UnifiedSessionsManager - Virtualisierung und Cloud-Computing

Arno-Can Üstünsöz
Ingenieurbüro Arno-Can Üstünsöz, München, Deutschland, acue@i4p.com

Kurzfassung

Das OpenSource-Produkt UnifiedSessionsManager(USM) stellt eine Middleware als Entwicklungs und Laufzeitumgebung für die Anwendung von lokalen und verteilten virtuellen Maschinen(VMs) bereit. Es werden Einzelplatz-Syteme - lokale und entfernte - private und öffentliche Clouds unterstützt. Der Einsatztbereich umfaßt Entwicklungsumgebungen und Emulationssysteme, ebenso Produktivsysteme für Endbenutzer und Systemadministratoren. Der Anwendungsbereich reicht von der Inventarisierung und Verwaltung virtualisierter DV-Landschaften und Cluster-Systemen bis zu Einzelplatzsystemen und virtualisierten Anwendungsumgebungen. Insbesondere bietet dies die Versionierung und Archivierung von Laufzeitumgebungen so auch Entwicklungs- und Testumgebungen, konsistenten Middleware-Strukturen und Simulations-Umgebungen.

1 Aktuelle Einsatzszenarien für VMs

Die typischen aktuellen Einsatzszenarien für VMs sind technologie orientiert, d.h. bieten die Bereitstellung einer virtualisierten Laufzeitumgebung als Infrastruktur(IaaS). Der durch den Benutzer angeforderte Dienst wird i.A. als spezifisches Template bereitgestellt, bzw. ist als individuelle Ausprägung durch diesen selbst zu integrieren. Diese sog. Appliances - vorkonfigurierte VMs - sind falls generisch i.A. nur für einen spezifischen Zweck vorkonfigurierte "virtualisierte Monolithen". Die VMs fungieren hierbei als Middleware-Container.

Dieses Einsatzszenario ist gleichermaßen typisch für sog. private und public Clouds. Die Verwaltung und Inventarisierung unterliegt der Obhut des Anwenders. So wäre im Falle einer Simulationsumgebung hier z.B. ein vollständiger Rechenknoten mit einer speziellen vorkonfigurierten Laufzeitumgebung durch den Benutzer bereitzustellen. Eine eventuelle Resource-Anforderung beispielsweise für lokale GPUs und zugehörige Laufzeitumgebungen und/oder erhöhte Speicheranforderungen bzw. erforderliche lokale Datenbank-Zugriffe ist vom Anwender zu handhaben.

Die Addressierung der virtuellen Maschine durch den jeweiligen Hypervisor, ebenso das Benutzer-Login in das Gastsystem sind durch das spezifische System vorgegeben. Ebenso ist die Definition von Gruppen-Objekten - sofern vorhanden - spezifisch für das jeweilige System. Somit ist z.B. eine integrierte Addressierung mittels vorgefertigter persistenter Scripts nicht möglich.

2 Erweiterte VM-Middleware

Durch das Toolset USM wird ein Satz von umfassenden Werkzeugen als Middleware bereitgestellt [1, USM] , die insbesondere den individuellen Einsatz virtualisierter Umgebungen unterstützen. Dies wird u.A. durch die einheitliche Schnittstellen-Syntax, das Herstellerübergreifende Addressierungskonzept ebenso durch die unterstützenden Dienste erreicht.

Der wesentliche Ansatz ist hierbei die Einbettung von Aufrufen in eine Laufzeitumgebung mit einer Anpassungs-Schicht(Bild 1), die sowohl kommandozeilen-basiert, als auch mittels Bibliothek-Schnittstellen beliebige Systeme integriert.

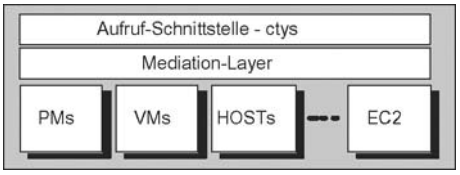

Bild 1. Laufzeitumgebung mit Plugin Struktur

Neben der reinen Laufzeitumgebung werden hier ergänzende Zusatzdienste und Funktionalitäten integriert, die eine Harminosierung der Zuordnung und Addressierung der Resourcen bereitstellen. Ziel ist hierbei die herstellerunabhängige integrierte Nutzung verschiedener Platformen und Produkte. Ein Wesentlicher Aspekt ist die nahtlose Addressierung aller unterstützter Komponenten. Dies umfaßt die physikalischen Host-Systeme, die virtuellen Maschinen, als auch die jeweiligen Konsolen und Desktops.

Unterstützt wird hierbei die integrierte Nutzung von lokalen Systemen, dem Benutzer-Rechner und Rechnern in einem privaten Netz - einer privaten Cloud - oder einem öffentlichen Rechnerverbung - einer public Cloud. Hierbei wird sowohl eine rein dynamisch arbeitende Umgebung ohne erforderliche Server, als auch eine Server basierte Umgebung bereitgestellt. Durch die Modularisierung ist prinzipiell eine Sprachunabhängigkeit der Submodule für spezifische Erweiterungen gegeben.

Der Vorteil der Variante ohne erforderliche Server ist hierbei die einfache Installation und flexible Anwendbarkeit ohne erforderliche Administratorrechte für einfache Desktop-Sitzungen. Die Server basierte Variante bietet eine bessere Performance durch den permanenten Daten-

abgleich der dynamischen Stati. Die Grundfunktionalität beider Varianten ist im Wesentlichen identisch, ein variabler Mischbetrieb deckt z.B. Anforderungen zur Administration einer Verwaltungs-Ebene für PMs und VMs durch die Server basierte Variante in Kombination mit der Nutzungsebene der VMs durch dynamische Login-Desktops für Administratoren und Benutzer ab.

2.1 Desktop als Integrations-Plattform

Der Grundansatz des UnifiedSessionsManager umfaßt insbesondere auch die Verwaltung von Desktop-Oberflächen Zwecks Integration von heterogenen Umgebungen zu einem individuellen Anwendungs-Desktop. Hierzu wurde als Basis Client basierter Dialoge die Unterscheidung zwischen DISPLAYFORWARDING und CONNECTIONFORWARDING getroffen.

- DISPLAYFORWARDING
  Anwendungs-Client wird analog zu X11 grafisch auf die Oberfläche der Benutzer-Maschine abgebildet, jedoch entfernt ausgeführt - i.A. co-alloziert mit dem Server-Prozeß. Die Verbindung erfolgt einheitlich(für X11) mittels 'ssh -X ...' durch X11-Forwarding(Bild 2).

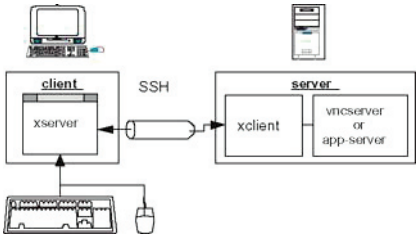

Bild 2. DISPLAYFORWARDING

- CONNECTIONFORWARDING
  Anwendungs-Client wird auf der Benutzer-Maschine ausgeführt und lokal auf die Oberfläche abgebildet. Die Verbindung erfolgt einheitlich mittels automatisch erzeugtem SSH-Tunnel durch Port-Forwarding(Bild 3).

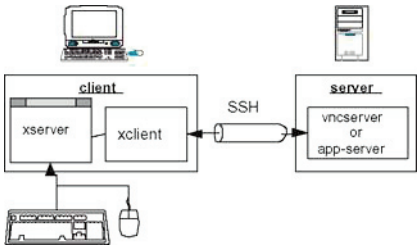

Bild 3. CONNECTIONFORWARDING

Dies ermöglicht insbesondere auch die Nutzung von propriäteren graphischen Clients, die in jedem Falle mittels DISPLAYFORWARDING dargestellt werden können.

Die gleichzeitige Nutzung von verschiedenen Sichten beruht hierbei insbesondere auf der Anwendung sog. Workspaces(Bild 4). Dies beinhaltet neben der Verwaltung von Desktop-Oberflächen auch die vollständige Automatisierung aller Funktionen.

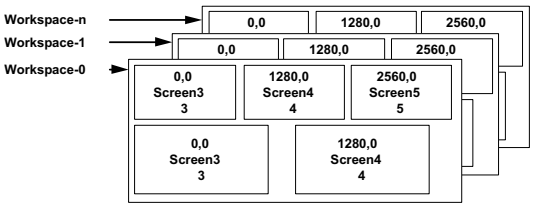

Bild 4. Workspaces als individuelle Oberflächen

Die Automatisierung umfaßt sowohl die Inventarisierung, als auch die Erstellung benutzerspezifischer Anwendungs-Makros und Scripte mit zusammengesetzten Komponenten.

Das Suchen und Erfassen der vorhandenen VMs(Bild 5) erfolgt beispielsweise mittels des Aufrufs

ctys-vdbgen – host01 host02 host03

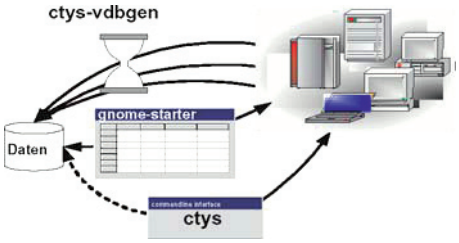

Bild 5. Automatisierte Inventarisierung - GUI-Integration

So können z.B. MS-Windows und Linux basierte Client und Server-Anwendungen auf den verschiedensten physikalischen und virtuellen Systemen durch eine persistente Konfiguration als integrierte Anwendungsoberfläche gespeichert und abgerufen werden.

Insbesondere wird durch das Abbilden von statischen benutzerdefinierten Bezeichnern zur Laufzeit auch für unabhängige herstellerübergreifende dynamische Addressierungselemente eine persistente Speicherung möglich.

Ein Beispiel ist hier die optionale Benutzung von Bezeichnern für die Addressierung der Konsolen aller unterstützter Systeme, so auch von VNC-Ports, RDP-Ports, KVM, QEMU, Xen, VirtualBox und VMware-Konsolen. Für VNC wird zum Beipiel die Aufrufreihenfolge

ssh -X host01
vncserver
vncviewer localhost:11

ersetzt durch

```
ctys -a create=LABEL:myLabel host01
```

Dies wird insbesondere auch in verteilten Multi-Benutzer-Umgebungen mit jeweils unabhängig vergebenen lokalen Consolen-Addressen auf eine statische Definition abgebildet. Ebenso wird durch die dynamische Verwaltung die herstellerübergreifende Neu-Vergabe der freien Consolen-Addressen gesteuert. Dies ermöglicht den einfachen script-gesteuerten Aufbau eines zusammengesetzten X11-Desktops mittels Nutzung symbolischer Bezeichner für das sog. DISPLAYFORWARDING.

2.2 Schichten-Architektur mittels VM-Stacks

Die Definition der zugrundeliegenden Schichten-Architektur orientiert sich an der funktionalen Trennung der enthaltenen Software-Komponenten. Dies hat insbesondere den Vorteil einer einfachen zukünftigen Erweiterung des Einsatzes verschachtelter VMs. Der aktuell verbreitete Ansatz der Unterteilung in Client- und Server-Virtualisierung wird hier ebenso abgebildet, die Unterscheidung entfällt. Die Darstellung(Bild 6) erfolgt in Anlehnung an die Abbildung von B-ISDN-Layern in den ITU-T Normen.

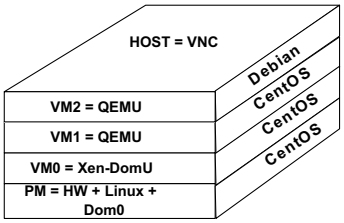

Bild 6. Automatisierte Inventarisierung - GUI-Integration

Der Einsatz von VMs kann in drei Schichten unterteilt werden.

- **PMs**
  Physikalische Maschinen als Container von VMs. Diesen kommt eine besondere Bedeutung zu, da diese zum Zeitpunkt der Dienst-Anforderung u.U. noch ausgeschaltet sind und somit für reale Systeme besondere Vorkehrung und Mechanismen für das Einschalten erfordern, beispielsweise mit WoL oder IPMI.
- **VMs**
  VMs werden i.A. über den entsprechenden Hypervisor aktiviert und erfordern gegebenenfalls besondere Zugriffsberechtigungen für spezifische Resourcen. Ebenso unterscheiden sich die Addressierung, Zugriffsmechanismen, Sicherheitseinrichtungen, und die Möglichkeiten der Benutzer-Konsolen erheblich, sowohl für spezifische Hypervisor, als auch zwischen den einzelnen Produkten. Ein Hypervisor kann grundsätzlich wiederum Hypervisor enthalten und für diesen eine vermeintliche physikalische Maschine(PM) representieren. Dies ist z.B. regelmäßig im Falle der Emulation von Embedded-Systemen in virtuellen Arbeitsumgebungen der Fall. Ein typisches Beispiel ist hier der Einsatz von QEMU für spezifische CPUs.
- **HOSTs**
  HOSTs representieren die Benutzersitzungen in das Gast-System. Dies sind sowohl interaktive, als auch Batch-Sitzungen.

Eine konkrete Instanziierung der jeweiligen virtuellen Maschinen in einem Stack bildet durch die vertikale Abhängigkeit eine logische Baum-Struktur.

Diese logisch vertikale Struktur(Bild 7) kann jedoch bei entsprechender Unterstützung durch das Betriebssystem und CPUs als flach verteilte Instanzen in einer CPU-

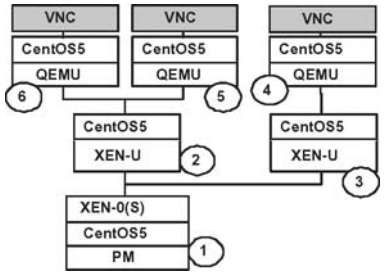

Bild 7. vStack-Instanzen als log. Baum-Struktur

Matrix(Bild 8) ausgeführt werden. D.h. insbesondere nebenläufig und praktisch ohne Laufzeit-Verluste.

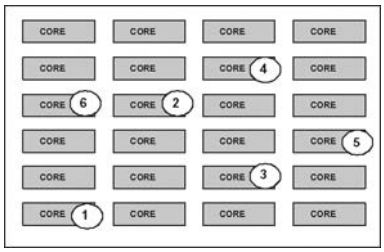

Bild 8. vStack-Instanzen verteilt auf CPU-Core-Matrix

In den aktuellen Tests werden Emulatoren - z.B. Qemu - eingesetzt, so daß dies zu kaskadierten CPU-Laufzeiten führt.

Dieser für die Performance in einer Cluster-Umgebung i.A. Nachteilige Ansatz einer Emulation ist im Bereich der CPU-Emulation, z.B. für Embedded Systeme jedoch insbesondere eine gute Möglichkeit kostengünstige Entwicklungsumgebungen bereitzustellen. Dies wird z.B. durch für die Android-Entwicklung praktiziert.

2.3 Schnittstellensyntax

Der USM ist in der 'serverless' Variante als selbst installierendes Kommandozeilen-Toolset mit einer dynamischen Plugin-Architektur ausgeführt. Es wird hierbei ein Basis-Satz von generischen externen Kommando-Optionen vorgegeben, die intern durch das jeweilige Technologie-Plugin implementiert sind.

Die einheitliche Schnittstellensyntax folgt dabei dem Muster:

ctys -t <type> -a <action> <opts> <target>

Wobei zu jeder <action> spezifische Parameter angegeben werden können

<action>:=<Action>[=<parameter>]

Ebenso werden zu jedem Ziel <target> der vollständige Umfang der Kontextoptionen unterstützt. So kann der obige Aufruf alternativ wie folgt formuliert werden:

```
ctys <local-opts> \
   <target>'(
      -t <type> \
      -a <action> \
      <remote-opts>
   )'
```

Der Aufruf enthält mit <opts> zusätzlich optionale Parameter die z.B. die Addressierung von Multi-Monitor Desktops unterstützen.

In der Variante ohne Server ist das Laufzeitsystem des USM selbst als komplettes Installationspaket ausgeführt und kann somit ohne zusätzliche Pakete lokal oder remote auf verteilte Systeme installiert werden. Der einfache Aufruf

ctys-distribute -P uhc [<user@>]<target1> [<user@>]<target2>

Installiert den USM auf dem Zielsystemen '[<user@>]<target1>' und '[<user@>]<target2>'.

2.4 Addressierungskonzept

Der USM implementiert als Integrationsplattform eine einheitliche Bedienschicht für unterschiedlichste vorhandene Komponenten. Das Addressierungskonzept definiert hierbei notwendigerweise die Obermenge der vorhandenen Addressierungsmöglichkeiten. Die aktuell unterstützten Systeme sind u.A. CLI,X11,RDP,VNC,KVM,QEMU,VirtualBox, VMware(Player/Server/Workstation), Xen und physikalische Maschinen - derzeit noch teilweise unterstützte Systeme sind XenServer und VMware-ESX/ESXi, in Arbeit sind zudem 'Amazon-EC2' und 'Google Apps Engine'. Die aufgeführten Systeme werden zudem sowohl als Host, als auch als Guest unter den verschiedensten Betriebssystemen unterstützt, so z.B. Linux-Varianten, BSD-Varianten, Solaris und Windows-Varianten. Die Vielfalt der unterstützten Systeme und deren Kombinationen resultiert in einer umfangreichen Anzahl von Varianten an Optionen und Addressierungsmöglichkeiten.

Der gewählte Ansatz definiert zunächts eine gemeinsame Address-Syntax für die jeweiligen Schichten. Diese werden sofern erforderlich durch Laufzeitkomponenten harmonisiert. So variiert die Addressierung der VMs jeweils stark. So - bei Einbeziehung der zu addressierenden Laufzeit-Aufrufe - auch zwischen den originär gleichen QEMU und KVM. Im Folgenden sind einige Eigenschaften beispielhaft aufgelistet.

1. KVM
   Keine Konfigurations-Datei, plattformabhängige Aufrufe, Address-Definition durch dynamische Kommandozeilen-Parameter.
2. QEMU
   Keine Konfigurations-Datei, plattformabhängige Aufrufe, Address-Definition durch dynamische Kommandozeilen-Parameter. Umfangreiche CPU-Emulationen, z.B. div. ARM-Varianten.
3. VBOX
   Addressierung mittels Registrierung bei zentralem Dienst bzw. in dem Laufzeit-Image hinterlegter Informationen und/oder einer getrennten Konfigurationsdatei.
4. VMW
   Verwaltung durch eigene Konfigurationsdatei, z.T. dynamische Anpassungen bei Veränderung der Dateiposition(siehe UUID).
5. XEN
   Verwaltung durch eigene Konfigurationsdatei als Python Script, dieses kann um dynamischen Programmcode ergänzt werden. Addressierung erfolgt durch Datei-Position bzw. nach Start durch Registrierung der sog. DomU bei der sog. Dom0. Verschiedene Konsolen mit dynamischen Addressen, auch im sog. Headless-Mode.

Die HOSTs-Sitzungen, d.h. die Benutzersitzungen in das jeweilige Gast-System haben i.A. keine eigene HW-Kennung sondern lediglich einen dynamischen Sitzungsschlüssel. Dieser ist bei den gegebenen Systemen ein i.A. nicht(oder nur bedingt) frei wählbarer numerischer Wert. Zu berücksichtigen ebenso ist die Verwaltung der freien Ports und die Sicherstellung der Eindeutigkeit der Schlüssel in einer verteilten Multi-Benutzer-Umgebung, da diese i.A. lediglich systemlokal eindeutig sind. Insbesondere überschneiden sich u.U. Wertebereiche von VNC-Ports, die z.B. neben VNC auch von KVM, QEMU, Xen und VMware-WS benutzt werden. Im Falle der unterbrechungsfreien Laufzeitverschiebung der virtuellen Maschinen sind zudem zusätzliche Vorkehrungen zur Gewährleistung der Verfügbarkeit freier Ports und der Synchronisation eventuell vorhandener Clients zu treffen.

Die PMs sind funktional Analog zu HOSTs, die Abweichung bezieht sich primär auf die Handhabung der Ein- und Ausschaltvorgänge.

Der Ansatz des USM ist hier die konsistente Definition einer Superpositionierten Address-Syntax mit spezifischen Ausprägungen für die jeweiligen Schichten. Der folgende Auszug repräsentiert die Address-Syntax des jeweiligen Knotens:

```
MachineAddress:=SEQUENCE{
  id          [0]IMPLICIT Identifier,
  base        [1]IMPLICIT Base,
  label       [2]IMPLICIT Label,
  filename    [3]IMPLICIT Filename,
  uuid        [4]IMPLICIT Uuid,
  mac         [5]IMPLICIT Mac,
  tcp         [6]IMPLICIT Tcp
}
```

So sind bei der Addressierung z.B. Bezeichner mit Angaben nach Tabelle 1 möglich. Es wird hier durch 'U' die durch USM erforderliche Bereitstellung, mit 'X' die Bereitstellung durch das jeweilige System gekennzeichnet.

Für die HOSTs Sessions CLI, RDP, VNC und X11 sind TCP/IP bzw. MAC-Addressen nicht verfügbar, als kanonische ID werden DISPLAY-Variable, TCP/IP-Ports bzw. Prozeß-PIDs genutzt. Einige Plugins unterstützen Produkte mit versionsabhängigen Ausprägungen der Bedienschnittstellen, so daß hier zur Vereinheitlichung der Schnittstelle eine interne Anspassung vorgenommen werden muß. Die Diversität der unterstützten Systeme wird durch die vereinzelte Notwendigkeit ergänzender spezifischer Optionen abgebildet.

Plugin	LABEL	ID|PATH	MAC	UUID
AENG	ffs	ffs	ffs	ffs
CLI	U	U	-	-
EC2	ffs	ffs	ffs	ffs
KVM	U	X	U	U
OVP	ffs	ffs	ffs	ffs
PM	U	U	U	U
QEMU	U	X	U	U
RDP	U	U	-	-
VBOX	U	X	U	X
VMW-ESX	ffs	ffs	ffs	ffs
VMW-ESXi	ffs	ffs	ffs	ffs
VMW-P	U	X	U	U
VMW-S	U	X	U	U
VMW-WS	U	X	U	U
VNC	U	X	-	-
X11	U	U	-	-
XEN	U	X	U	U
XEN-S	ffs	ffs	ffs	ffs

Tabelle 1. Anwendbare Address-Elemente

Die vereinfachte Aufruf-Syntax

```
ctys <local-opts> \
   <target>'(
      -t QEMU \
      -a CREATE=<machine-address> \
      <remote-opts>
   )'
```

definiert <target> als den Ausführungshost, die <machine-address> in den Parametern zu CREATE definiert die auszuführende VM. Z.B.

-a CREATE=MAC:00:11:22:33:44:55

2.5 Netzwerk-Dienste

Die Integration einer vereinfachten Verwaltung der Netzwerk-Integration für den End-Nutzer ist ein Wesentlicher Aspekt für die Nutzung von großer Mengen virtueller Maschinen als Softwarekomponenten. Dies betrifft die Erzeugung neuer Maschinen als auch den Zugriff auf verschachtelte virtuelle Maschinen, die bei Speicherung innerhalb einer virtuellen Maschine vor dem Start der enthaltenden nicht 'sichtbar' sind. Grundsätzlich kann eine VM in jeder Ebene eines virtuellen Stacks gestartet werden, sofern die Laufzeitvoraussetzungen erfüllt sind. Die Voraussetzungen werden i.A. dynamisch geprüft.

Hier wird zwischen den folgenden Grundsätzlichen Fällen unterschieden:

- VMs sind über ein Netzlaufwerk sichtbar
- VMs sind über ein Netzlaufwerk nicht sichtbar

Sichtbare VMs können dynamisch verwaltet werden, wohingegen gekapselte VMs mittels Unterstützung durch eine statische Datenbank addressiert werden. Der reale Aufbau eines virtuellen Stacks erfordert umfangreiche virtualisierte Kommunikations-Elemente(Bild: 9). So beispielsweise virtuelle Netwerkinterfaces, virtuelle Switches und virtuelle Router.

Als erster Ansatz ist eine statische Verwaltung der TCP/IP- und MAC-Addressen einschließlich der DNS-Namem implementiert. Diese ist mittels Konvertierungs-Werkzeugen statisch mit den Datenbeständen der Standard-Dienste DNS/Bind und DHCP integriert. Neue

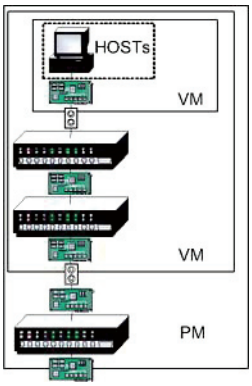

Bild 9. Kommunikationselemente virtueller Stacks

Datensätze werden bereits bei der Erzeugung neuer VMs automatisiert um Zuordnung der TCP und MAC-Addressen ergänzt.

Die zentrale Datenbank wird automatisiert generiert und enthält den vollständigen Datensatz eines Knotens. So auch enthaltende Maschine und Beschreibung der Systemresourcen wie z.B. CPUs, RAM und Plattenplatz. Ebenso wird bei der Einrichtung eine standardisierte dynamische Leistungsmessung der CPU, Speicher und Festplatten-Geschwindigkeit durchgeführt, diese werden derzeit um GPU-Kenngrößen ergänzt. Diese Parameter werden u.A. zur automatisierten Lastverteilung verwendet.

Durch die integrierte zentrale Datenhaltung kann vom Benutzer jederzeit eine Abfrage und Zuordnung zu weiteren Parametern erfolgen. Die Datenbank bietet ebenso eine einfache Export-Schnittstelle zur individuellen Weiterverarbeitung.

2.6 Inventarisierung

Die Erfassung der VMs erfolgt manuell oder durch automatisiertes Scannen. Dies erleichtert insbesondere die Verwaltung von Test- und Entwicklungsumgebungen mit einer großen Anzahl von VMs, ebenso die Erfassung von VMs verschiedener Typen.

Die Erfassung ermöglicht die einfache Angabe von Filtern, so daß lediglich ausführbare VMs inventarisiert werden, dies bietet bei Einsatz auf Produktiv-Maschinen eine schnelle Übersicht aktuell verfügbarer Resourcen. Der Filter zur Auflistung aller erreichbarer VMs bietet Vorteile bei Test- und Entwicklungs-Umgebungen mit der Möglichkeit des Kernel- bzw. Betriebssystem-Wechsels. So ist es möglich virtualisierte Laufzeitumgebungen zu verschiedenen Hypervisorn und HOST-Betriebssystemen einzuscannen und für einen vorgegebenen Knoten alle unterstützten Systeme aufzulisten.

Ebenso werden sog. Sichten unterstützt. Diese entsprechen einer Datenbank, die in der einfachen Variante eine Office-Kompatible Tabelle als Text-Datei enthält. Durch gefilterte Inventarisierung und/oder manuelles editieren werden somit Mengen sichtbarer Resourcen definiert, die beispielweise als kompatibler Pool zu einer Anwendung vordefiniert werden. Ebenso kann in Verbindung mit Zugriffsrechten hierüber die Verwaltung benutzerindividueller Systemumgebungen erfolgen. Die Auswahl erfolgt bei

Datei basierten Datenbanken einfach durch die Angabe eines Verzeichnispfades.

2.7 Automatisierung

Die aktuelle Schnittstelle ist als Kommandozeilenaufruf ausgeführt, so daß eine einfache Einbindung in benutzerspezifische Programme und Scripte erfolgen kann.

Die aktuelle Version, ebenso die Folgeversionen, sind in einer Plugin-Architektur implementiert. Dies ermöglicht eine einfache Ergänzung um zusätzliche spezifische Komponenten, die wiederum durch die Kommandozeilen-Schnittstelle aufgerufen werden. Somit wird bei Integration neuer Plugins durch die Laufzeitumgebung bereits ohne Zusatzaufwand eine Automatiserungs-Schnittstelle bereitgestellt.

2.8 Sicherheit

Die Kommunikation erfolgt ausschließlich mittels SSH-Verbindungen, dies gewährleistet eine weitestgehend sichere Kommunikation, durch sog. Tunneln auch im Falle von Graphischen Desktops.

Die Nutzung einer getrennten und einheitlichen Protokoll-Ebene bietet eine Vereinfachte Entwicklung und Anwendung. Hierzu wurde als Basis zur Dialog-Absicherung die Definition von DISPLAYFORWARDING(Bild 2) und CONNECTIONFORWARDING(Bild 3) getroffen. Dies ermöglicht insbesondere die einheitliche Nutzung von gesicherten Verbindungen. In Fällen mit erforderlichem Protokoll-Wechsel - z.B. RDP - zum Zugriff auf eine PM wird ein Ansatz des 'Letzten-Hops' analog zu SSH verwendet.

2.9 Integrierte Addressierung

Die Addressierung erfolgt neben der Basis-Angabe der <machine-address> auf Basis zusätzlicher Filter-Kriterien. Die Überprüfung der Parameter is grundätzlich in eine lokale Vor-Prüfung und eine abschließende Prüfung auf dem Zielsystem gegliedert. So erfolgt z.B. als erster Schritt die lokale Syntax-Prüfung aller Parameter, als letzter Schritt die dynamische Status-Prüfung des Plugins auf der Ziel-Maschine. Hierzu wird insbesondere für Parameter des Zielsystems bezüglich der statischen Verfügbarkeit einer VM die integrierte Datenbank genutzt.

Der Vorteil eines integrierten Systems mit zentraler Datenbank resultiert insbesondere aus der Zusammenfassung von Knoten-Addressen mit statischen Informationen zu Containment-Hierarchie und Resource-Attributen, ergänzt um Benutzerspezifische Angaben. Durch die Kombination in einem einfachen Format mit Abfrage-Möglichkeit wird die Implementierung von Anwenderspezifischen Zuordnungskriterien und die schnelle Dialog-Abfrage unterstützt.

Als Erweiterung zur Aktuellen Version des USM wird mit der Erweiterung der Address-Syntax die einfache Einbeziehung von Zusatzattributen einschließlich dynamischer Größen integriert. Dies ermöglicht insbesondere die Addressierung von Laufzeitumgebungen mit spezifischen Resource-Anforderungen. Die zusätzliche Einbettung der Address-Syntax als Datentyp einer abstrakten Sprache ermöglicht die dynamische Verwaltung durch einfache Operationen.

3 Virtuelle Stacks und Komponenten

Der aktuelle Einsatz von virtuellen Maschinen ist mittels flacher Strukturen von virtualisierten Appliances definiert. Diese werden im USM als virtuelle Komponenten definiert, die in einer VM als Container ausgeführt werden. Dies führt zu dem Konzept von verschachtelten VMs, wobei virtuelle Maschinen wiederum in virtuellen Maschinen ausgeführt werden. Die Weiterentwicklung dieses Ansatzes führt zu virtualisierten Komponenten die in einem Baukasten-System zu neuen Diensten zusammengesetzt werden können.

Der Ansatz virtualisierter Komponenten als 'Komplette Maschinen' stellt eine Analogie zu CORBA dar. Es wird aufbauend auf einer 'durchgängigen Hypervisor-Landschaft' eine 3-dimensionale Ausführungsmatrix definiert, die als einzige DV-Laufzeitvoraussetzung einen jeweils kompatiblen Hypervisor erfordert. Eine dynamische lokalisierung der virtuellen Maschine auf einem ausführenden Knoten erfolgt mit den Mitteln der jeweiligen Hypervisor. Zudem erfordert die Auszeit bei der Re-Lokalisierung im Laufenden Betrieb bei gegenwärtigen Produkten i.A. nur wenige Millisekunden. Die Definition der Zuordnung zu ausführenden Knoten bzw. die Relokalisierung kann sowohl nach Kriterien der dynamischen Last-Verteilung, als auch manuell erfolgen. Ein Kriterium der Zuordnung sind hier zudem die erforderlichen System-Resourcen. so kann eine Anwendung per Definition die Verfügbarkeit einer GPU erfordern, ebenso ist ein zulässiges Kriterium der Zugriff auf die GPU, wie auch der in Verbindung verfügbare Hauptspeicher bzw. die Anzahl der CPUs. Das Zugriffskriterium kann z.B. definiert werden als 'Maximal im 2. Layer' oder 'pyhsikalischer Zugriff erforderlich'.

Neben den Anwendungsspezifischen Kriterien sind ebenso DV bezogene Kriterien erforderlich. So ist es sinnvoll und möglich bei erforderlichen Wartungsarbeiten aktive lang-laufende Anwendungen auf alternative ausführende Knoten zu verschieben.

Die verschachtelte vertikale Hierarchie bildet eine Containment-Beziehung ab, die insbesondere die gebündelte Verteilung und Relokalisierung unterstützt. Mit Mitteln der Hypervisor ist es ebenso möglich eine dynamische Neuverteilung vorzunehmen.

Dieser Ansatz im USM unterstützt das vereinfachte bedarfsweise Starten und das geordnete Ausschalten bzw. Einfrieren von kompletten VM-Stacks einschließlich zugehöriger physikalischer Host-Maschinen. So ist es z.B. möglich eine Client-Server-Landschaft mit geringem Aufwand in eine Container-Maschine zu konsolidieren und in einem rechnerübergreifend synchronisierten Zustand 'einzufrieren'. Dadurch können diese persistent gespeichert und bedarfsweise als Gruppe gestartet werden.

Dies kann zur schrittweisen dynamischen Annährung an die Ziel-Kapazität in Ausgelasteten Grids, ebenso zur temporären Reduktion und Relokalisierung von Ausführungskapazitäten eingesetzt werden.

3.1 Performance von Virtuellen Stacks

Die Ausführung von verschachtelten virtuellen Maschinen ist bei den gegenwärtig vorhandenen Systemen nur durch Emulation möglich, bei Unterstützung auf Betriebssystemebene und Hypervisor erfolgt eine flache Verteilung der VMs des hierarchischen VM-Stacks auf die lokalen und evtl. entfernte Prozessorkerne, so daß eine Emulation entfällt und diese parallel ausgeführt werden können. Dadurch werden virtuelle Maschinen mit logisch vertikaler Baum-Struktur in flachen Prozessor-Arrays ausgeführt. Somit ist auch einen Verschachtelung über mehrere Ebenen mit vernachlässigbaren Performance-Einbußen möglich.

Eine Emulation bleibt jedoch i.A. erforderlich, wenn Host lokale - von der Host-CPU abweichende - Architekturen erforderlich sind. Alternativ ist es jedoch auch möglich eine HW-Componente entsprechender Architektur in den VM-STack zu integrieren und den entsprechenden Teilbaum auf einem logisch entfernten Rechner durch den USM zu verwalten.

3.2 Vorteile Virtueller Stacks

Der Vorteil dieses generischen Ansatzes für die Definition von zusammengesetzten Software-Modulen aus virtuellen Maschinen ist insbesondere auch die Reduktion der Systemvoraussetzungen ausschließlich auf die Kompatibilität des Hypervisors. Dies gilt auch für zusammengesetzte Sub-Cluster in einem größeren Verbund.

Daraus resultiert auch die vereinfachte Verwaltbarkeit zusammengesetzter Gruppen, ebenso das weitestgehend generische status-konsistente Einfrieren von Rechner-Gruppen mit beliebigen Anwendungen. Desweiteren resultiert die einfache Sicherung von Rechner-Verbünden.

Im Bereich der Simulation und umfangreicher Berechnungen wird eine flexible Skalierung erforderlicher Rechenleistung durch Hinzubuchen von internen und externen Kapazitäten erreicht. Diese können durch dynamische Verlagerung von Instanzen auch ohne spezielle Unterstützung durch die Anwendung dynamisch rekonfiguriert werden.

Auch heutige Systeme mit 2-3Ebenen nutzen die Vorteile von virtuellen Stacks. Beispielsweise ist die Nutzung von CPU-Emulatoren formal immer auch mit einem virtuellen Stack verbunden, dies ermöglicht im Bereich der Entwicklung eingebetteter Systeme ohne harte Echtzeitanforderung die Bereitstellung kostengünstiger Test- und Entwicklungsumgebungen.

Der Vorteil im Bereich der Test- und Entwicklungssysteme ist insbesondere die einfache Bereitstellung von komplexen Systemen mit konsistenten Komponenten, ebenso die einfache automatisierte Erfassung, Verwaltung und Sicherung.

4 Anwendungsbeispiele

Die folgenden ausführliche Beispiele und Use-Cases sind(werden zeitnah) unter *UnifiedSessionsManager.org* [1, USM] bereitgestellt.

- Dynamische Desktop-Konfiguration
- Einsatz der automatisierten Inventarisierung
- Entwicklungsumgebungen mit Emacs
- Entwicklungsumgebungen mit Eclipse
- Entwicklungsumgebungen mit MS-VisualStudio
- Virtualisierte Build Systeme
- Schaltungs und PCB-Entwicklungsumgebungen mit EDWin
- Virtualisierte SPICE-Simulation
- Virtualisierte Scilab Umgebung
- Aufbau einer GPU-Umgebung mit CUDA
- Integration von Amazon-EC2
- Integration von Amazon-EC2 mit GPU
- Integration von Google App Engine

Weitere Beispiele sind unter *UnifiedSessionsManager.org* [1, USM] und *ctys-doc* [3, Sourceforge-ctys-doc] in dem User-Manual und HowTo vorhanden.

Literatur

[1] UnifiedSessionsManager: http://www.UnifiedSessionsManager.org/

[2] UnifiedSessionsManager: http://sourceforge.net/projects/ctys Seit 02/2008.

[3] UnifiedSessionsManager-doc: http://sourceforge.net/projects/ctys-doc

[4] UnifiedSessionsManager-Benutzermanual: enthält ca. 100 Literaturhinweise http://sourceforge.net/projects/ctys-doc