IBM pSeries, AIX & Linux Technical University, München, 26.-29. Oktober 2004

VIRTUELLE TESTUMGEBUNGEN MIT USER MODE LINUX

Armin M. Warda, Postbank Systems AG,
Armin(dot)Warda(at)Postbank(dot)De

ZUSAMMENFASSUNG

Bei „User Mode Linux“ (UML) handelt es sich um einen Linux-Kernel, der als regulärer Prozess auf einem Linux- (oder Windows-) Host-System ausgeführt werden kann. UML wurde von Jeff Dike initiiert. Mittels UML können mehrere virtuelle Linux-Systeme („Gäste“) gleichzeitig auf demselben physischen System („Host“) ausgeführt werden. Jeder Gast kann dabei auf unterschiedlichen Linux-Distributionen und Versionen des Linux-Kernels basieren. Gäste benutzen virtuelle Festplatten, die auf dem Host als Dateien abgebildet werden, und sie können über virtuelle Ethernets und serielle Adapter untereinander und mit dem Host kommunizieren.

UML eignet sich sehr gut, um virtuelle Testumgebungen aufzubauen, wegen seiner ausgeprägten Möglichkeiten zur Fault-Injection insbesondere für funktionale Tests von Hochverfügbarkeitssystemen. (Für Last- und Performance-Tests eignet es sich wegen seiner eher mässigen Effizienz und Performance allerdings weniger.) Man kann komplexe Test-Szenarien bestehend aus mehreren Linux-Systemen und Ethernets bereits auf einem Notebook aufsetzen, sodass diese virtuellen Labore in einfacher Weise transportiert, präsentiert und zusammen mit anderen studiert werden können. Und schliesslich kann man einfach ein ganzes Labor auf eine CD/DVD brennen und so anderen verfügbar machen.

USER MODE LINUX

Bei „User Mode Linux“ (UML) handelt es sich um einen Linux-Kernel, der derart kompiliert ist, dass er als regulärer Prozess auf einem Linux- (oder Windows-) Host-System ausgeführt werden kann. Das UML-Projekt wurde von Jeff Dike inittiert und ursprünglich entwickelt, um das Debugging des Linux-Kernels zu vereinfachen, indem ein in Entwicklung befindlicher Linux-Kernel im User-Space eines anderen Linux-Systems ausgeführt wird, sodass bspw. der GNU Debugger gdb und andere bei der Software-Entwicklung von regulären Linux-Programmen übliche Tools eingesetzt werden können.

Mit UML können mehrere virtuelle Linux-Systeme („Gäste“) gleichzeitig auf demselben physischen Linux-System („Host“) ausgeführt werden. Jeder Gast kann dabei unterschiedliche Versionen des Linux-Kernels (z.B. 2.4.28, 2.6.9-bb4) ausführen und auf verschiedenen Linux-Distributionen (z.B. SuSE Professional 9.2, Debian 3.0, Redhat,..) basieren.

UML benötigt keine speziellen Anpassungen des Host-Systems und i.a. noch nicht einmal Root-Rechte, d.h. die Gast-Systeme können auf dem Host-System unter nicht-privilegierten User-Kennungen laufen. Lediglich wenn man dem Host-System die Kommunikation mit seinen Gast-Systemen ermöglichen möchte, sind Root-Rechte erforderlich, um geeignete virtuelle Ethernet-Adapter im Host-System zu konfigurieren. Gleiches gilt, wenn man vom Host-System aus per „Loopback-Mount“ (siehe unten) auf den Inhalt virtueller Festplatten zugreifen möchte.

VIRTUELLE FESTPLATTEN

UML-Gast-Systeme benutzen virtuelle Festplatten, die auf dem Host als Dateien abgebildet werden. Daher kann man UML-Gäste einfach durch Kopieren dieser Dateien clonen. Die Inhalte der virtuellen Festplatten können dem Linux-Host-System per „Loopback-Mount“ der entsprechenden Dateien zugänglich gemacht werden, was sehr praktisch ist, um bspw. eine Reparatur an der System-Platte eines UML-Gasts vorzunehmen, die eine logische Datenkorruption enthält, die einen Boot des Gasts von dieser Festplatte nicht mehr zulässt.

Neben normalen Dateien als virtuelle Festplatten unterstützt UML auch sogenannte COW-Disks. COW steht für Copy-On-Write. Eine COW-Disk besteht aus zwei Dateien. Die erste Datei ist Read-Only und wird

„Hintergrunddatei“ genannt, die zweite Datei ist Read/Write, zeichnet aber nur die Änderungen gegenüber der ersten Datei auf, sie kann man als „Delta-Datei“ bezeichnen. Eine COW-Disk wird erzeugt, indem man einfach die einer normalen virtuellen Festplatte entsprechende Datei Read-Only (und somit zur Hintergrunddatei) macht und eine leere zweite (Delta-) Datei hinzudefiniert. Ab jetzt ändert sich der Inhalt der Hintergrunddatei nicht mehr und alle Änderungen werden in der Delta-Datei aufgezeichnet.

Mit COW-Disks kann man viel physischen Festplattenplatz auf dem Host sparen, wenn man mehrere ähnliche Gäste betreibt (z.B. Cluster mit identischen Knoten), weil mehrere COW-Disks dieselbe Hintergrunddatei (die ja nur Read-Only ist) gemeinsam nutzen können. Jede COW-Disk muss aber eine dedizierte Delta-Datei haben. Dies spart deshalb Platz, weil die Delta-Dateien als Sparse-Files (spärlich besetzte Dateien, Dateien mit Lücken) realisiert sind, sprich nur in etwas soviel Platz belegen, wie Änderungen gegenüber der Hintergrunddatei aufgezeichnet sind. Beispielsweise könnten die vier virtuellen 2 GB Festplatten eines 4-Node-Clusters bei Realisierung als COW-Disk eine gemeinsame 2 GB grosse Hintergrunddatei verwenden, die vier Delta-Dateien würden typischerweise nur wenige hundert MB gross sein, statt 4 x 4 GB = 16 GB würden auf den Host dann vielleicht nur knapp 5 GB belegt.

COW-Disks haben aber weitere interessante Anwendungsbereiche. Durch einfaches Kopieren der Delta-Datei kann ein virtueller Snapshot/Flashcopy der COW-Datei erstellt werden, den man einerseits verwenden kann, um eine weitere COW-Datei (mit derselben Hintergrunddatei) zu erzeugen (Cloning), den man andererseits aber auch zu einem späteren Zeitpunkt zum Zurücksetzen der COW-Disk auf den Snapshot-Zeitpunkt einsetzen kann (Rollback). Die Rollback-Fähigkeit der COW-Disks entspricht bei VMware den „Undoable Disks“, die ideal sind, um ohne grosses Risiko eine gravierende Systemänderung – wie bspw. einen neuen Patch-Stand – zu testen: vor der Durchführung der Systemänderung erstellt man eine Kopie der Delta-Datei. Falls die Systemänderung wie erwartet gelingt, dann verwirft man diese Kopie einfach. Falls das Ergebnis der Systemänderung aber nicht den Erwartungen entspricht, dann bringt man das System wieder in den Zustand wie vor der Systemänderung, indem man die aktuelle Delta-Datei einfach durch die zuvor erstellte Kopie der Delta-Datei ersetzt.

VIRTUELLE ETHERNETS

UML-Gäste können über virtuelle Ethernets untereinander und mit dem Host vernetzt werden. Virtuelle Ethernet-Switches werden durch im Host laufende Prozesse realisiert und virtuelle Ethernet-Adapter durch lokale Unix-Sockets des Hosts. (VMware realisiert hingegen das virtuelle Networking durch ein in den Host-Kernel geladenes Kernel-Modul „vmnet“.) Die Realisierung des virtuellen Ethernets durch Prozesse statt Kernel-Module hat den Vorteil, dass auf einfache Weise die Injektion von Fehlern möglich ist („Fault-Injection“), bspw. indem man einen derartigen Prozess mittels „kill -SIGSTOP“ und „kill -SIGCONT“ temporär supendiert, um Netzwerk-Ausfälle zu simulieren, was u.a. für Tests von HA-Clustern von Interesse ist. Darüberhinaus könnte man mit einfachen Änderungen im Source-Code des virtuellen Ethernet-Switches weitere Möglichkeiten zur Fault-Injection schaffen, bspw. zufallsgesteuerte Verfälschungen von Paketinhalten oder hohe Latenzzeiten, um unzuverlässige oder sehr lange Kommunikationsstrecken zu simulieren, wodurch sich u.a. die Robustheit von Netzwerkprotokollen auf die Probe stellen lässt.

VIRTUELLE TTYS

Neben virtuellen Ethernets unterstützt UML auch virtuelle TTYs (serielle Schnittstellen und Terminals) zur Kommunikation von Gästen untereinander, von Gästen mit dem Host und mit Benutzern. TTYs können auf unterschiedliche Weise im Host emuliert werden. Das TTY eines Gasts kann im Host als XTerm erscheinen, was bspw. sinnvoll ist, wenn im Gast auf diesem TTY ein Login-Prozess (getty) läuft. Das TTY eines Gasts kann aber im Host auch durch ein Pseudo-Terminal-Slave (/dev/pts/...) emuliert werden, auf das der Host mittels des Kommandos „cu“ (aus dem UUCP-Paket) oder eines Terminalprogramms, z.B. Minicom, zugreifen kann.

Die Emulation von TTYs als Pseudo-

Terminal-Slave ist aber auch sehr geeignet, um die TTYs zweier Gäste zu koppeln, die einen HA-Cluster bilden und über eine serielle Heartbeat-Verbindung verfügen sollen. Hierzu werden die Pseudo-Terminal-Slaves der beiden Gäste bidirektional verbunden, bspw. mittels „cat /dev/pts/17 >> /dev/pts/18 & cat /dev/pts/18 >> / dev/pts/17“. Sehr interessant ist hier die Möglichkeit, die virtuelle serielle Verbindung vom Host aus zu belauschen (Sniffing), indem man statt des Kommandos „cat“ einfach „tee“ verwendet. Auf diese Weise kann man bspw. Heartbeat-Protokolle auch auf seriellen Verbindungen analysieren, was in realen Testumgebungen spezielle Hardware erfordern würde.

UML, VMWARE, LPAR

Es gibt einige Gemeinsamkeit, aber auch interessante Unterschiede zwischen UML, VMware und Logical Partitions (LPAR).

EMC VMware (und Microsoft Virtual Server) emuliert im Unterschied zu UML eine komplette PC-Hardware (Virtual Machine, VM) einschliesslich BIOS, auf der nicht nur Linux, sondern jedes beliebige PC-Betriebssystem (z.B. Windows, OS/2, Netware,..) ausgeführt werden kann. Mit VMware kann im Gegensatz zu UML deshalb ein regulärer Linux-Kernel (bspw. von einer SuSE Distributions CD) in einer VM ausgeführt werden, wohingegen der UML-Kernel ja ein speziell modifizierter Linux-Kernel ist, der nur auf dem Host-Betriebssystem – nicht aber auf der blossen Hardware (egal ob real oder emuliert) – ausgeführt werden kann.

Logical Partitions von IBM eServer Systemen, bspw. LPARs von zSeries-Mainframes oder POWER5-Micropartitions, werden durch den in der System-Firmware implementierten „Hypervisor“ des IBM eServer in ähnlicher Weise wie bei VMware realisiert. In einer LPAR kann deshalb ebenfalls ein ganz normales Betriebssystem mit einem regulärem (Linux-, AIX-, z/OS-, OS/400-,..) Kernel ausgeführt werden, der aber beim Booten feststellt, dass er in einer LPAR läuft, und dann bei bestimmten Hardware-Zugriffen (z.B. bei DMA-Zugriffen auf Devices oder Modifikationen der Page-Translation-Tables) spezielle Hypervisor-Calls durchführt.

Der offensichtlichste Unterschied von UML gegenüber VMware und LPARs ist aber, dass UML – als auf dem regulären Linux-Kernel aufbauendes Projekt – auch unter der GNU Public License (GPL) steht (stehen muss), also Free Software / Open Source Software ist, die kostenlos genutzt werden kann, an deren Weiterentwicklung jeder teilnehmen kann, oder die zur Basis anderer (dann aber ebenfalls unter der GPL stehender) Projekte gemacht werden kann. Bei VMware handelt es sich hingegen um eine klassische kommerzielle Closed Source Software, an der man nur ein sehr eingeschränktes Nutzungsrecht erwerben kann, und bei der IBM System-Firmware handelt es sich natürlich ebenfalls um verschlossene Software – IBM betrachtet System-Firmware und Microcode sogar als Teil der Hardware.

LPARs sind wesentlich effizienter als UML und VMware, was bedeutet, dass sie nur einen sehr geringen Overhead und wenig negativen Einfluss auf die Performance der Gäste haben. VMware ist zwar nicht ganz so effizient wie LPAR, aber deutlich performanter als UML. Das „User“ in „User Mode Linux“ unterstreicht, dass UML eine reine User-Space-Implementierung ist, also ohne spezielle Kernel-Module oder Microcodes auskommt, was bei der UML-Realisierung der virtuellen Ethernets im Vergleich zu VMware besonders deutlich wird. Zusammen mit der Offenheit der Source-Codes ermöglicht die Implementierung im User-Space deutlich mehr Flexibilität und Möglichkeiten der Fault-Injection, was sehr bedeutend für den Einsatz als virtuelle Testumgebung ist. Der Preis dafür ist aber eine geringere Effizienz, insbesondere geringere Performance als bei einer Implementier.ung, die tiefer im Host-System angesiedelt ist. Man kann deshalb den Schluss ziehen, dass sich virtuelle Testumgebungen auf Basis von UML in vielen Fällen für funktionale Tests gut eignen, wegen der geringen Performance aber i.a. nicht für Last- und Performance-Tests.

EINSATZSZENARIEN

Mittels User Mode Linux wurden folgende virtuelle Testumgebungen aufgebaut, in denen dann funktionale HA-Tests durchgeführt wurden:

- **2-Node-HA-Cluster.** Zwei UML-Gäste

bilden einen HA-Cluster, der über ein Public-Ethernet, ein Private-Heartbeat-Ethernet, eine Serial-Heartbeat-Connection, eine Shared Disk und eine umschaltbare IP-Alias-Adresse als Service-IP-Adresse für den NFS-Service verfügt und mit der Open-Source-Software „Heartbeat“ (aus dem „Linux-HA-Projekt“ von Alan Roberts) einen hochverfügbaren NFS-Service realisiert. Die beiden Cluster-Knoten bilden dabei einen einfachen Active/Passive-Cluster, sprich höchstens ein Knoten hat zu jedem Zeitpunkt die Resourcen Disk und Service-IP-Adresse exklusiv zugeordnet und bietet den NFS-Service über das Public-Ethernet an. Der andere Cluster-Knoten wartet auf den Ausfall des ersten und übernimmt dann die Resourcen Disk und Service-IP-Adresse und startet den NFS-Service.

Ein dritter UML-Gast greift in diesem Test-Setup als NFS-Client über das Public-Ethernet auf das per NFS und Service-IP-Adresse gemountete Filesystem zu.

Heartbeat ist ein sehr einfaches, freies HA-Cluster-Produkt, das sich gut für sehr einfache HA-Anforderungen eignet.

- **Remote-Disk-Mirroring.** Mittels der ebenfalls aus dem Linux-HA-Projekt stammenden Software Distributed Remote Block Device (DRBD) wurde die Shared Disk in dem zuvor beschriebenen Szenario durch ein über das Ethernet und IP-Protokoll gespiegeltes Disk-Paar ersetzt.

  Diese Technologie ähnelt EMC Symmetrix Remote Data Facility (SRDF), IBM Peer-to-Peer-Remote-Copy (PPRC), sowie IBM Global Mirror Devices (GMD). Sie wird typischerweise in HA-Clustern eingesetzt, wenn die Cluster-Knoten sehr weit voneinander entfernt sind, sodass die Realisierung von Shared Disks nicht mehr möglich oder sinnvoll ist.

  Die Datenreplikation zwischen den Disk-Paaren kann synchron und asynchron erfolgen.

- **4-Node-OSPF/VIPA-Domain.** Vier UML-Gäste werden über dedizierte Ethernets im Ring zusammengeschaltet. Jeder virtuelle Knoten verfügt also über zwei Ethernet-Adapter eth0 und eth1, durch die er mit den zwei benachbarten Knoten im Ring über einen Ethernet-Switch verbunden ist. Jeder Ethernet-Adapter hat eine physikalische IP Adresse (PIPA). Darüber hinaus hat jeder Knoten eine statische virtuelle IP Adresse (VIPA) auf dem Interface dummy0, dem kein Ethernet-Adapter zugeordnet ist. Eine derartige VIPA ist unabhängig von einzelnen Ethernet-Adaptern und kann – geeignetes IP-Routing vorausgesetzt – solange erreicht werden, wie der Knoten noch mindestens einen funktionsfähigen Ethernet-Adapter hat. Eine VIPA ist also eine hochverfügbare IP-Adresse.

  Durch die Open-Source-Software Quagga (ehemals Zebra) wird auf den vier Knoten das dynamische OSPF-Routing-Protokoll realisiert, das dafür sorgt, dass die Kernel-Routing-Tabellen aller Knoten immer automatisch mit Routen versorgt werden, um stets alle theoretisch erreichbaren PIPAs und VIPAs auch tatsächlich erreichen zu können, selbst wenn zwischenzeitlich einzelne Ethernet-Adapter, Netzwerkabschnitte oder Knoten ausgefallen sind.

  Die Konvergenzzeit, d.h. Zeitdauer nach einem Ausfall bis zur Isolation und Umgehung der ausgefallenen Komponente, beträgt in derartigen Systemen typischerweise 10-20 Sekunden.

- **IBM Tivoli System Automation (TSA).** Aufbauend auf der zuvor beschriebenen 4-Node-OSPF/VIPA-Domain wurde mittels IBM RSCT (Reliable Scalable Cluster Technology) ein sogenannter Remote-Peer-Domain-Cluster definiert, der mit seinen Topology- und Group-Services die Basis für diverse IBM HA-Produkte (HACMP, GPFS, CSM, TSA) bildet. Zu den Topology- und Group-Services gehören u.a. Heartbeat-Protokolle und Methoden zum erzwungenen Ausschluss (Fencing) fehlverhaltender Knoten, wie bspw. Dead-Man-Switches.

  Auf diesem RPD-Cluster wurde dann

mittels TSA eine Resource Group definiert, die eine dynamische VIPA – die im Gegensatz zu einer statischen VIPA nicht auf einen einzigen Knoten festgelegt ist – und eine beispielhafte Server-Anwendung – der Einfachheit halber nur ein SSH-Daemon – enthält. Alle Ethernet-Adapter wurden zu einer Equivalency zusammengefasst. Schliesslich wurden Depends-On-Relationen zwischen den Resourcen SSH-Daemon und dyn. VIPA einerseits, sowie dyn. VIPA und der Equivalency der lokalen Ethernet-Adapter andererseits, definiert.

TSA ist ein sehr elaboriertes, kommerzielles HA-Cluster-Produkt, das sich gut für komplexe HA-Anforderungen eignet.

Abgesehen von Szenarien im Kontext von Hochverfügbarkeits-Systemen bieten sich virtuelle UML Testumgebungen auch gut für Proofs-of-Concepts oder Penetrations-Tests von geplanten System-Architekturen im Kontext von Security-Systemen, z.B. mit Firewalls, DMZs, Intrusion-Detection-Systemen, Web-Servern, usw. an.

FAZIT

UML eignet sich sehr gut, um virtuelle Labor- und Test-Umgebungen zu erstellen, wegen seiner ausgeprägten Möglichkeiten zur Fault-Injection insbesondere für funktionale Tests von Hochverfügbarkeitssystemen. Man kann komplexe Test-Szenarien, bestehend aus mehreren Linux-Systemen mit Festplatten, seriellen Verbindungen und Ethernets bereits auf einem Notebook aufsetzen, sodass diese virtuellen Labore in einfacher Weise transportiert, präsentiert und gemeinsam mit anderen studiert werden können. Und schliesslich kann man einfach ein ganzes Test-Labor auf eine CD/DVD brennen und so anderen zugänglich machen.