.linbit.com

um ein redundantes Hardware-Raid handeln, da ansonsten die exportierte Festplatte wieder einen Single Point of Failure darstellt. Mit wenigen Kommandos ist dann der I-SCSI-Server wie in Listing 2 zu konfigurieren.

Der erste Befehl startet den Target-Daemon und aktiviert ihn für den nächsten Reboot. Zeile 2 erzeugt das I-SCSI-Target, das durch eine eindeutige ID und einen Namen gekennzeichnet ist. Clients nutzen diese Identifikationsmerkmale, um später Zugriff auf das Target zu erhalten. Im nächsten Schritt ordnet der Admin diesem Target ein oder mehrere lokale Blockdevices des Servers zu. Abschließend ist es möglich, den Zugriff auf den Server mit Hilfe von Access-Control-Listen (ACL) einzuschränken. Im Beispiel bekommt jedoch jeder Rechner Zugriff auf den Server. Hat alles funktioniert, sollte der Aufruf von »tgtadm« die fertige Konfiguration anzeigen (Listing 3).

Listing 3: I-SCSI-Server läuft

```
01 # tgtadm --lld iscsi --op show --mode target
02 Target 1: iqn.2008-01.com.example:storage.iscsi.disk1
03         System information:
04         Driver: iscsi
05         Status: running
06     I_T nexus information:
07     LUN information:
08         LUN: 0
09             Type: controller
10             SCSI ID: deadbeaf1:0
11             SCSI SN: beaf10
12             Size: 0
13             Backing store: No backing store
14         LUN: 1
15             Type: disk
16             SCSI ID: deadbeaf1:1
17             SCSI SN: beaf11
18             Size: 500G
19             Backing store: /dev/sda
20     Account information:
21     ACL information:
22         ALL
```

Listing 4: »ifcfg-eth0«

```
01 DEVICE=eth0
02 ONBOOT=yes
03 USERCTL=no
04 SLAVE=yes
05 MASTER=bond0
```

Listing 5: »ifcfg-bond0«

```
01 DEVICE=bond0
02 IPADDR=192.168.10.200
03 NETMASK=255.255.255.0
04 ONBOOT=yes
05 USERCTL=no
06 BONDING_OPTS="mode=1 miimon=100"
```

Netzwerk-Bonding

Damit nicht die Netzwerkkarten und somit der Anschluss an das Cluster- und Storage-Netzwerk einen Single Point of Failure bilden, sind auch diese redundant auszulegen. Idealerweise ist dabei jede Netzwerkkarte mit einem anderen Switch verbunden, um auch hier Ausfälle unbeschadet zu überstehen. Unter Linux dient für das Zusammenlegen von Netzwerkkarten üblicherweise das Bonding-Modul des Kernels.

Im Setup kommt der Master-Slave-Modus zum Einsatz, da hierfür keine besondere Konfiguration der Netzwerk-Switches notwendig ist und dieser den Ausfall einer Karte verschmerzen kann. Das Listing 4 zeigt beispielhaft die Konfiguration für die erste Netzwerkkarte in der Datei »/etc/sysconfig/network-scripts/ifcfg-eth0«. Alle anderen Karten des gleichen Bond sind entsprechend konfiguriert. Listing 5 zeigt die Konfiguration »/etc/sysconfig/network-scripts/ifcfg-bond0« für das Bond-Device. Über die Anweisung »BONDING_OPTS« ist der gewünschte Mode einzustellen.

Eine detaillierte Beschreibung mit allen Konfigurationsmöglichkeiten zum Bonding-Treiber steht im Kernel-Dokumentationsordner [5]. Im laufenden Betrieb liefert die Datei »/proc/net/bonding/bond0« hilfreiche Informationen zum Status des Device und seiner Slaves.

Bonding virtuell

Was für den I-SCSI-Server gilt, das gilt genauso für die Clusterknoten selbst. Um einen SPoF im Netzwerk zu vermeiden, sind sowohl die Netzwerkkarten für das öffentliche Netzwerk (»eth0« und »eth2«) als auch die Karten für das Cluster- und Storage-Netzwerk (»eth1«, »eth3«) zu einem virtuellen Bond-Device zusammenzulegen. Die Konfiguration erfolgt entsprechend der Bonding-Konfiguration des I-SCSI-Server.

Damit nun auch auf den Clustersystemen das exportierte I-SCSI-Device zur Verfügung steht, ist im nächsten Schritt das Paket »iscsi-initiator-utils« zu installieren. Auch dieses findet sich in den Standard-Software-Repositories der meisten Distributoren. Die folgenden Kommandos erlauben dann den Zugriff auf das exportierte I-SCSI-Target (Listing 6).

Hier zeigt ein I-SCSI-Discovery alle Targets auf dem angegebenen Server an, bevor im nächsten Schritt das Login auf dem Server erfolgt. Danach steht das exportierte Device lokal zur Verfügung. Der Aufruf von »fdisk -l« sollte nun neben dem lokalen Device auch das I-SCSI-Device auflisten. Aus diesem Device setzt sich nachher die Volume Group des Linux Device-Mappers zusammen.

Wie beim LVM üblich erfolgt deren Konfiguration über folgende Kommandos:

```
# pvcreate /dev/sdb1
# vgcreate vg_cluster /dev/sdb1
# lvcreate -n lv_vm1 -L 50G vg_cluster
# lvcreate -n lv_vm2 -L 50G vg_cluster
```

Der erste Befehl hinterlegt die notwendige LVM-Signatur auf der ersten Partition des eingebundenen I-SCSI-Device. Im nächsten Schritt erzeugt der Aufruf von »vgcreate« die Volume-Gruppe »vg_cluster«, die aus einem einzelnen Device besteht. Abschließend erzeugt »lvcreate« die logischen Laufwerke, die später als Backend für die virtuellen Maschinen-Instanzen dienen. Damit die so erzeugten logischen Laufwerke allen Clusterknoten zur Verfügung stehen, ist der »clvmd«-Dienst auf allen Clusterknoten zu starten. Dies ist jedoch erst nach der Einrichtung des Clusters möglich.

Cluster-Setup

Um nun den HA-Cluster einzurichten, gibt es mehrere Möglichkeiten. Mit »system-config-cluster« (Abbildung 4) und Conga gibt es zwei grafische Anwendungen, jedoch ist es auch keine Problem, die XML-basierte Konfigurationsdatei manuell zu erzeugen und den Cluster auf diese Weise zu konfigurieren. Viele Optionen stehen über die grafischen Tools auch gar nicht zur Verfügung, sodass ein manuelles Bearbeiten der Datei sowieso in den meisten Fällen notwendig ist. Listing 7 zeigt eine bespielhafte Cluster-Konfigurationsdatei.

Die Konfigurationsdatei besteht aus mehreren Abschnitten. Zuerst sind die ein-

zelnen Clusterknoten aufgeführt. Hier ist darauf zu achten, dass die verwendeten Rechnernamen über das interne Clusternetzwerk auflösbar sind, sonst findet die ganze Cluster-Kommunikation über das öffentliche Netzwerk statt. Für jeden Knoten ist mindestens eine Fencing-Methode anzugeben. Da im Beispiel ein Blade Center zum Einsatz kommt, liegt es nahe, das eigene Blade-Managementsystem als primären Notaus-Mechanismus zu verwenden.

Als Backup-Methode ist manuelles Fencing konfiguriert. Der Clustermanager basiert auf dem Open-AIS-Framework. Für dessen Heartbeat-Meldungen wandert ein Token zwischen den einzelnen Clusterknoten hin und her. Für dessen Übertragung kommt Multicast zum Einsatz. Sollten die verwendeten Netzwerk-Switches Probleme mit der Default-Multicast-Adresse haben, so lässt sich diese beliebig ändern.

Als Nächstes sind die eigentlichen Fence-Devices näher zu beschreiben, hier beispielsweise die IP-Adressen und Logindaten für das Blade-Managementsystem. Eine Liste der unterstützten Fence-Devices findet sich unter [6]. Soll ein virtuelles System primär auf einem bestimmten Clusterknoten laufen, dann lässt sich dies über so genannte Failover-Domains realisieren. Dieser Mechanismus ist gerade beim Einsatz von mehreren Systemen sehr nützlich, möchte man verhindern, dass auf einmal alle virtuellen Maschinen auf dem gleichen System laufen.

Nachhilfe für KVM

Im letzten Abschnitt folgt schließlich der Ressourcen-Manager. Jede virtuelle Maschinen-Instanz ist hier mit Namen aufzuführen. Kommt als Hypervisor Xen zum Einsatz, lässt sich in diesem Abschnitt auch der Speicherort der Konfigurationsdateien mit angeben. Diese können bei Xen nämlich auf einem von allen Knoten gemeinsam genutzen Datenspeicher liegen.

Für KVM existiert diese Möglichkeit noch nicht. Hier muss der Admin die Konfigurationsdateien der virtuellen Maschinen im lokalen Dateisystem ablegen und zwischen den einzelnen Clusterknoten synchronisieren. In einer der nächsten Versionen der Clustersuite wird allerdings auch hier die Speicherung der Konfigurationsdateien auf Shared Storage möglich sein. Da die virtuellen Maschinen zu diesem Zeitpunkt noch nicht existieren, ist dieser Abschnitt in der aufgeführten Konfigurationsdatei noch auskommentiert.

Config verteilen

Wer die Konfigurationsdatei den eigenen Gegebenheiten angepasst hat, kann sie auf alle Clusterknoten verteilen, beispielsweise mit »scp«. Im laufenden Clusterbetrieb lässt sich diese Aufgabe später auch durch »ccs_tool update /etc/cluster/cluster.conf« erledigen. Wichtig dabei ist, dass nach jeder Änderung die Versionsnummer der Konfigurationsdatei erhöht wird, sonst bekommt der Clustermanager von einer Änderung an der Datei nichts mit.

Liegt nun allen Clusterknoten die Datei vor, geht der Cluster mit »service cman start« in Betrieb. Auch der Cluster-fähige Volume Manager (CLVM) ist nun funkti-

Listing 6: »ifcfg-bond0«

```
01 # chkconfig iscsi on; service iscsi start
02 # iscsiadm -m discovery -t st -p 192.168.10.200
03 iqn.2008-01.com.example:storage.iscsi.disk1
04 # iscsiadm -m node -T iqn.2008-01.com.
   example:storage.iscsi.disk1 -p 192.168.10.200 -l
```

Listing 7: »/etc/cluster/cluster.conf«

```
01 <?xml version="1.0"?>
02 <cluster alias="kvmcluster" config_version="1"
   name="examplecluster">
03     <fence_daemon clean_start="0" post_fail_
   delay="0" post_join_delay="30"/>
04     <clusternodes>
05         <clusternode name="host1-cluster.
   example.com" nodeid="1" votes="1">
06             <fence>
07                 <method name="1">
08                         <device name="balde-mgmt.
   example.com" blade="1"/>
09                 </method>
10                 <method name="2">
11                         <device name="manual"
   nodename="host1.example.com"/>
12                 </method>
13             </fence>
14         </clusternode>
15         <clusternode name="host2-cluster.
   example.com" nodeid="2" votes="1">
16             <fence>
17                 <method name="1">
18                         <device name="blade-mgmt.
   example.com" blade="2"/>
19                 </method>
20                 <method name="2">
21                         <device name="manual"
   nodename="host2.example.com"/>
22                 </method>
23             </fence>
24         </clusternode>
25     </clusternodes>
26     <cman
27         <multicast addr="225.0.0.250"/>
28         <expected_votes="1" two_node="1"/>
29     </cman>
30     <fencedevices>
31         <fencedevice agent="fence_bladecenter"
   ipaddr="192.168.10.100" login="fence"
   name="blade-mgmt" passwd="redhat"/>
32         <fencedevice agent="fence_manual"
   name="manual"/>
33     </fencedevices>
34
35 <!--
36     <rm>
37         <failoverdomains>
38             <failoverdomain name="kvm-node1"
   ordered="0" restricted="0">
39                     <failoverdomainnode
   name="host1-cluster.example.com" priority="1"/>
40             </failoverdomain>
41             <failoverdomain name="kvm-node2"
   ordered="0" restricted="0">
42                     <failoverdomainnode
   name="host2-cluster.example.com" priority="1"/>
43             </failoverdomain>
44         </failoverdomains>
45         <resources/>
46         <vm name="vm1" domain="kvm-node1"/>
47         <vm name="vm2" domain="kvm-node2"/>
48         <vm name="vm3" domain="kvm-node1"/>
49         <vm name="vm4" domain="kvm-node2"/>
50     </rm>
51 -->
52 </cluster>
```

Listing 8: »virt-install«

```
01 # virt-install \
02   --network bridge:br0 \
03   --name vm1 \
04   --ram 1024 \
05   --disk path=/dev/vg_cluster/lv_vm1 \
06   --location http://192.168.0.254/pub/os/fc12/
                                          \
07   --nographics \
08   --os-type linux \
09   --os-variant fedora12 \
10   --extra-args 'console=xvc0 ksdevice=br0 \
11   ks=http://192.168.0.254/pub/ks/vm1.cfg'
```

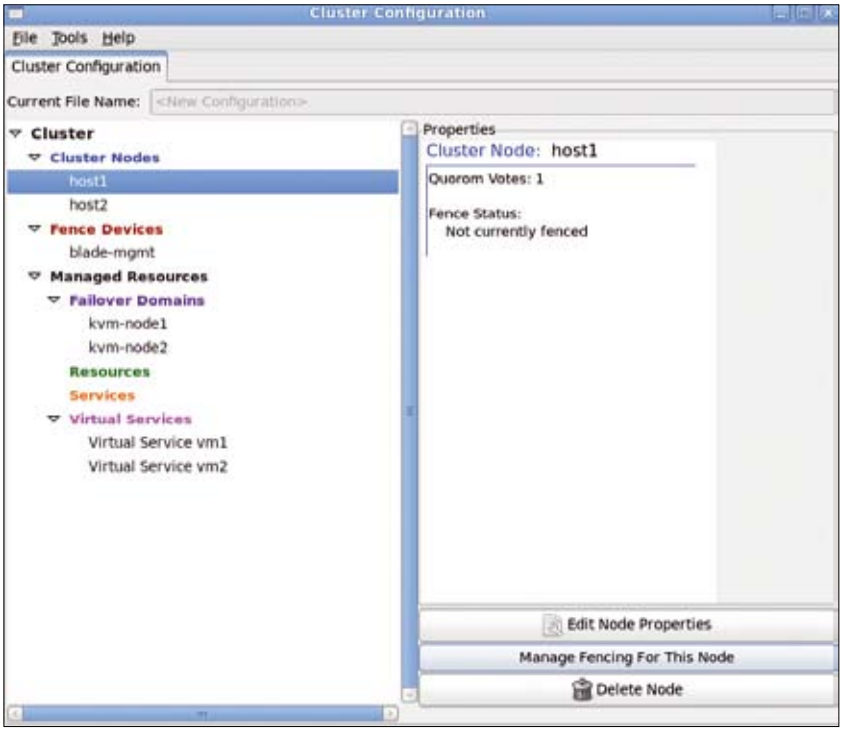

Abbildung 4: »system-config-cluster« ist eine grafische Anwendung zur Konfiguration des HA-Clusters.

onsfähig und nimmt mit »service clvmd start« seinen Betrieb auf. Der Aufruf »clustat« sollte bestätigen, dass beide Knoten aktive Clustersysteme sind.

Bevor es nun an die Konfiguration der virtuellen Maschinen geht, fehlt noch die passende Netzwerkumgebung. Da sich die VMs mit den Clustersystemen in einem Netzwerk befinden sollen, ist das mit einer Bridge zu bewerkstelligen. Diese lässt sich wie in Kasten 1 gezeigt konfigurieren.

Um schließlich die eigentlichen virtuellen Maschinen-Instanzen zu erzeugen, stehen wieder mehrere Wege offen. So existiert mit »virt-manager« (Abbildung 5) eine grafische Anwendung, alternativ bietet das Kommandozeilen-Tool »virt-install« eine Vielzahl von Optionen, wie der Aufruf zum Setup der ersten virtuellen Maschine verdeutlicht (Listing 8).

Die Angabe einer Kickstart-Datei ist optional, es lassen sich auch sämtliche Konfigurationsanweisungen für das zu installierende System manuell angeben. Wichtig ist die Angabe der zuvor erzeugten Bridge »br0« und des logischen Laufwerks »lv_vm1« auf dem I-SCSI-Server, das nun als Storage-Backend für die virtuelle Maschine dient.

Die XML-basierten Konfigurationsdateien für die so erzeugten virtuellen Maschinen liegen auf dem lokalen Dateisystem im Verzeichnis »/etc/libvirt/qemu«. Dieses ist zwischen allen Clusterknoten, hier also zwischen Host1 und Host2, synchron zu halten.

Nun ist es Zeit, auch den letzten Abschnitt der Cluster-Konfigurationsdatei zu aktivieren. Hierfür entfernt der Admin einfach die Kommentarzeichen und überträgt die neue Konfigurationsdatei mittels »ccs_tool update« auf alle anderen Clusterknoten. Natürlich nicht ohne die Versionsnummer der Datei hochzuzählen. Mittels »service rgmanager« nimmt dann dieser Dienst seinen Betrieb auf.

SSH ohne Passphrase

Da der Cluster-Ressource-Manager zur Migration der virtuellen Maschinen auf

Kasten 1: Bridge für das öffentliche Netzwerk

Um auch im öffentlichen Netzwerk keinen SPoF beim Setup der Netzwerkkarten zu haben, kommt ein Bond-Device »bond1« zum Einsatz. Es setzt sich jeweils aus beiden Karten »eth0« und »eth2« zusammen. Die Bridge selbst muss nicht unbedingt eine IP-Adresse erhalten, sollen aber die Clusterknoten aus dem öffentlichen Netzwerk erreichbar sein, ist die Angabe einer IP-Adresse in der Bridge-Konfiguration möglich. Dies nutzt das Beispiel, da die Installationsroutine der virtuellen Maschinen auf einen Rechner im öffentlichen Netzwerk zugreift, der ein Fedora-Installationsrepository sowie Kickstart-Dateien zur automatischen Installation bereitstellt. Die Netzwerk-Init-Skripte sind in wenigen Schritten erzeugt:

/etc/sysconfig/network-scripts/ifcfg-eth0:

```
DEVICE=eth0
HWADDR=00:16:76:D6:C9:45
ONBOOT=yes
SLAVE=yes
USERCTL=no
Master=bond1
```

/etc/sysconfig/network-scripts/ifcfg-eth1:

```
DEVICE=eth0
HWADDR=00:16:76:D6:C9:45
ONBOOT=yes
SLAVE=yes
USERCTL=no
MASTER=bond1
```

/etc/sysconfig/network-scripts/ifcfg-bond1:

```
DEVICE=bond1
ONBOOT=yes
USERCTL=no
BRIDGE=br0
BONDING_OPTS="mode=1 miimon=100"
```

/etc/sysconfig/network-scripts/ifcfg-br0:

```
DEVICE=br0
TYPE=Bridge
IPADDR=192.168.0.100
NETMASK=255.255.255.0
ONBOOT=yes
DELAY=0
```

Der abschließende Iptables-Befehl leitet den ganzen Netzwerk-Traffic über die Bridge an die virtuellen Maschinen:

```
# iptables -I FORWARD -m physdev --physdev-is-bridged -j ACCEPT
```

Schließlich liest ein »service network restart« die Konfiguration ein und startet das Bridge-Device, wie die Ausgabe von »brctl« beweist:

```
# brctl show
bridge name     bridge id           STP     enabled interfaces
br0             8000.000e0cb30440   no      br0
```

Abbildung 5: Mit »virt-manager« steht eine grafische Anwendung zur komfortablen Konfiguration von virtuellen Systemen zur Verfügung.

SSH zurückgreift, ist für jeden Clusterknoten ein SSH-Schlüsselbund zu erzeugen. Dabei ist darauf zu achten, beim Anlegen des Schlüssels keine Passphrase zu verwenden. Den Schlüssel erzeugt dann einfach »ssh-keygen« (Listing 9).

Den öffentlichen Teil, also die Datei »/root/.ssh/id_rsa_cluster.pub«, verteilt der Admin auf alle Knoten im Cluster. In der SSH-Client-Config »/root/.ssh/config« muss er noch für alle Clusterknoten die Option »StrictHostKeyChecking« deaktivieren (Listing 10).

Alle hier aufgeführten Schritte, also das Anpassen der Config und das Erzeugen der SSH-Schlüssel sowie deren Verteilung, müssen auf allen Clusterknoten erfolgen. Bei einem abschließenden Test sollte das Root-Login via SSH von jedem Clusterknoten zu jedem anderen Knoten ohne Angabe des Passworts möglich sein. Ist dies nicht der Fall, steht Fehlersuche an, sonst wird der Cluster nicht fehlerfrei funktionieren.

Ressourcen-Manager

Sind diese ganzen Vorarbeiten abgeschlossen, sind nur noch die einzelnen virtuellen Maschinen über den Clustermanager zu aktivieren. Das Tool »clusvcadm« erweckt sie zum Leben:

```
# clusvcadm -e vm:vm1 -m host1.example.com
# clusvcadm -e vm:vm2 -m host2.example.com
```

Die Ausgabe von »clustat« sollte nun zusätzlich zu den physischen auch die beiden virtuellen Systeme anzeigen (Zeilen 11 und 12, Listing 11).

Fällt ein System aus, dann starten alle darauf laufenden virtuellen Systeme auf dem anderen Host. Außerdem ist auch eine Online-Migration möglich. Hierfür ist »clusvcadm« wie folgt aufzurufen:

```
# clusvcadm -M vm:vm1 -m
host2.example.com
```

Zu Demonstrationszwecken eignet sich folgendes Szenario: Auf der »vm1« läuft die Kompilierung des Linux-Kernels. Während des Kompiliervorgangs stößt der oben aufgeführte Befehl die Migration der virtuellen Maschine auf den anderen Host an. Für den Benutzer ist die Migration komplett transparent.

Natürlich lässt sich das bestehende Setup auch noch um einen zweiten Cluster innerhalb der virtuellen Maschinen ergänzen. Dieser könnte sich dann beispielsweise um einzelne Netzwerkdienste innerhalb der virtuellen Maschine selbst kümmern. Das Setup ist nahezu identisch, lediglich bei der Konfiguration der (virtuellen) Fence-Devices ist darauf zu achten, den speziellen Agent »fence_virt« zu verwenden. Dieser versorgt den »fence_virtd« mit entsprechenden Anweisungen, sollte eine virtuelle Maschine nicht mehr reagieren.

Fazit

Ein Cluster-Setup mit virtuellen Maschinen als Cluster-Ressourcen ist schnell aufgebaut. Im Vergleich zur nativen Live-Migration auf Basis von Libvirt kümmert sich der Clustermanager auch um die Migration von Maschinen, wenn das Hostsystem ausgefallen ist. Für den Benutzer ist eine solche Migration komplett transparent und erfolgt ohne jeden manuellen Eingriff.

Auch wenn in diesem Artikel und seinen Beispielen lediglich von Linux-basierten virtuellen Maschinen die Rede war, so ist doch auch der Einsatz anderer Betriebssysteme innerhalb eines virtuellen Systems möglich. *(ofr)* ■

Infos

[1] Xen-Hypervisor: [http://www.xen.org]

[2] KVM-Hypervisor: [http://www.linux-kvm.org]

[3] Red Hat Enterprise Virtualizsation (RHEV): [http://www.redhat.com/virtualization/rhev/server]

[4] Unterstützte KVM-Gastsysteme: [http://www.linux-kvm.org/page/Guest_Support_Status]

[5] Bonding-Treiber-Dokumentation im Linux-Kernel-Baum: [/usr/share/doc/kernel-doc/Documentation/networking/bonbding.txt]

[6] Unterstützte Fence-Devices der Red Hat Cluster Suite: [http://sources.redhat.com/cgi-bin/cvsweb.cgi/cluster/fence/agents/?cvsroot=cluster]

Der Autor

Thorsten Scherf arbeitet als Senior Consultant für Red Hat EMEA. Er ist oft als Vortragender auf Konferenzen anzutreffen. Wenn ihm noch Zeit bleibt, nimmt er gerne an Marathonläufen teil.

Listing 9: »ssh-keygen«

```
01 # ssh-keygen -f /root/.ssh/id_rsa_cluster
02 Generating public/private rsa key pair.
03 Created directory '/root/.ssh'.
04 Enter passphrase (empty for no passphrase):
05 Enter same passphrase again:
06 Your identification has been saved in /root/.ssh/
   id_rsa_cluster.
07 Your public key has been saved in /root/.ssh/id_rsa_
   cluster.pub.
08 The key fingerprint is:
09 bb:0f:e3:7f:3e:ab:c3:5d:32:78:85:04:a2:da:f1:61 root@
   host1
```

Listing 10: »/root/.ssh/config«

```
01 Host host*
02         StrictHostKeyChecking no
03         IdentityFile /root/.ssh/id_rsa_cluster
```

Listing 11: »clustat«

```
01 # clustat
02 Member Status: Quorate
03
04   Member N                ID   Status
05   ------ ----              ---- ------
06   host1.example.com       1 Online, Local, rgmanager
07   host2.example.com       2 Offline
08
09   Service Name      Owner (Last)            State
10   ------- ----      ----- ------            -----
11   vm:vm1            host1.example.com       started
12   vm:vm2            host2.example.com       started
```