White paper
Fujitsu Software openUTM Go2Cluster

Gerade dann, wenn Anwendungen in unternehmenskritischen Bereichen eingesetzt werden, können Ausfallzeiten nicht toleriert werden. Entsprechend dem Paradigma *dynamischer Infrastrukturen* unterstützt Fujitsu Software openUTM diese Hochverfügbarkeit durch die UTM Cluster Funktionalität.

Inhalt

Einleitung

Gerade dann, wenn Anwendungen in unternehmenskritischen Bereichen eingesetzt werden, können Ausfallzeiten nicht toleriert werden. Hochverfügbarkeit der Anwendungen sowie zu Hochlastzeiten die Möglichkeit, die anfallende Last automatisch zu verteilen, haben höchste Priorität und sind ein wesentliches Merkmal dynamischer Infrastrukturen. Darüber hinaus werden Anwendungen in zunehmendem Maße mit einem Web-Auftritt ausgerüstet, der eine 7x24 Stunden Verfügbarkeit erforderlich macht. Wie in dem White Paper *Fit4Cluster* dargestellt, ist die Cluster-Unterstützung von openUTM eine Antwort auf diese Anforderungen.
In dem vorliegenden White Paper *Go2Cluster* wird anhand verschiedener Beispiel-Szenarien im Detail erläutert, wie eine stand-alone Anwendung zu einer UTM-Cluster-Anwendung umgerüstet werden kann. Es richtet sich an Entwickler und Administratoren, die die Umstellung vornehmen wollen und soll ihnen durch eine kompakte beispielhafte Darstellung das mühsamere Zusammensuchen der erforderlichen Schritte aus diversen Manualen ersparen. Bei der Erläuterung der Schritte wird aber auf die jeweiligen Manual-Stellen verwiesen, wo weitergehende Informationen nachzulesen sind.
Jedes Beispiel beginnt mit einer Darstellung der als Ausgangssituation vorliegenden stand-alone Anwendung. Dann folgt die detaillierte Erläuterung der für die Umstellung erforderlichen Schritte. Abschließend werden in einigen Fällen noch Varianten des jeweiligen Beispiels diskutiert.
Im Anschluss an die Beispiele wird auf Performance-Überlegungen und -Tuning eingegangen. Insbesondere werden dort Hinweise über sinnvolle Konfigurations-Einstellungen im Zusammenspiel mit dem XCS-Verbund bei BS2000 gegeben.

Referenzen

Die Erläuterungen zu den Beispielen verweisen auf folgende Manuale:

[1] openUTM V6.3 – Anwendungen generieren: http://manuals.ts.fujitsu.com
[2] openUTM-Client V6.3 für Trägersystem UPIC: http://manuals.ts.fujitsu.com
[3] openUTM V6.3 – Einsatz von openUTM-Anwendungen unter BS2000: http://manuals.ts.fujitsu.com
[4] openUTM V6.3 - Einsatz von openUTM-Anwendungen unter Unix- und Windows-Systemen: http://manuals.ts.fujitsu.com
[5] SESAM/SQL-Server V8.0A – Datenbankbetrieb: http://manuals.ts.fujitsu.com
[6] HIPLEX MSCF – BS2000 Rechner im Verbund – Benutzerhandbuch: http://manuals.ts.fujitsu.com
[7] Oracle® Database Installation and Administration Guide 10g for Fujitsu BS2000: http://download.oracle.com/docs/cd/B19306_01/install.102/e10319/toc.htm
[8] Oracle® Database Net Services References: http://download.oracle.com/docs/cd/B19306_01/network.102/b14213/toc.htm
[9] ipvsadm(8) - Linux man page: http://linux.die.net/man/8/ipvsadm
[10] LVS Documentation - Virtual Server via NAT http://www.linuxvirtualserver.org/VS-NAT.html

Erstes Beispiel: UPIC Clients – UTM-Anwendung – SESAM

Ausgangssituation: Stand-alone Anwendung mit SESAM/SQL

Die in diesem Beispiel betrachtete UTM-Anwendung läuft auf einem Mainframe `SERV1` mit BS2000 als Betriebssystem, ggf. als Gastsystem unter VM2000. Sie verwendet SESAM/SQL als Datenbanksystem, wobei der Datenbank-Handler auf dem gleichen Server läuft wie die UTM-Anwendung. Es wird angenommen, dass – wie bei SESAM möglich und auch üblich – die Anwendungsdaten nach inhaltlichen Gesichtspunkten auf mehrere Datenbanken verteilt sind, die von einem einzigen Datenbank-Handler (DBH) betrieben werden. Auf der Client-Seite liegen UPIC-Clients vor, die auf den Client-PCs unter MS Windows® laufen, siehe Abbildung 1. Es wird davon ausgegangen, dass die UTM-Anwendung über GSSB und ULS hinaus keine weiteren Vorgangs-übergreifend verwendeten Datenbereiche nutzt.

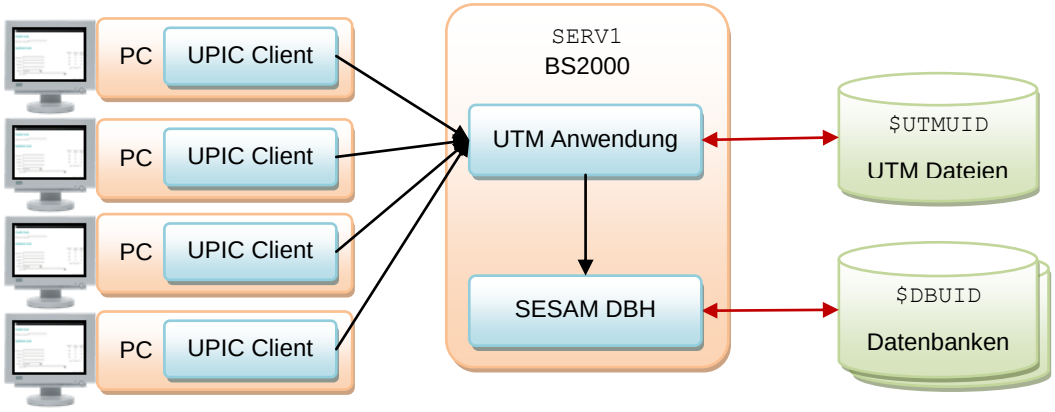

Abbildung 1: Stand-alone Anwendung

Generierung der stand-alone Anwendung

Die in Abbildung 2 aufgelisteten `KDCDEF` Steueranweisungen zur Generierung dieser UTM-Anwendung beschränken sich auf das für dieses Beispiel und die anschließend erläuterte Transformation in eine UTM-Cluster-Anwendung Wesentliche. So werden z.B. keine `USER` generiert, und auch auf die Generierung anderer Zugangskontrollen wird verzichtet.
Es werden beispielhaft nur 3 TACs generiert, ein Administrations-TAC, der mit `KDCWADMI` verknüpft ist und für die Administration durch WinAdmin genutzt wird, und zwei Aktions-TACs, die mit vom Anwender erstellten Teilprogrammen `ACTION1` bzw. `ACTION2` verknüpft sind. Als Datenbanksystem wird SESAM/SQL gewählt. Es wird ein Pool für bis zu 100 UPIC-Clients generiert. Details zu diesen Steueranweisungen können in [1], Kapitel 6 nachgelesen werden.

```
OPTION     GEN = ALL, ROOTSRC = SRC.KDCROOT1
ROOT       KDCROOT1
MAX        KDCFILE     = $UTMUID.SAMPLE1
MAX        APPLINAME   = SAMPLE1
MAX        TASKS       = 4
TAC        KDCWADMI,   PROGRAM = KDCWADMI, ADMIN = YES,
TAC        ACT1TAC,    PROGRAM = ACTION1
TAC        ACT2TAC,    PROGRAM = ACTION2
PROGRAM    KDCWADMI,   COMP = ILCS
PROGRAM    ACTION1,    COMP = ILCS
PROGRAM    ACTION2,    COMP = ILCS
DATABASE   TYPE = SESAM, ENTRY = SESSQL, LIB = LOGICAL-ID(SYSLNK)
TPOOL      BCAMAPPL = SAMPUPIC, PRONAM = *ANY, PTYPE = UPIC-R,          -
           LTERM    = TPOOL#    NUMBER = 100
```

Abbildung 2: KDCDEF Steueranweisungen

Beim Ausführen von KDCDEF mit diesen Steueranweisungen werden die UTM-Dateien unter der Benutzerkennung `$UTMUID` mit Präfix `SAMPLE1` generiert.

Generieren der Datenbank

Die SESAM/SQL-Datenbank wird mit den in Abbildung 3 aufgelisteten Steueranweisungen konfiguriert, die sich wiederum auf das für das vorliegende Beispiel Wesentliche beschränken. In Anlehnung an den Namen der UTM-Anwendung `SAMPLE1` wird `DBH-NAME = 1` gewählt; als Konfigurationsname wird V gewählt. Es wird von 4 Datenbanken (`SQLDB1, SQLDB2, SQLDB3, SQLDB4`) ausgegangen. Entsprechend den 4 UTM-Tasks werden 4 Datenbank-Threads konfiguriert. Details zu diesen Steueranweisungen können in [5], Kapitel 3 nachgelesen werden.

```
//SET-DBH-OPTIONS                                                    -
//  DBH-IDENTIFICATION = *PARAMETERS (CONFIGURATION-NAME = V,         -
//                                   DBH-NAME           = 1),        -
//  SYSTEM-LIMITS = *PARAMETERS (THREADS = 4)
//ADD-SQL-DATABASE-CATALOG-LIST                                      -
//  ENTRY-1 = *CATALOG (CATALOG-NAME = SQLDB1, USER-ID = DBUID)
//ADD-SQL-DATABASE-CATALOG-LIST                                      -
//  ENTRY-2 = *CATALOG (CATALOG-NAME = SQLDB2, USER-ID = DBUID)
//ADD-SQL-DATABASE-CATALOG-LIST                                      -
//  ENTRY-3 = *CATALOG (CATALOG-NAME = SQLDB3, USER-ID = DBUID)
//ADD-SQL-DATABASE-CATALOG-LIST                                      -
//  ENTRY-4 = *CATALOG (CATALOG-NAME = SQLDB4, USER-ID = DBUID)
```

Abbildung 3: Steueranweisungen zur Konfiguration der SESAM-Datenbanken

Beim Ausführen dieser Steueranweisungen werden die Datenbank-Dateien unter der Benutzerkennung `$DBUID` erwartet.

Starten der stand-alone Anwendung

Nachdem die Datenbank eingerichtet und der Database-Handler hochgefahren ist, kann die stand-alone Anwendung gestartet werden. Die beim Starten angegebenen Startparameter sind in Abbildung 4 aufgelistet. Details zu den Startparametern können in [3], Kapitel 5.1.1 nachgelesen werden.

```
.UTM START FILEBASE  = $UTMUID.SAMPLE1
.UTM START STARTNAME = $UTMUID.SAMPLE1.ENTER
.UTM START DB-CONNECT-TIME = 30
.UTM END
END
```

Abbildung 4: openUTM Start-Parameter

UPIC-Clients

Die UPIC-Clients nutzen die in Abbildung 5 dargestellte *Side Information Datei* (`upicfile`). Es wird in diesem Beispiel davon ausgegangen, dass die Werte für `HOSTNAME` und `TSEL` im Client-Programm nicht mittels `SET`-Anweisungen modifiziert werden. Details über den Aufbau der Einträge in der `upicfile` können in [2], Kapitel 6.2 nachgelesen werden.

```
HDACT10001 SAMPLE1.SERV1 ACT1TAC HOSTNAME=SERV1
HDACT20001 SAMPLE1.SERV1 ACT2TAC HOSTNAME=SERV1
```

Abbildung 5: `upicfile`

Umstellung auf eine UTM-Cluster-Anwendung

Angestrebte UTM-Cluster-Anwendung

Es soll ein zweiter Mainframe-Server `SERV2` mit BS2000 als Betriebssystem hinzugenommen werden, der ggf. auch parallel zu `SERV1` als Gastsystem unter VM2000 etabliert sein kann, siehe Abbildung 6. Im Folgenden wird einheitlich der Begriff *System* (statt *Server*) verwendet, um beide Fälle gleichermaßen abzudecken. Die stand-alone Anwendung soll in eine UTM-Cluster-Anwendung mit `SERV1` und `SERV2` als Knoten umgerüstet werden. Der Pubset, auf dem die Kennung `$UTMUID` zur Ablage der UTM-Dateien liegt, wird zu einen XCS-Pubset gemacht.
Die vier Datenbanken werden auf die beiden Systeme verteilt. Auf jedem der beiden Systeme wird ein DBH und zusätzlich eine Verteil-Komponente DCN eingerichtet, so dass von jedem System auch auf die Datenbanken des anderen Systems zugegriffen werden kann.
Die UPIC-Clients sollen eine Lastverteilung durchführen und Verbindungen abwechselnd zu `SERV1` und zu `SERV2` aufbauen, wobei angenommen wird, dass `SERV1` leistungsstärker ist und daher häufiger ausgewählt werden soll als `SERV2`.

Aufbau der Hardware-Konfiguration

Zunächst muss das zweite BS2000 System `SERV2` etabliert werden, wie schon gesagt, entweder als ein weiterer Server oder als weiteres Gastsystem unter VM2000 auf dem bestehenden Server.
Auf beiden Systemen ist nun zusätzlich HIPLEX-MSCF (V4.0 oder höher) zu installieren. Für die Cluster-Funktionalität nutzt openUTM das dann zur Verfügung stehende *Distributed Lock Management* (DLM). Details zur Installation von HIPLEX-MSCF können in [6], Kapitel 5 nachgelesen werden.

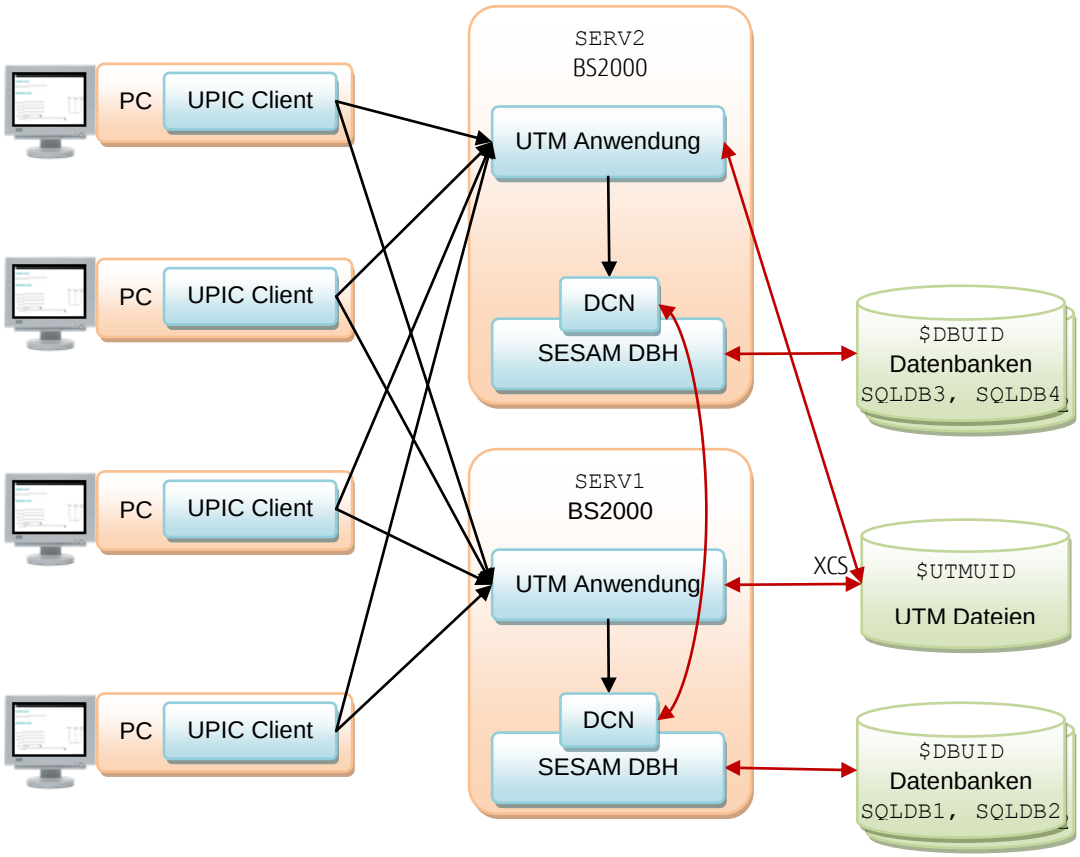

Abbildung 6: UTM-Cluster-Anwendung

Ein XCS-Verbund mit Namen `CLUST`, bestehend aus den Systemen `SERV1` und `SERV2`, wird mit der in Abbildung 7 dargestellten MSCF-Konfigurationsdatei konfiguriert. Dabei beschränken wir uns wieder auf das für den Betrieb des UTM-Clusters Wesentliche; so wird z.B. auf die Festlegung von Passwörtern verzichtet und bei vielen Parametern der Standard-Wert gewählt. Details zu den Konfigurations-Kommandos können in [6], Kapiteln 5.2.4 und 9 nachgelesen werden.

```
/SET-MSCF-ENVIRONMENT  XCS-NAME         = CLUST
/START-MSCF-CONNECTION PROCESSOR-NAME   = SERV1,                    -
                       CONNECTION-TYPE  = *CLOSELY-COUPLED
/START-MSCF-CONNECTION PROCESSOR-NAME = SERV2,                      -
                       CONNECTION-TYPE  = *CLOSELY-COUPLED
```

Abbildung 7: MSCF Konfigurationsdatei

Nun wird das XCS Pubset mit dem in Abbildung 8 aufgeführten Kommando konfiguriert. Der bereits von der stand-alone Anwendung genutzte Pubset `UTMC` soll weiterhin genutzt werden. Daher wird das Kommando `MODIFY-MASTER-CATALOG-ENTRY` ausgeführt. Details zu diesem Kommando sowie der notwendigen Einstellung des BS2000-Systemparameters `MCXSPXCS` können in [6], Kapitel 5.6 nachgelesen werden. Allgemeine Informationen zum XCS-Verbund sind in Kapitel 6.7 nachzulesen.

```
/MODIFY-MASTER-CATALOG-ENTRY ENTRY-NAME        = UTMC,              -
/                            SHARED-PUBSET     = *YES,              -
/                            XCS-CONFIGURATION = *YES
/SET-PUBSET-ATTRIBUTES       PUBSET            = UTMC,              -
/                            SHARE             = *YES,              -
/                            MASTER            = *NONE,             -
/                            BACKUP-MASTER     = *NONE,             -
```

Abbildung 8: Eintragen der Shared Pubset in den MRSCAT

Generierung der UTM-Cluster-Anwendung

Die Umstellung der stand-alone Anwendung auf eine UTM-Cluster-Anwendung erfolgt nach den in [3], Kapitel 7.11.1 beschriebenen Schritten. Zunächst werden die Steueranweisungen für KDCDEF angepasst, siehe Abbildung 9 (Änderungen ggü. den Steueranweisungen für die ursprüngliche stand-alone Anwendung sind in blau dargestellt).

Als Cluster-Filebase und auch als User-Filebase wird `$UTMUID.CLUST.SAMPLE1` eingestellt. Neben den beiden Knoten `SERV1` und `SERV2` wird vorsorglich noch ein Reserve-Knoten `RESRV` generiert, um für spätere Hochrüstungen gewappnet zu sein. Auf den Operanden `FAILURE-CMD` wird bei der Erläuterung des automatischen Warmstarts eingegangen, auf die Operanden `RESTART-TIMER-SEC` und `EMERGENCY-CMD` wird später bei Erläuterungen der Umschaltung der Datenbanken bei Ausfall eines Knotens eingegangen. Die optionalen fünften und sechsten Prozedurparameter (`PROC-PAR`) wurden mit openUTM V6.2 eingeführt.

In den `CLUSTER-NODE`-Anweisungen wird ein logischer Knotenname definiert, was ab openUTM V6.2 möglich ist. Dies erleichtert späteren Austausch der zugrunde liegenden Server, denn dabei kann der logische Knotenname unverändert beibehalten werden, auch wenn sich der `HOSTNAME` ändern sollte. Bei Verzicht auf die optionale Angabe von `NODE-NAME` können die Steueranweisungen auch mit openUTM V6.1 verwendet werden.

```
OPTION     GEN = (CLUSTER, KDCFILE, ROOTSRC), ROOTSRC = KDCROOT1
ROOT       KDCROOT1
CLUSTER    CLUSTER-FILEBASE  = $UTMUID.CLUST.SAMPLE1,                    -
           BCAMAPPL          = UTMCPING,                                 -
           USER-FILEBASE     = $UTMUID.CLUST.SAMPLE1,                    -
           RESTART-TIMER-SEC = 180,                                      -
           FAILURE-CMD       =                                           -
         'FROM-FILE = $UTMUID.FAILURE, PROC-PAR = (%s,%s,%s,%s,%s,%s)', -
           EMERGENCY-CMD     =                                           -
         'FROM-FILE = $UTMUID.EMERGENCY, PROC-PAR = (%s,%s,%s,%s,%s,%s)'
CLUSTER-NODE FILEBASE  = $UTMUID.SERV1.SAMPLE1, HOSTNAME = SERV1,       -
             NODE-NAME = NODE1
CLUSTER-NODE FILEBASE  = $UTMUID.SERV2.SAMPLE1, HOSTNAME = SERV2,       -
             NODE-NAME = NODE2
CLUSTER-NODE FILEBASE  = $UTMUID.RESRV.SAMPLE1, HOSTNAME = RESRV,       -
             NODE-NAME = NODE3
MAX        KDCFILE    = $UTMUID.CLUST.SAMPLE1
MAX        APPLINAME  = SAMPLE1
MAX        TASKS      = 4
TAC        KDCWADMI,  PROGRAM = KDCWADMI, ADMIN = YES
TAC        ACT1TAC,   PROGRAM = ACTION1
TAC        ACT2TAC,   PROGRAM = ACTION2
PROGRAM    KDCWADMI,  COMP = ILCS
PROGRAM    ACTION1,   COMP = ILCS
PROGRAM    ACTION2,   COMP = ILCS
```

Abbildung 9: Modifizierte KDCDEF Steueranweisungen

Beim Ausführen von KDCDEF mit diesen Steueranweisungen wird eine initiale `KDCFILE $UTMUID.CLUST.SAMPLE1.KDCA` generiert. Diese Datei muss nun auf Knoten-spezifische Dateien kopiert werden, z.B. mit den in Abbildung 10 aufgeführten Kommandos.

Da die neuen KDCFILES einen anderen Namen als die bisherige KDCFILE der stand-alone Anwendung haben, bleibt die bisherige KDCFILE für einen KDCUPD-Lauf erhalten.

```
/COPY-FILE FROM-FILE = $UTMUID.CLUST.SAMPLE1.KDCA,                      -
           TO-FILE   = $UTMUID.SERV1.SAMPLE1.KDCA
/COPY-FILE FROM-FILE = $UTMUID.CLUST.SAMPLE1.KDCA,                      -
           TO-FILE   = $UTMUID.SERV2.SAMPLE1.KDCA
```

Abbildung 10: Kopieren der KDCFILE

Nun wird die stand-alone Anwendung normal beendet. Wir wählen den Knoten `SERV1` aus, um mit einem KDCUPD-Lauf Benutzerdaten (z.B. Kennworte) und noch nicht ausgeführte Hintergrund-Aufträge und Asynchron-Vorgänge von der stand-alone Anwendung in die Cluster-Anwendung zu übertragen. In Abbildung 11 sind die Steueranweisungen für diesen KDCUPD-Lauf angegeben. Details zu den KDCUPD-Steueranweisungen können in [1], Kapitel 8.3 nachgelesen werden.

```
KDCFILE  NEW = $UTMUID.SERV1.SAMPLE1.KDCA,                              -
         OLD = $UTMUID.SAMPLE1.KDCA
TRANSFER
END
```

Abbildung 11: Steueranweisungen für KDCUPD

Automatischer Warmstart einer ausgefallenen Knoten-Anwendung

Um Hochverfügbarkeit zu gewährleisten sollte der Warmstart einer ausgefallenen Knotenanwendung automatisiert werden. Dies wird dadurch erreicht, dass eine Failure-Prozedur `$UTMUID.FAILURE` konfiguriert ist, siehe Abbildung 9, die bei Ausfall einer Knoten-Anwendung von der Cluster-Ringüberwachung automatisch aufgerufen wird. Ein Beispiel einer solchen Failure-Prozedur ist in Abbildung 12 dargestellt. Die Prozedur geht davon aus, dass die Anwendung mit Job-Variablen-Überwachung gestartet wurde. Die Parameter `NODENAME` und `TRMAREASON` stehen erst ab openUTM V6.2 zur Verfügung.

```
/BEGIN-PARAMETER-DECLARATION
/   DECLARE-PARAMETER NAME = APPLICATIONNAME
/   DECLARE-PARAMETER NAME = FILEBASE
/   DECLARE-PARAMETER NAME = HOSTNAME
/   DECLARE-PARAMETER NAME = VIRTUALHOST
/   DECLARE-PARAMETER NAME = NODENAME
/   DECLARE-PARAMETER NAME = TRMAREASON
/END-PARAMETER-DECLARATION
/WAIT-EVENT UNTIL = *JV(CONDITION = (&FILEBASE,1,1)='T')
/ENTER-JOB FROM-FILE = &(FILEBASE).ENTER.HOST_'&(HOSTNAME)'
```

Abbildung 12: Failure Prozedur

Anpassung der Datenbank-Konfiguration

Damit auch vom Knoten `SERV2` aus auf die Datenbanken zugegriffen werden kann, müssen auf beiden Knoten `SERV1` und `SERV2` Datenbank-Handler und SESDCN Verteil-Komponenten mit den in Abbildung 14 bzw. Abbildung 16 aufgelisteten SESDCN-Steueranweisungen gestartet werden. Damit werden auf jedem Server Remote-Zugriffe auf die Datenbanken des anderen Servers eingerichtet. Es sind keine Anpassungen am Anwendungsprogramm erforderlich. Die in Abbildung 3 aufgeführten Steueranweisungen zur Konfiguration der Datenbanken werden auf die beiden Server aufgeteilt, wobei auf `SERV1` die Datenbanken `SQLDB1` und `SQLDB2` und auf `SERV2` die Datenbanken `SQLDB3` und `SQLDB4` eingerichtet werden, siehe Abbildung 13 bzw. Abbildung 15. Details zu den SESDCN-Steueranweisungen können in [5], Kapitel 4.4 nachgelesen werden.

```
//SET-DBH-OPTIONS                                                       -
//  DBH-IDENTIFICATION = *PARAMETERS (CONFIGURATION-NAME = V,            -
//                                   DBH-NAME           = 1),           -
//  SYSTEM-LIMITS = *PARAMETERS (THREADS = 4)
//ADD-SQL-DATABASE-CATALOG-LIST                                         -
//  ENTRY-1 = *CATALOG (CATALOG-NAME = SQLDB1, USER-ID = DBUID)
//ADD-SQL-DATABASE-CATALOG-LIST                                         -
//  ENTRY-2 = *CATALOG (CATALOG-NAME = SQLDB2, USER-ID = DBUID)
//END
```

Abbildung 13: Steueranweisungen für die Konfiguration der SESAM-Datenbanken auf `SERV1`

```
//SET-DCN-OPTIONS                                                    -
//  DCN-IDENTIFICATION = *PARAMETERS                                 -
//    (CONFIGURATION-NAME = V,                                       -
//     DCN-NAME           = D)
//ADD-DISTRIBUTION-RULE-LIST                                         -
//    CATALOG-NAME-1 = SQLDB1 (LINK-NAME = LOCAL1,  DBH-NAME = 1)
//ADD-DISTRIBUTION-RULE-LIST                                         -
//    CATALOG-NAME-1 = SQLDB2 (LINK-NAME = LOCAL1,  DBH-NAME = 1)
//ADD-DISTRIBUTION-RULE-LIST                                         -
//    CATALOG-NAME-1 = SQLDB3 (LINK-NAME = REMOTE2, DBH-NAME = 2)
//ADD-DISTRIBUTION-RULE-LIST                                         -
//    CATALOG-NAME-1 = SQLDB4 (LINK-NAME = REMOTE2, DBH-NAME = 2)
//ADD-NETWORK-LINK-LIST                                              -
//  LINK-NAME-1 = REMOTE2 (PROCESSOR-NAME     = SERV2,               -
//                         CONFIGURATION-NAME = V,                   -
//                         DCN-NAME           = D)
//ADD-NETWORK-LINK-LIST                                              -
//  LINK-NAME-2 = LOCAL1  (PROCESSOR-NAME     = SERV1,               -
```

Abbildung 14: DCN Steueranweisungen für SERV1

```
//SET-DBH-OPTIONS                                                    -
//  DBH-IDENTIFICATION = *PARAMETERS (CONFIGURATION-NAME = V,         -
//                                   DBH-NAME           = 1),        -
//  SYSTEM-LIMITS = *PARAMETERS (THREADS = 4)
//ADD-SQL-DATABASE-CATALOG-LIST                                      -
//  ENTRY-3 = *CATALOG (CATALOG-NAME = SQLDB3, USER-ID = DBUID)
//ADD-SQL-DATABASE-CATALOG-LIST                                      -
//  ENTRY-4 = *CATALOG (CATALOG-NAME = SQLDB4, USER-ID = DBUID)
//END
```

Abbildung 15: Steueranweisungen für die Konfiguration der SESAM-Datenbanken auf SERV2

```
//SET-DCN-OPTIONS                                                    -
//  DCN-IDENTIFICATION = *PARAMETERS                                 -
//    (CONFIGURATION-NAME = V,                                       -
//     DCN-NAME           = D)
//ADD-DISTRIBUTION-RULE-LIST                                         -
//    CATALOG-NAME-1 = SQLDB1 (LINK-NAME = REMOTE1, DBH-NAME = 1)
//ADD-DISTRIBUTION-RULE-LIST                                         -
//    CATALOG-NAME-1 = SQLDB2 (LINK-NAME = REMOTE1, DBH-NAME = 1)
//ADD-DISTRIBUTION-RULE-LIST                                         -
//    CATALOG-NAME-1 = SQLDB3 (LINK-NAME = LOCAL2,  DBH-NAME = 2)
//ADD-DISTRIBUTION-RULE-LIST                                         -
//    CATALOG-NAME-1 = SQLDB4 (LINK-NAME = LOCAL2,  DBH-NAME = 2)
//ADD-NETWORK-LINK-LIST                                              -
//  LINK-NAME-1 = REMOTE1 (PROCESSOR-NAME     = SERV1,               -
//                         CONFIGURATION-NAME = V,                   -
//                         DCN-NAME           = D)
//ADD-NETWORK-LINK-LIST                                              -
//  LINK-NAME-2 = LOCAL2  (PROCESSOR-NAME     = SERV2                -
```

Abbildung 16: DCN Steueranweisungen für SERV2

Umschalten der Datenbanken bei Ausfall eines Cluster-Knotens

Um Hochverfügbarkeit zu gewährleisten, muss nicht nur openUTM selbst auf den Ausfall eines Knotens reagieren, sondern auch die Datenbanken. Wenn ein Knoten ausgefallen ist, sollten alle Datenbanken von dem Datenbank-Handler auf dem noch laufenden Knoten bedient werden. Dies wird dadurch erreicht, dass eine Prozedur `$UTMUID.EMERGENCY` konfiguriert ist, siehe Abbildung 9, die in diesem Fall von der

Cluster-Ringüberwachung automatisch aufgerufen wird, falls innerhalb der mit RESTART-TIMER-SEC = 180 spezifizierten 3 Minuten kein erfolgreicher Warmstart gelungen ist.
Eine derartige Emergency-Prozedur ist in Abbildung 17 dargestellt. Sie stellt die Datenbank-Konfiguration in Abhängigkeit von dem ausgefallenen Knoten entsprechend um. Angenommen SERV2 ist ausgefallen (Then-Teil des IF-Kommandos): Dann wird im ersten SESADM-Auftruf veranlasst, dass SERV1 sich nun auch um die Datenbanken SQLDB3 und SQLDB4 kümmert. Im zweiten Aufruf wird der DCN auf SERV1 umkonfiguriert, so dass für ihn alle Datenbanken lokal sind. Im dritten Aufruf wird als Vorkehrung für einen eventuellen späteren Wiederanlauf von SERV2 festgelegt, dass im DCN für SERV2 alle Datenbanken remote sind.
Die Behandlung des Falls, dass SERV1 ausgefallen ist (ELSE-Teil des IF-Kommandos) ist symmetrisch. Details zu den SESAM Administrations-Anweisungen können in [5], Abschnit 5.1.3.4 nachgelesen werden.
Die Parameter NODENAME und TRMAREASON stehen erst ab openUTM V6.2 zur Verfügung. Sie werden für die weiter unten erläuterte Knoten-Recovery benötigt. Ebenso wird /SET-PROCEDURE-OPTIONS nur für die Knoten-Recovery benötigt.

```
/SET-PROCEDURE-OPTIONS DATA-ESC = STD
/BEGIN-PARAMETER-DECLARATION
/   DECLARE-PARAMETER NAME = APPLICATIONNAME
/   DECLARE-PARAMETER NAME = FILEBASE
/   DECLARE-PARAMETER NAME = HOSTNAME
/   DECLARE-PARAMETER NAME = VIRTUALHOST
/   DECLARE-PARAMETER NAME = NODENAME
/   DECLARE-PARAMETER NAME = TRMAREASON
/END-PARAMETER-DECLARATION
/IF ('&HOSTNAME' EQ 'SERV2')
/   START-SESADM
//     START-DBH-ADMINISTRATION PASSWORD = 'SesAdmPw', DBH-NAME = 1,   -
//         CONFIGURATION-NAME = V, HOST-NAME = SERV1
//     ADD-SQL-DB-CATALOG-ENTRY CATALOG-NAME = SQLDB3, USER-ID = DBUID
//     ADD-SQL-DB-CATALOG-ENTRY CATALOG-NAME = SQLDB4, USER-ID = DBUID
//  END
/   START-SESADM
//     START-DCN-ADMINISTRATION PASSWORD = 'SesAdmPw', DCN-NAME = D,   -
//         CONFIGURATION-NAME = V, HOST-NAME = SERV1
//     MODIFY-DISTRIBUTION-RULE-ENTRY HOST-NAME=SERV2, NEW-NAME=SERV1
//  END
/   START-SESADM
//     START-DCN-ADMINISTRATION PASSWORD = 'SesAdmPw', DCN-NAME = D,   -
//         CONFIGURATION-NAME = V, HOST-NAME = SERV2
//     MODIFY-DISTRIBUTION-RULE-ENTRY HOST-NAME=SERV2, NEW-NAME=SERV1
//  END
/ELSE
/   START-SESADM
//     START-DBH-ADMINISTRATION PASSWORD = 'SesAdmPw', DBH-NAME = 2,   -
//         CONFIGURATION-NAME = V, HOST-NAME = SERV2
//     ADD-SQL-DB-CATALOG-ENTRY CATALOG-NAME = SQLDB1, USER-ID = DBUID
//     ADD-SQL-DB-CATALOG-ENTRY CATALOG-NAME = SQLDB2, USER-ID = DBUID
//  END
/   START-SESADM
//     START-DCN-ADMINISTRATION PASSWORD = 'SesAdmPw', DCN-NAME = D,   -
//         CONFIGURATION-NAME = V, HOST-NAME = SERV2
//     MODIFY-DISTRIBUTION-RULE-ENTRY HOST-NAME=SERV1, NEW-NAME=SERV2
//  END
```

Abbildung 17: Emergency Prozedur

Starten der Knoten-Anwendungen auf SERV1 und SERV2
Der KDCUPD-Lauf wurde auf dem Knoten SERV1 ausgeführt. Deshalb muss die Knoten-Anwendung auf diesem Knoten zuerst gestartet werden, wobei die in Abbildung 18 aufgelisteten Startparameter genutzt werden. Sobald die Knoten-Anwendung auf SERV1 erfolgreich gestartet ist (z.B. mit WinAdmin „*Verfügbarkeit prüfen*“ festzustellen), kann die Knoten-Anwendung auf SERV2 mit den gleichen Startparametern ebenfalls gestartet werden.

```
.UTM START CLUSTER-FILEBASE = $UTMUID.CLUST.SAMPLE1
.UTM START STARTNAME = $UTMUID.SAMPLE1.ENTER
.UTM START DB-CONNECT-TIME = 60
.UTM END
END
```

Abbildung 18: Modifizierte Startparameter für die Knoten-Anwendungen

Anpassung der UPIC-Clients

Für die UPIC-Clients genügt es die `upicfile` wie in Abbildung 19 dargestellt anzupassen.
Die HD-Einträge werden durch CD-Einträge ersetzt, wobei die CD-Einträge für `SERV1` doppelt aufgeführt werden, um dem Wunsch der stärkeren Gewichtung von `SERV1` Rechnung zu tragen. `CONVERSION=IMPLICIT` bewirkt den Effekt, den man bei der stand-alone Anwendung durch den Präfix `HD` (statt `SD`) gehabt hat. Details zu den CD-Einträgen können in [2], Kapitel 6.2.2 nachgelesen werden.
Die Client-Programme können unverändert beibehalten werden und müssen weder neu übersetzt noch neu installiert werden.

```
CDACT10001 SAMPLE1.SERV1 ACT1TAC HOSTNAME=SERV1 CONVERSION=IMPLICIT
CDACT10001 SAMPLE1.SERV1 ACT1TAC HOSTNAME=SERV1 CONVERSION=IMPLICIT
CDACT10001 SAMPLE1.SERV2 ACT1TAC HOSTNAME=SERV2 CONVERSION=IMPLICIT
CDACT20001 SAMPLE1.SERV1 ACT2TAC HOSTNAME=SERV1 CONVERSION=IMPLICIT
CDACT20001 SAMPLE1.SERV1 ACT2TAC HOSTNAME=SERV1 CONVERSION=IMPLICIT
```

Abbildung 19: Modifizierter `upicfile`

Verbesserung der Hochverfügbarkeit

Automatische Knoten-Recovery

Falls ein Cluster-Knoten ausfällt und ein Warmstart nicht möglich ist, sind bereits Vorkehrungen für eine unterbrechungsfreie Verfügbarkeit der UTM-Anwendung auf den verbleibenden Knoten getroffen. In seltenen Fällen kann es jedoch vorkommen, dass zum Zeitpunkt des Ausfalls eines Cluster-Knotens dieser noch Sperren auf globale Ressourcen hält. Diese können ab openUTM V6.2 durch die Knoten-Recovery freigeben werden, siehe [3], Kapitel 7.5.6. Die Knoten-Recovery kann sowohl manuell als auch automatisch angestoßen werden.
Für eine automatische Knoten-Recovery kann im vorliegenden Beispiel die in Abbildung 17 gezeigte Emergency-Prozedur wie in Abbildung 20 dargestellt erweitert werden. Diese Ergänzung startet die UTM-Anwendung mit Startparametern für die Knoten-Recovery, wobei angenommen wird, dass die gebundene Anwendung das Element `APPL` in der Bibliothek `LIB` ist. Die Startparameter sind die gleichen wie beim normalen Start der Anwendung (siehe Abbildung 18), ergänzt um die Angabe `NODE-TO-RECOVER`, deren Wert über den Parameter `NODENAME` der Emergency-Prozedur bezogen wird. Konfigurationseinstellungen für SESAM werden in der Datei `SESAM.CONFIG` angenommen. Wahlweise kann noch `RESET-PTC = YES` angegeben werden, um auch Transaktionen im Zustand *Prepare to Commit* (PTC) zurückzusetzen. Auf die Angabe `STARTNAME` kann bei Knoten-Recovery verzichtet werden.

```
…
/ASSIGN-SYSDTA TO-FILE = *SYSCMD
/BEGIN-BLOCK PROGRAM-INPUT = *MIXED-WITH-CMD
/SET-FILE-LINK LINK-NAME = SESCONF, FILE-NAME = SESAM.CONFIG
/START-EXECUTABLE-PROGRAM FROM-FILE = *LIBRARY-ELEMENT (LIB, APPL)
.UTM START CLUSTER-FILEBASE = $UTMUID.CLUST.SAMPLE1
.UTM START NODE-TO-RECOVER  = &(NODENAME)
.UTM END
END
/END-BLOCK
```

Abbildung 20: Erweiterung der Emergency-Prozedur

Falls zum normalen Starten der UTM-Anwendung eine Startprozedur existiert, kann diese auch um einen optionalen Parameter, der statt des normalen Anwendungsstarts die Knoten-Recovery bewirkt, erweitert werden. Dann kann die so modifizierte Startprozedur mit ihrem Zusatzparameter am Ende der Emergency-Prozedur aufgerufen werden. Auf diese Weise ist es leichter, die Konsistenz zwischen den Startparametern für den normalen Anwendungsstart und den Startparametern für die Knoten-Recovery zu gewährleisten.
Falls die UTM-Anwendung mit Datenbanken interagiert, die in die Knoten-Recovery einbezogen werden sollen, sind SESAM ab V8.0, UDS ab V2.7 oder XA-Datenbanken erforderlich.

Inbetriebnahme des Reserve-Knotens

Wenn der ausgefallene Server repariert und wieder in Betrieb genommen ist, kann die Knoten-Anwendung auf diesem Server einfach wieder gestartet werden. Bei vorausgegangener Knoten-Recovery ist dies dann in der Regel ein Kaltstart der Knoten-Anwendung.

Falls jedoch der ausgefallene Server nicht (sofort) wieder in Betrieb genommen werden kann und stattdessen ein anderer Server hinzugenommen werden soll, kann der in Abbildung 9 definierte Reserve-Knoten umkonfiguriert werden.
Nehmen wir an, der neue Server heiße `SERV3`. Dann muss zunächst die initiale KDCFILE `$UTMUID.CLUST.SAMPLE1.KDCA` analog zu Abbildung 10 auch auf `$UTMUID.SERV3.SAMPLE1.KDCA` kopiert werden. Danach können mittels Administration (z.B. über WinAdmin) die Eigenschaften des bisher als `RESRV` geführten Knotens geändert werden, insbesondere als Host-Name `SERV3` eingetragen und die Filebase auf `$UTMUID.SERV3.SAMPLE1` angepasst werden. Anschließend kann die Knoten-Anwendung auf `SERV3` gestartet werden. Sie übernimmt dann automatisch alle administrativen Änderungen, die seit der Generierung der initialen KDCFILE durchgeführt wurden.

Variante dieses Beispiels: Dynamische Bestimmung von LU-Namen mittels `SET`

In einer Variante des obigen Beispiels wird davon ausgegangen, dass die Auswahl der Anwendung in den UPIC-Clients programmgesteuert erfolgt. Das kann z.B. erforderlich sein, wenn die UPIC-Clients universell sein sollen und mit verschiedenen UTM-Anwendungen kommunizieren können: neben der auf `SERV1` laufenden Anwendung SAMPLE1 auch noch mit anderen Anwendungen, die evtl. auf anderen Servern laufen.

Ausgangssituation

Es wird angenommen, dass die UPIC-Clients den jeweiligen LU-Namen dynamisch bestimmen (in Abbildung 21 durch Aufruf einer vom Client-Programm bereitgestellten Funktion `determine_LU_name` dargestellt). Die Einstellung erfolgt dann mittels `Set_Partner_LU_Name`. Dann kann auf ein `upicfile` verzichtet werden, und davon wird in diesem Beispiel Gebrauch gemacht. Ebenso wird der TAC dynamisch bestimmt (in Abbildung 21 durch Aufruf einer vom Client-Programm bereitgestellten Funktion `determine_TAC_name` dargestellt).
Abbildung 21 zeigt skizzenhaft einen Auszug aus einem solchen Client-Programm. Auf die Abfrage der Return-Codes und einer entsprechenden Fehlerbehandlung wurde der Übersichtlichkeit wegen verzichtet.

```
CONVERSATION_ID convid;
CM_RETURN_CODE  rc;
CM_INT32        len;
unsigned char   name[20];
…
Initialize_Conversation (convid, "       ", rc);
determine_LU_name (name, &len);
Set_Partner_LU_name (convid, name, &len, &rc);
determine_TAC_name (name, &len);
Set_TP_Name (convid, name, &len, &rc);
Allocate (convid, &rc);
/* send_receive_loop */
…
```

Abbildung 21: Auszug aus dem Client-Programm

Anpassung des UPIC-Clients

Als Vorbereitung für die Umstellung auf Lastverteilung für eine UTM-Cluster-Anwendung müssen `Set_Partner_LU_Name`-Aufrufe eliminiert werden. Dazu wird eine `upicfile` eingerichtet, in der der später auf Cluster umzustellende LU-Name *als Symbolic_Destination_Name* vordefiniert ist, siehe Abbildung 22. Diese `upicfile` muss dann auf alle Clients verteilt werden.
Das Client-Programm wurde entsprechend angepasst, so dass die Auswahl der anzusprechenden UTM-Cluster-Anwendung für diesen LU-Namen über `Initialize_Conversation` mit Bezug auf den *Symbolic_Destination_Name* statt über `Set_Partner_LU_Name` erfolgt. Nach dieser Umstellung kann wie oben beschrieben der UPIC-Client zu einem Lastverteiler umkonfiguriert werden. Die `Set_TP_Name`-Aufrufe können dabei beibehalten werden. Auch die `Set_Partner_LU_Name`-Aufrufefür andere als die UTM-Cluster-Anwendung können unverändert beibehalten werden.
Natürlich muss das so modifizierte Client-Programm erneut übersetzt und gebunden werden und auf allen Clients anstelle des bisherigen Client-Programms installiert werden. Dies kann vorab, vor der Umstellung auf Cluster durchgeführt und getestet werden.
In der `upicfile` wird bei der anschließenden Umstellung auf Cluster wie oben unter „Anpassung der UPIC-Clients" beschrieben die `HD`-Anweisung in `CD`-Anweisungen umgewandelt.

```
HDSAM1SRV1 SAMPLE1.SERV1 HOSTNAME=SERV1
```

```
CONVERSATION_ID convid;
CM_RETURN_CODE  rc;
CM_INT32        len;
unsigned char   name[20];
…
determine_LU_name (name, &len);
if (strcmp (name, "SAMPLE1.SERV1") == 0)
   Initialize_Conversation (convid, "SAM1SRV1", rc);
else
{
   Initialize_Conversation (convid, "       ", rc);
   Set_Partner_LU_Name (convid, name, &len, &rc);
}
determine_TAC_name (name, &len);
Set_TP_Name (convid, name, &len, &rc);
Allocate (convid, &rc);
/* send_receive_loop */
…
```

Abbildung 22: **`upicfile`** und Auszug aus dem angepassten Client-Programm

Zweites Beispiel: MT9750 Clients – UTM-Anwendung – Oracle

Ausgangssituation: Stand-alone Anwendung mit Oracle® Datenbank

Die in diesem Beispiel betrachtete UTM-Anwendung läuft auf einem Mainframe SERV1 mit BS2000 als Betriebssystem, ggf. als Gastsystem unter VM2000. Sie verwendet ein auf einen eigenen Linux-Server SERVDB ausgelagertes Oracle® Datenbanksystem (Oracle® Instanz), wobei im Folgenden die Version 10g angenommen wird. Auf der Client-Seite werden PCs mit der Terminalemulation MT9750 genutzt, siehe Abbildung 23. Die Clients sind über ein Local Area Network (LAN) mit SERV1 verbunden. In Abbildung 23 ist der letzte Router vor SERV1 explizit dargestellt, da dieser später für die Lastverteilung bei der Umstellung auf eine UTM-Cluster-Anwendung eine Rolle spielen wird. Es wird diesmal davon ausgegangen, dass die UTM-Anwendung globale Datenbereiche GSSB nutzt.

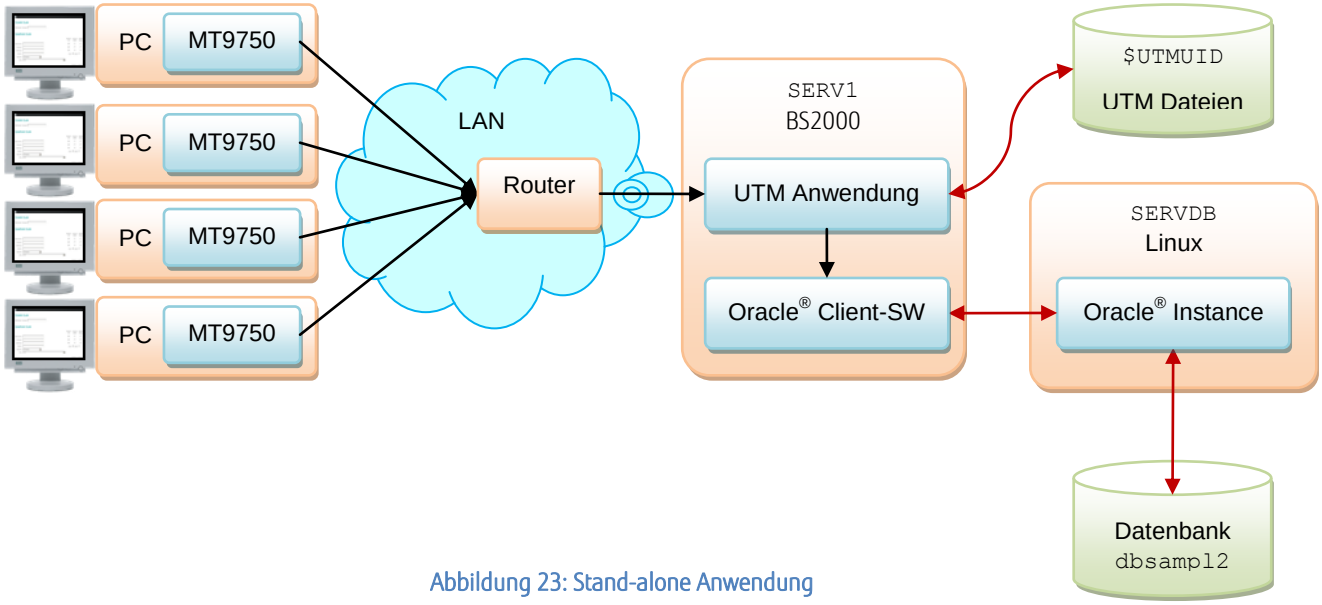

Abbildung 23: Stand-alone Anwendung

Generierung der stand-alone Anwendung

Die in Abbildung 24 aufgelisteten KDCDEF Steueranweisungen zur Generierung dieser UTM-Anwendung beschränken sich – wie schon im ersten Beispiel – auf das Wesentliche. Als Datenbank-System wurde diesmal Oracle® Version 10g gewählt und ein eigener Linux-basierter Datenbank-Server SERVDB eingesetzt. Dazu ist die Oracle® Client-Software auf dem Mainframe SERV1 unter der Kennung $ORAC1020 installiert und kommuniziert mit der auf SERVDB installierten Oracle® Instanz. Die DATABASE Steueranweisung benennt die auf dem Mainframe installierte Oracle® Client-Software, die die Verbindung zur Oracle® Instanz aufbaut, siehe auch [7], Kapitel 8. Der Zugang zur Datenbank (User und Passwort) wird durch die Generierungsinformation für die Datenbank festgelegt. Es wird angenommen, dass die Oracle® Client-Software statisch eingebunden ist.

Es wird ein Pool für bis zu 100 Terminals, die mittels MT9750 emuliert werden, generiert. Details zu diesen Steueranweisungen können in [1], Kapitel 6 nachgelesen werden.

Es wird angenommen, dass die beiden Teilprogramme ACTION1 und ACTION2 auf bis zu 10 gemeinsame GSSBs zugreifen.

```
OPTION    GEN = ALL, ROOTSRC = SRC.KDCROOT2
ROOT      KDCROOT2
MAX       KDCFILE    = $UTMUID.SAMPLE2
MAX       APPLINAME = SAMPLE2
MAX       TASKS      = 4
MAX       GSSBS      = 10
TAC       KDCWADMI,  PROGRAM = KDCWADMI, ADMIN = YES
TAC       ACT1TAC,   PROGRAM = ACTION1
TAC       ACT2TAC,   PROGRAM = ACTION2
PROGRAM   KDCWADMI,  COMP = ILCS
PROGRAM   ACTION1,   COMP = ILCS
PROGRAM   ACTION2,   COMP = ILCS
DATABASE  TYPE = XA  ENTRY = XAOSWD
```

Abbildung 24: KDCDEF Steueranweisungen

Konfiguration des Datenbanksystems

Zur Konfiguration der Oracle® Client-Software auf dem Mainframe wird die in Abbildung 25 dargestellte Konfigurationsdatei `tnsnames.ora` gewählt, in der die Verbindung zur Oracle® Instanz auf `SERVDB` als `SERVICE1` spezifiziert ist. Details zu diesen Konfigurations-Parametern können in [7], Kapitel 9 bzw. in [8], Kapitel 6 nachgelesen werden.

```
SERVICE1 =
    (DESCRIPTION =
        (ADDRESS =
            (PROTOCOL = TCP)
            (HOST      = SERVDB)
            (PORT      = 1521)
        )
        (CONNECT_DATA =
            (SERVICE_NAME  = dbsampl2.db.service)
        )
```

Abbildung 25: Oracle® Konfigurationsdatei tnsnames.ora

Starten der stand-alone Anwendung

Nachdem die Datenbank eingerichtet und hochgefahren ist, kann die stand-alone Anwendung gestartet werden. Die beim Starten angegebenen Startparameter sind in Abbildung 26 aufgelistet. Die Zugangsdaten zur Datenbank (User und Passwort) wurden bei der Generierung festgelegt, d.h. hier sind nur die Platzhalter anzugeben. Details zu den .UTM-Startparametern können in [3], Kapitel 5.1.1 und Details zum `.RMXA`-Startparameter in [7], Kapitel 8 nachgelesen werden.

```
.UTM START FILEBASE  = $UTMUID.SAMPLE2
.UTM START STARTNAME = $UTMUID.SAMPLE2.ENTER
.UTM START DB-CONNECT-TIME = 60
.UTM END
.RMXA RM="Oracle_XA",OS="Oracle_XA+Acc=P/*UTMUSER/*UTMPASS+SqlNet=SERVICE1+Se
sTm=60"
END
```

Abbildung 26: openUTM Start-Parameter

Nach dem Starten der Anwendung können sich Clients mittels der Terminal-Emulation MT9750 auf `SERV1` bei der Anwendung `SAMPLE2` anmelden.

Umstellung auf eine UTM-Cluster-Anwendung mit Oracle® RAC

Angestrebte Cluster-Architektur

Es wird – analog wie im ersten Beispiel – ein zweites Mainframe-System `SERV12` mit BS2000 als Betriebssystem hinzugenommen. Das bisherige System `SERV1` wird diesmal umbenannt in `SERV11`, und `SERV1` wird als virtueller System-Name für die Lastverteilung beibehalten, siehe Abbildung 27. Dadurch ist gewährleistet, dass Clients sich weiterhin auf `SERV1` an `SAMPLE2` anmelden können, um jetzt die UTM-Cluster-Anwendung zu erreichen. Der letzte Router vor dem ehemaligen `SERV1` wird nun zu einem Lastverteiler umkonfiguriert.
Die stand-alone Anwendung soll in eine UTM-Cluster-Anwendung mit `SERV11` und `SERV12` als Knoten umgerüstet werden. Dazu wird der Pubset, auf dem die Kennung `$UTMUID` zur Ablage der UTM-Dateien liegt, – wie schon beim ersten Beispiel erläutert (vgl. Abbildung 6) – zu einem XCS-Pubset gemacht.
Es wird auch ein zweiter Datenbank-Server `SERVDB2` mit Linux als Betriebssystem hinzugenommen, so dass die Oracle® Datenbank in einer RAC-Konfiguration betrieben werden kann. Die Datenbank `dbsampl2` muss auf einem Network File System (NFS), Oracle Cluster Files System (OCFS) oder Oracle ASM liegen, um von beiden Datenbank-Servern zugreifbar zu sein.

Generierung der UTM-Cluster-Anwendung

Die Umstellung der stand-alone Anwendung auf eine UTM-Cluster-Anwendung erfolgt – analog wie beim ersten Beispiel – nach den in [3], Kapitel 7.11.1 beschriebenen Schritten. Die angepassten Steueranweisungen sind in Abbildung 28 aufgeführt, wobei die Änderungen ggü. der Generierung der stand-alone Anwendung wieder in blau dargestellt sind. Auf `RESTART-TIMER-SEC` und `EMERGENCY-CMD` wird später im Rahmen der Lastverteilung eingegangen.
Die mit `FAILURE-CMD` spezifizierte Failure-Prozedur dient wieder dem automatischen Warmstart, wie im ersten Beispiel ausführlich erläutert. Dabei ist aber zu beachten, dass Betriebsmittel (wie z.B. in diesem Beispiel genutzte GSSB), die zum Zeitpunkt des Ausfalls einer Knoten-Anwendung gesperrt waren, bis zum Abschluss des Warmstarts gesperrt bleiben. Während es in Batch-Anwendungen in der Regel genügt, `MAX RESWAIT` entsprechend großzügig zu wählen, ist es für Dialog-Anwendungen meist geschickter, bei Erhalt des Returncodes „40Z" dem Nutzer eine Zwischenmeldung zu geben und den Versuch zu wiederholen.

Die beim Ausführen von KDCDEF entstehende initiale `KDCFILE $UTMUID.CLUST.SAMPLE2.KDCA` muss wieder auf Knoten-spezifische Dateien kopiert werden. Dies ist im Rahmen des ersten Beispiels ausführlich erläutert, weshalb hier auf Details verzichtet werden kann.

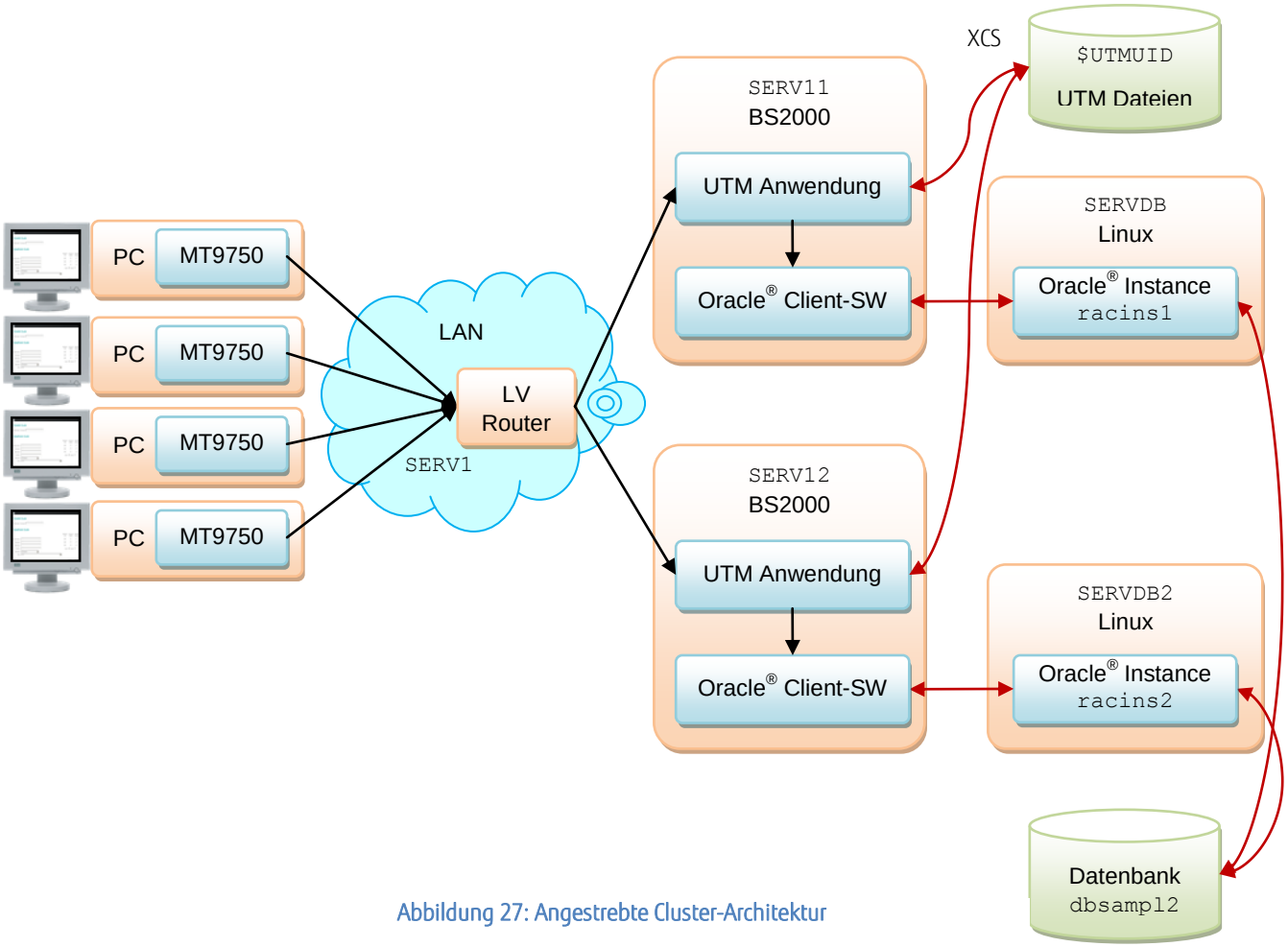

Abbildung 27: Angestrebte Cluster-Architektur

```
OPTION     GEN = (CLUSTER, KDCFILE, ROOTSRC), ROOTSRC = KDCROOT2
ROOT       KDCROOT2
CLUSTER    CLUSTER-FILEBASE  = $UTMUID.CLUST.SAMPLE2,                  -
           BCAMAPPL          = UTMCPING,                               -
           USER-FILEBASE     = $UTMUID.CLUST.SAMPLE2,                  -
           RESTART-TIMER-SEC = 180,                                    -
           FAILURE-CMD       =                                         -
               'FROM-FILE = $UTMUID.FAILURE, PROC-PAR = (%s,%s,%s,%s)'  -
           EMERGENCY-CMD     =                                         -
               'FROM-FILE = $UTMUID.EMERGENCY, PROC-PAR = (%s,%s,%s,%s)'
CLUSTER-NODE FILEBASE  = $UTMUID.SERV11.SAMPLE2, HOSTNAME = SERV11,    -
             NODE-NAME = NODE1
CLUSTER-NODE FILEBASE  = $UTMUID.SERV12.SAMPLE2, HOSTNAME = SERV12,    -
             NODE-NAME = NODE2
CLUSTER-NODE FILEBASE  = $UTMUID.RESERV.SAMPLE2, HOSTNAME = RESERV,    -
             NODE-NAME = NODE3
MAX        KDCFILE    = $UTMUID.CLUST.SAMPLE2
MAX        APPLINAME  = SAMPLE2
MAX        TASKS      = 4
MAX        GSSBS      = 10
TACCLASS   1, TASKS   = 3
TAC        KDCWADMI,  PROGRAM = KDCWADMI, ADMIN = YES
TAC        ACT1TAC,   PROGRAM = ACTION1, TACCLASS = 1
TAC        ACT2TAC,   PROGRAM = ACTION2, TACCLASS = 1
PROGRAM    KDCWADMI,  COMP = ILCS
```

Abbildung 28: Modifizierte KDCDEF Steueranweisungen

Da die beiden Teilprogramme `ACTION1` und `ACTION2` auf gemeinsame GSSBs zugreifen, muss verhindert werden, dass alle Tasks gleichzeitig auf die GSSBs zugreifen wollen und es dadurch zu einem Task-Deadlock kommen kann, siehe [3], Kapitel 7.2.2. Dazu werden die zugehörigen TACs in eine TAC-Klasse 1 gelegt, deren Task-Zahl auf 3 begrenzt wird, während die Knoten-Anwendungen jeweils mit 4 Tasks gestartet werden.

Anpassung der Datenbank-Konfiguration

Die Konfiguration der Oracle® Client-Software auf dem Mainframe wird wie in der in Abbildung 29 dargestellten Konfigurationsdatei `tnsnames.ora` gewählt, in der die Verbindungen zu den Oracle® Instanzen auf `SERVDB` und `SERVDB2` spezifiziert sind. Details zu diesen Konfigurations-Parametern können in [8], Kapitel 6 nachgelesen werden. Es werden zwei symmetrische Services `service12` und `service21` definiert.

Die in Abbildung 29 dargestellte `tnsnames.ora` Datei wird auf beiden BS2000-Servern, `SERV11` und `SERV12` oder in einem von beiden Servern zugreifbaren Shared-Pubset bereitgestellt.

```
SERVICE12 =
    (DESCRIPTION =
        (ADDRESS = (PROTOCOL = TCP) (HOST = SERVDB ) (PORT = 1521))
        (ADDRESS = (PROTOCOL = TCP) (HOST = SERVDB2) (PORT = 1521))
        (FAILOVER = on)
        (CONNECT_DATA =
            (SERVER = DEDICATED)
            (SERVICE_NAME = service12.db.service)
            (FAILOVER_MODE =
                (TYPE    = SELECT)
                (METHOD  = BASIC)
                (RETRIES = 10)
                (DELAY   = 5)
            )
         )
    )
SERVICE21 =
    (DESCRIPTION =
        (ADDRESS = (PROTOCOL = TCP) (HOST = SERVDB2) (PORT = 1521))
        (ADDRESS = (PROTOCOL = TCP) (HOST = SERVDB ) (PORT = 1521))
        (FAILOVER = on)
        (CONNECT_DATA =
            (SERVER = DEDICATED)
            (SERVICE_NAME = service21.db.service)
            (FAILOVER_MODE =
                (TYPE    = SELECT)
                (METHOD  = BASIC)
```

Abbildung 29: Oracle® Konfigurationsdatei `tnsnames.ora`

Darüber hinaus müssen auch die Konfigurationen der Oracle® Instanzen auf `SERVDB` und `SERVDB2` angepasst werden. Dabei sind `racins1` und `racins2` die jeweiligen Oracle-Instanzen auf den beiden RAC-Knoten und `dbsampl2` die von der stand-alone-Anwendung übernommene Datenbank. Dazu werden die in Abbildung 30 aufgelisteten Kommandos auf einem der beiden Linux-Server, z.B. auf `SERVDB`, ausgeführt.

```
srvctl add service -d dbsampl2 -s service12 -r racins1 -a racins2 -P BASIC
srvctl add service -d dbsampl2 -s service21 -r racins2 -a racins1 -P BASIC
srvctl start service -d dbsampl2 -s service12
srvctl start service -d dbsampl2 -s service21

sqlplus /nolog
SQL> connect sys/password as sysdba
SQL> execute dbms_service.modify_service (service_name => 'service12', dtp => true);
SQL> execute dbms_service.modify_service (service_name => 'service21', dtp => true);
```

Abbildung 30: Einrichtung der Oracle RAC Knoten

Lastverteilung

Der letzte Router vor dem ehemaligen `SERV1` wird nun zu einem Lastverteiler umkonfiguriert, wobei die mit `SERV1` assoziierte IP-Adresse zur virtuellen IP-Adresse für die Lastverteilung gemacht wird. Der ehemalige Server `SERV1` wird in `SERV11` umbenannt und erhält eine neue reale IP-Adresse.

Normalerweise ist der umzustellende Router ein handelsüblicher Router, der Lastverteilungsfunktionalität (ggf. als getrennt zu bestellende Zusatz-Funktionalität) anbietet. Um keine bestimmte Marke zu bevorzugen, demonstrieren wir hier die Lastverteilungs-Konfiguration anhand des *Linux Virtual Servers* (LVS), der diese Funktionalität ebenfalls anbietet, siehe Abbildung 31.

In der ersten `ipvsadm` Anweisung in Abbildung 31 wird `SERV1` als virtueller Service konfiguriert, wobei hier beispielhaft als Lastverteilungs-Strategie „reihum" (round robin, `rr`) ausgewählt wird. Die beiden nächsten `ipvsadm` Anweisungen konfigurieren die zu diesem Service gehörigen realen Server. Als Port wird in allen Fällen 102 gewählt. Die Option `-m` (oder `--masquerading`) entscheidet, welche Paket-Weiterleitungs-Methode und damit verbundene Adressmodifikation eingesetzt wird, für Details siehe [10]. Details zur `ipvsadm` Anweisung können in [9] nachgelesen werden.

```
ipvsadm --add-service --tcp-service SERV1:102 --scheduler rr
ipvsadm --add-server  --tcp-service SERV1:102 --real-server SERV11:102 -m
ipvsadm --add-server  --tcp-service SERV1:102 --real-server SERV12:102 -m
```

Abbildung 31: Lastverteilungs-Konfiguration für LVS

Falls ein Knoten ausfällt und auch ein automatischer Warmstart nicht gelingt, sollte dieser Knoten bei der Lastverteilung nicht mehr berücksichtigt werden. Dies wird dadurch erreicht, dass eine Prozedur `$UTMUID.EMERGENCY` konfiguriert ist, siehe Abbildung 28, die in diesem Fall von der Cluster-Ringüberwachung automatisch aufgerufen wird, falls innerhalb der mit `RESTART-TIMER-SEC = 180` spezifizierten 3 Minuten kein erfolgreicher Warmstart gelungen ist.
Eine derartige Emergency-Prozedur ist in Abbildung 32 dargestellt. Sie setzt in der Lastverteilungs-Konfiguration das Gewicht des ausgefallenen Knotens auf 0, was bewirkt, dass dieser Knoten bei neuen Verbindungsanforderungen vorerst nicht mehr berücksichtigt wird. Sobald der ausgefallene Knoten repariert und wieder hochgefahren ist, sollte der Administrator das Gewicht des betroffenen Knotens wieder auf 1 zurücksetzen.
Die in Abbildung 32 dargestellte Emergency-Prozedur nimmt an, dass `LV` der Name des Lastverteilungs-Rechners ist und dass ein Administrator-Zugang über die Kennung `lvsadm` eingerichtet ist. Es gibt nun mehrere Möglichkeiten, eine Aktion auf dem Lastverteiler-Rechner auszuführen: Nutzung von `rsh` bzw. `ssh` unter POSIX, oder Nutzung von openFT. In Abbildung 32 wird beispielhaft der Weg über `rsh` und POSIX gewählt.
Die zugehörigen `rsh`-Kommandos sind zur Vereinfachung in eigene Kommando-Dateien `LVSADM-SERV11-WEIGHT-0` und `LVSADM-SERV12-WEIGHT-0` ausgelagert.

`EMERGENCY`

```
/BEGIN-PARAMETER-DECLARATION
/   DECLARE-PARAMETER NAME = APPLICATIONNAME
/   DECLARE-PARAMETER NAME = FILEBASE
/   DECLARE-PARAMETER NAME = HOSTNAME
/   DECLARE-PARAMETER NAME = VIRTUALHOST
/END-PARAMETER-DECLARATION
/   EXEC-POSIX-CMD LVSADM-&HOSTNAME.-WEIGHT-0
```

`LVSADM-SERV11-WEIGHT-0`

```
rsh -l lvsadm LV sudo ipvsadm --edit-server --tcp-service SERV1:102 \
--real-server SERV11:102 --weight 0'
```

`LVSADM-SERV12-WEIGHT-0`

```
rsh -l lvsadm LV sudo ipvsadm --edit-server --tcp-service SERV1:102 \
--real-server SERV12:102 --weight 0'
```

Abbildung 32: Emergency-Prozedur mit rsh-Kommando-Dateien

Starten der UTM-Cluster-Anwendung

Es wird angenommen, dass ein KDCUPD-Lauf auf dem Knoten `SERV11` ausgeführt wurde. Deshalb muss die Knoten-Anwendung auf diesem Knoten zuerst gestartet werden, wobei die in Abbildung 33 aufgelisteten Startparameter genutzt werden. Sobald die Knoten-Anwendung auf `SERV11` erfolgreich gestartet ist (z.B. mit WinAdmin „*Verfügbarkeit prüfen*“ festzustellen), kann die Knoten-Anwendung auf `SERV12` mit den analogen Startparametern (mit `SERVICE21` statt `SERVICE12`) ebenfalls gestartet werden. Details zu den `.RMXA`-Startparametern für die Oracle® RAC-Konfiguration können in [4], Kapitel 7.4.2 nachgelesen werden.

```
.UTM START CLUSTER-FILEBASE = $UTMUID.CLUST.SAMPLE2
.UTM START STARTNAME = $UTMUID.SAMPLE2.ENTER
.UTM START DB-CONNECT-TIME = 60
.UTM END
.RMXA RM="Oracle_XA",OS="Oracle_XA+Acc=P/*UTMUSER/*UTMPASS+SqlNet=SERVICE12+SesTm=60",
RAC=YES
END
```

Abbildung 33: Modifizierte Startparameter für die Knoten-Anwendungen

Software-Update der Knoten im laufenden Betrieb (Online Update)

Bei UTM-Cluster-Anwendungen ist es nun auch möglich, im laufenden Betrieb ein Update der System-Software vorzunehmen und z.B. Korrekturversionen von Systemkomponenten einzuspielen. Dazu wird knotenweise vorgegangen. Wir gehen beispielsweise davon aus, dass ein Update von openUTM V6.3A00 auf die Korrekturversion V6.3A10 eingespielt werden soll.

Dazu wird zunächst mit einem der beiden Knoten, z.B. mit Knoten `SERV11` begonnen. Im Lastverteiler wird das Gewicht für diesen Knoten auf 0 gesetzt, z.B. mit einem Kommando an den Lastverteiler ähnlich wie in Abbildung 32 dargestellt. Dann wird die auf `SERV11` laufende Knoten-Anwendung mit `KDCSHUT WARN, TIME = 60, SCOPE = LOCAL` beendet. Dies bewirkt, dass Nutzer informiert werden, dass die Knoten-Anwendung in einer Stunde beendet werden wird und dass sie sich bis dahin abmelden sollen. Sie können sich nach dem Abmelden aber sofort wieder anmelden; der Lastverteiler wird sie dann automatisch mit der noch aktiven auf `SERV12` laufenden Knoten-Anwendung verbinden. Damit ist ein nahezu unterbrechungsfreier Betrieb der Cluster-Anwendung gewährleistet.

Sobald die Knoten-Anwendung auf `SERV11` beendet ist, kann auf diesem Knoten das Software-Update durchgeführt werden. Anschließend wird die Knoten-Anwendung auf `SERV11` wieder gestartet und im Lastverteiler das Gewicht für `SERV11` wieder auf 1 gesetzt. Dadurch wird `SERV11` beim Aufbau neuer Verbindungen wieder berücksichtigt. Es laufen nun vorübergehend auf den beiden Knoten unterschiedliche UTM Korrekturstände, was problemlos möglich ist, wenn diese zueinander ausreichend kompatibel sind, wie dies bei Korrekturversionen üblich ist.

Anschließend kann dann das Software-Update in gleicher Weise auf dem zweiten Knoten `SERV12` durchgeführt werden. Danach ist das Update für die gesamte Cluster-Anwendung durchgeführt und alle Knoten fahren wieder den gleichen Korrekturstand V6.3A10 von openUTM.

Drittes Beispiel: UTM-Partneranwendung auf Linux – UTM-Anwendung auf BS2000

Ausgangssituation: Stand-alone Partner-Anwendungen

In diesem dritten Beispiel wird eine verteilte Transaktionsverarbeitung mit zwei UTM-Anwendungen betrachtet. Konkret wird das im Manual [1] in Kapitel 6.7 ausführlich beschriebene Beispiel eines Reise-Buchungssystems zugrunde gelegt, wobei wir uns hier auf das Wesentliche beschränken. Es wird angenommen, dass ein Reisebüro eine Anwendung TRAVEL betreibt, die über OSI TP mit einem zentralen Buchungssystem (Reservation Management System, RMS) gekoppelt ist. RMS sei eine UTM-Anwendung auf einem BS2000 Server, TRAVEL eine UTM-Anwendung auf einem Unix-System. Die in [1] erwähnten anderen beteiligten Partner-Anwendungen und die beteiligten Datenbanksysteme werden zur Vereinfachung hier nicht betrachtet. Die so vereinfachte Situation ist in Abbildung 34 dargestellt.

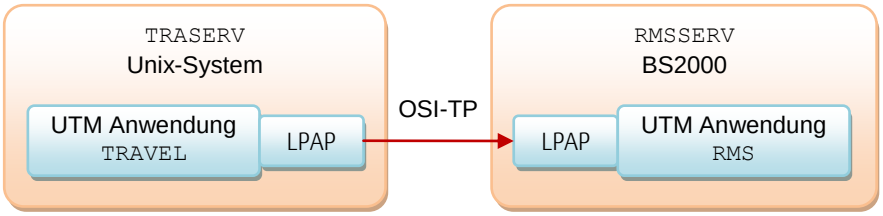

Abbildung 34: Stand-alone Partner-Anwendungen

Generierung der Stand-alone-Anwendung TRAVEL

In Abbildung 35 sind die hier wesentlichen KDCDEF-Steueranweisungen zur Generierung der UTM-Anwendung TRAVEL aufgeführt.

```
OPTION     GEN = ALL, ROOTSRC = SRC.TRAVROOT
ROOT       TRAVROOT
MAX        KDCFILE     = TRAVFILE
MAX        APPLINAME   = APTRAVEL
MAX        TASKS       = 7
TAC        KDCWADMI,   PROGRAM = KDCWADMI, ADMIN = YES
TAC        INF01,      PROGRAM = ACTION1
TAC        INF02,      PROGRAM = ACTION2
PROGRAM    KDCWADMI,   COMP = C
PROGRAM    ACTION1,    COMP = C
PROGRAM    ACTION2,    COMP = C
UTMD       APPLICATION-PROCESS-TITLE = (1, 2, 3, 21)
ABSTRACT-SYNTAX       EUROSI,                                          -
                      OBJECT-IDENTIFIER = (1, 3, 9990, 1, 3, 12)
APPLICATION-CONTEXT   EUOSICCR,                                        -
                      OBJECT-IDENTIFIER = (1, 3, 9990, 1, 4, 13),      -
                      ABSTRACT-SYSNTAX  = (EUROSI, CCR)
ACCESS-POINT TRAVEL,  TRANSPORT-SELECTOR           = C'TRAV',          -
                      SESSION-SELECTOR             = *NONE,            -
                      PRESENTATION-SELECTOR        = *NONE,            -
                      APPLICATION-ENTITY-QUALIFIER = 1,                -
                      LISTENER-PORT                = 30003
OSI-CON RMS,          LOCAL-ACCESS-POINT           = TRAVEL,           -
                      OSI-LPAP                     = RMS,              -
                      NETWORK-SELECTOR             = C'RMSSERV',       -
                      TRANSPORT-SELECTOR           = C'RMS',           -
                      SESSION-SELECTOR             = (C'SRMS', ASCII), -
                      PRESENTATION-SELECTOR        = (C'PRMS', ASCII), -
                      LISTENER-PORT                = 102
OSI-LPAP RMS,         ASSOCIATION-NAMES            = RMS,              -
```

Abbildung 35: KDCDEF Steueranweisungen für TRAVEL

In Abbildung 36 sind die wesentlichen KDCDEF-Steueranweisungen für RMS aufgeführt. Für Details siehe jeweils [1].

```
OPTION    GEN = ALL, ROOTSRC = SRC.RMSROOT
ROOT      RMSROOT
MAX       KDCFILE    = RMSFILE
MAX       APPLINAME  = APRMS
MAX       TASKS      = 10
TAC       KDCWADMI,  PROGRAM = KDCWADMI, ADMIN = YES
TAC       INF01,     PROGRAM = ACTION1
TAC       INF02,     PROGRAM = ACTION2
PROGRAM   KDCWADMI,  COMP = ILCS
PROGRAM   ACTION1,   COMP = ILCS
PROGRAM   ACTION2,   COMP = ILCS
UTMD      APPLICATION-PROCESS-TITLE = (1, 2, 3, 10)
ABSTRACT-SYNTAX      EUROSI,                                          -
                     OBJECT-IDENTIFIER = (1, 3, 9990, 1, 3, 12)
APPLICATION-CONTEXT  EUOSICCR,                                        -
                     OBJECT-IDENTIFIER = (1, 3, 9990, 1, 4, 13),      -
                     ABSTRACT-SYSNTAX  = (EUROSI, CCR)
ACCESS-POINT RMS,    TRANSPORT-SELECTOR           = C'RMS',           -
                     SESSION-SELECTOR             = (C'SRMS', ASCII), -
                     PRESENTATION-SELECTOR        = (C'PRMS', ASCII), -
                     APPLICATION-ENTITY-QUALIFIER = 1
OSI-CON TRAVEL,      LOCAL-ACCESS-POINT           = RMS,              -
                     OSI-LPAP                     = TRAVEL,           -
                     NETWORK-SELECTOR             = C'TRASERV',       -
                     TRANSPORT-SELECTOR           = C'TRAV',          -
                     SESSION-SELECTOR             = *NONE,            -
                     PRESENTATION-SELECTOR        = *NONE
OSI-LPAP TRAVEL,     ASSOCIATION-NAMES            = TRAVEL,           -
                     ASSOCIATIONS                 = 4
```

Abbildung 36: KDCDEF Steueranweisungen für RMS

Umstellung des dritten Beispiels auf Cluster-Anwendungen

Angestrebte Cluster-Architektur

Zur Umstellung auf eine Cluster-Architektur wird die UTM-Anwendung RMS zu einer UTM-Cluster-Anwendung mit zwei Servern RMSSERV1 und RMSSERV2 gemacht. Ebenso wird die UTM-Anwendung TRAVEL zu einer Cluster-Anwendung mit zwei Servern TRASERV1 und TRASERV2 gemacht. Jede der beiden TRAVEL Knoten-Anwendungen kommuniziert über ein LPAP-Bündel mit den beiden RMS Knoten-Anwendungen, siehe Abbildung 37.
Details über die Umrüstung beteiligter Datenbanken und Terminal-Pools werden hier nicht ausgeführt; wie diese auf Cluster umgestellt werden wurde ja ausführlich in den vorherigen Beispielen erläutert.

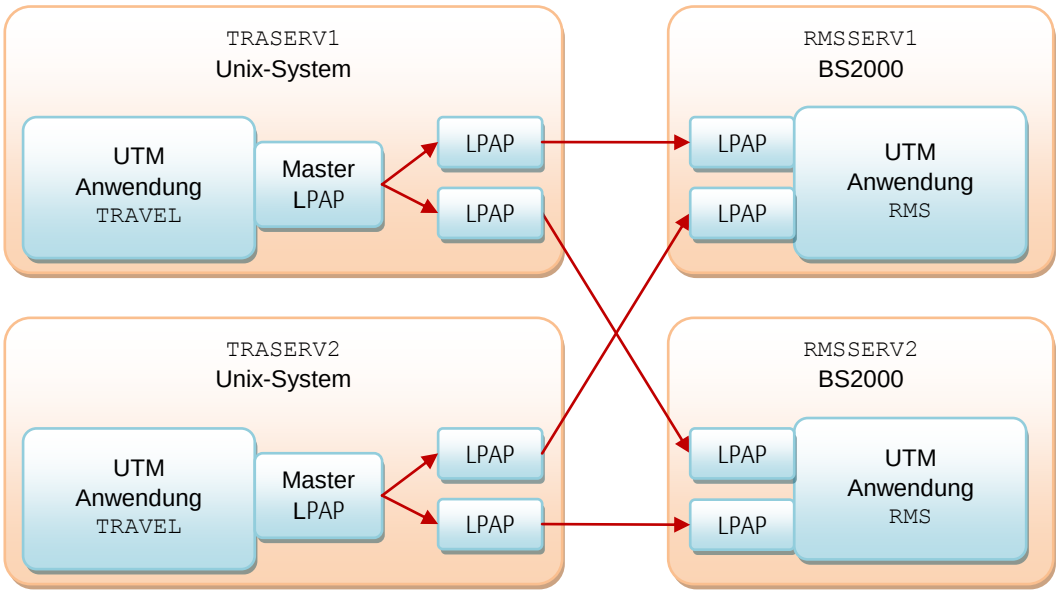

Abbildung 37: Angestrebte Cluster-Architektur

Generierung der UTM-Cluster-Anwendung TRAVEL

Die Umstellung der stand-alone Anwendung auf eine UTM-Cluster-Anwendung erfolgt – analog wie beim ersten Beispiel – nach den in [3], Kapitel 7.11.1 beschriebenen Schritten. Wir konzentrieren uns hier auf die Umstellung der TRAVEL Anwendung, bei der LPAP-Bündel eingeführt werden. Wenn wir von einer einseitigen Kommunikation ausgehen, bei der TRAVEL Anfragen an RMS stellt, braucht auf der RMS-Seite kein LPAP-Bündel generiert zu werden, siehe Abbildung 37.
Die für die Anwendung TRAVEL angepassten KDCDEF-Steueranweisungen sind in Abbildung 38 aufgeführt, die für RMS angepassten Steueranweisungen in Abbildung 39, wobei die Änderungen ggü. der Generierung der stand-alone Anwendung wieder in blau dargestellt sind.

```
OPTION     GEN = (CLUSTER, KDCFILE, ROOTSRC), ROOTSRC = SRC.TRAVROOT
ROOT       TRAVROOT
CLUSTER    CLUSTER-FILEBASE  = CLUST.TRAVFILE,                          -
           BCAMAPPL          = UTMCPING, LISTENER-PORT = 9009,          -
           USER-FILEBASE     = CLUST.TRAVFILE
CLUSTER-NODE FILEBASE  = TRASERV1.TRAVFILE, HOSTNAME = TRASERV1,       -
             NODE-NAME = TRANODE1
CLUSTER-NODE FILEBASE  = TRASERV2.TRAVFILE, HOSTNAME = TRASERV2,       -
             NODE-NAME = TRANODE2
MAX        KDCFILE    = CLUST.TRAVFILE
MAX        APPLINAME  = APTRAVEL
MAX        TASKS      = 7
TAC        KDCWADMI,  PROGRAM = KDCWADMI, ADMIN = YES
TAC        INF01,     PROGRAM = ACTION1
TAC        INF02,     PROGRAM = ACTION2
PROGRAM    KDCWADMI,  COMP = C
PROGRAM    ACTION1,   COMP = C
PROGRAM    ACTION2,   COMP = C
UTMD       APPLICATION-PROCESS-TITLE = (1, 2, 3, 21)
ABSTRACT-SYNTAX      EUROSI,                                           -
                     OBJECT-IDENTIFIER = (1, 3, 9990, 1, 3, 12)
APPLICATION-CONTEXT  EUOSICCR,                                         -
                     OBJECT-IDENTIFIER = (1, 3, 9990, 1, 4, 13),       -
                     ABSTRACT-SYNTAX  = (EUROSI, CCR)
ACCESS-POINT TRAVEL, TRANSPORT-SELECTOR           = C'TRAV',           -
                     SESSION-SELECTOR             = *NONE,             -
                     PRESENTATION-SELECTOR        = *NONE,             -
                     APPLICATION-ENTITY-QUALIFIER = 1,                 -
                     LISTENER-PORT                = 30003
OSI-CON RMS1,        LOCAL-ACCESS-POINT           = TRAVEL,            -
                     OSI-LPAP                     = RMS1,              -
                     NETWORK-SELECTOR             = C'RMSSERV1',       -
                     TRANSPORT-SELECTOR           = C'RMS',            -
                     SESSION-SELECTOR             = (C'SRMS', ASCII),  -
                     PRESENTATION-SELECTOR        = (C'PRMS', ASCII),  -
                     LISTENER-PORT                = 102
OSI-CON RMS2,        LOCAL-ACCESS-POINT           = TRAVEL,            -
                     OSI-LPAP                     = RMS2,              -
                     NETWORK-SELECTOR             = C'RMSSERV2',       -
                     TRANSPORT-SELECTOR           = C'RMS',            -
                     SESSION-SELECTOR             = (C'SRMS', ASCII),  -
                     PRESENTATION-SELECTOR        = (C'PRMS', ASCII),  -
                     LISTENER-PORT                = 102
MASTER-OSI-LPAP RMS, APPLICATION-CONTEXT          = EUOSICCR
OSI-LPAP RMS1,       BUNDLE                       = RMS,               -
                     ASSOCIATION-NAMES            = RMS1,              -
                     ASSOCIATIONS                 = 4,                 -
                     CONTWIN                      = 4,                 -
                     APPLICATION-CONTEXT          = EUOSICCR,          -
                     APPLICATION-PROCESS-TITLE    = (1, 2, 3, 10)      -
```

Abbildung 38: Modifizierte KDCDEF Steueranweisungen für TRAVEL

```
OPTION     GEN = (CLUSTER, KDCFILE, ROOTSRC), ROOTSRC = SRC.RMSROOT
ROOT       RMSROOT
CLUSTER    CLUSTER-FILEBASE   = CLUST.RMSFILE,                          -
           BCAMAPPL           = UTMCPING                                -
           USER-FILEBASE      = CLUST.TRAVFILE
CLUSTER-NODE FILEBASE   = RMSSERV1.RMSFILE, HOSTNAME = RMSSERV1,      -
             NODE-NAME = RMSNODE1
CLUSTER-NODE FILEBASE   = RMSSERV2.RMSFILE, HOSTNAME = RMSSERV2,      -
             NODE-NAME = RMSNODE2
MAX        KDCFILE    = RMSFILE
MAX        APPLINAME  = APRMS
MAX        TASKS      = 10
TAC        KDCWADMI,  PROGRAM = KDCWADMI, ADMIN = YES
TAC        INF01,     PROGRAM = ACTION1
TAC        INF02,     PROGRAM = ACTION2
PROGRAM    KDCWADMI,  COMP = ILCS
PROGRAM    ACTION1,   COMP = ILCS
PROGRAM    ACTION2,   COMP = ILCS
UTMD       APPLICATION-PROCESS-TITLE = (1, 2, 3, 10)
ABSTRACT-SYNTAX      EUROSI,                                           -
                     OBJECT-IDENTIFIER = (1, 3, 9990, 1, 3, 12)
APPLICATION-CONTEXT  EUOSICCR,                                         -
                     OBJECT-IDENTIFIER = (1, 3, 9990, 1, 4, 13),       -
                     ABSTRACT-SYNTAX  = (EUROSI, CCR)
ACCESS-POINT RMS,    TRANSPORT-SELECTOR           = C'RMS',            -
                     SESSION-SELECTOR             = (C'SRMS', ASCII),  -
                     PRESENTATION-SELECTOR        = (C'PRMS', ASCII),  -
                     APPLICATION-ENTITY-QUALIFIER = 1
OSI-CON TRAVEL1,     LOCAL-ACCESS-POINT           = RMS,               -
                     OSI-LPAP                     = TRAVEL1,           -
                     NETWORK-SELECTOR             = C'TRASERV1',       -
                     TRANSPORT-SELECTOR           = C'TRAV',           -
                     SESSION-SELECTOR             = *NONE,             -
                     PRESENTATION-SELECTOR        = *NONE
OSI-CON TRAVEL2,     LOCAL-ACCESS-POINT           = RMS,               -
                     OSI-LPAP                     = TRAVEL2,           -
                     NETWORK-SELECTOR             = C'TRASERV2',       -
                     TRANSPORT-SELECTOR           = C'TRAV',           -
                     SESSION-SELECTOR             = *NONE,             -
                     PRESENTATION-SELECTOR        = *NONE
OSI-LPAP TRAVEL1,    ASSOCIATION-NAMES            = TRAVEL1,           -
                     ASSOCIATIONS                 = 4,                 -
                     CONTWIN                      = 0,                 -
                     APPLICATION-CONTEXT          = EUOSICCR,          -
                     APPLICATION-PROCESS-TITLE    = (1, 2, 3, 21)      -
```

Abbildung 39: Modifizierte KDCDEF Steueranweisungen für RMS

Variante dieses Beispiels: LU6.1 statt OSI-TP

In einer Variante des dritten Beispiels wird davon ausgegangen, dass die beiden Anwendungen TRAVEL und RMS nicht über OSI-TP sondern über LU6.1 kommunizieren. Dann müssen alle vorgesehenen parallelen Verbindungen und Sessions explizit aufgeführt werden. Bei Cluster-Anwendungen bedeutet dies, dass Verbindungen und Sessions jedes Knotens der einen Cluster-Anwendung mit jedem Knoten der anderen Cluster-Anwendung definiert werden müssen. Der Abbildung 37 kann man entnehmen, dass das in unserem Fall 4 Gruppen von Sessions sind. Im Beispiel sehen wir zwischen jedem Knoten der Cluster-Anwendung `TRAVEL` und jedem Knoten der Cluster-Anwendung `RMS` jeweils 4 parallele Verbindungen vor.

Dabei kann jedoch Knoten `TASERV1` nur die eine Hälfte und Knoten `TRASERV2` die andere Hälfte der diesen Verbindungen zugeordneten Sessions nutzen. Dies kann ab openUTM V6.2 durch die Angabe `NODE-NAME` in der `LSES`-Anweisung festgelegt werden. Als `NODE-NAME` wird dabei der logische Knotenname angegeben, der in der `CLUSTER-NODE`-Anweisung definiert ist.

In Abbildung 40 und Abbildung 41 sind nun die erforderlichen KDCDEF-Steueranweisungen für die LU6.1 Kommunikation aufgeführt. Andere Steueranweisungen wie z.B. `CLUSTER`, `MAX`, `TAC`, `PROGRAM` bleiben unverändert wie in Abbildung 38 bzw. Abbildung 39 und sind hier nicht wiederholt.

```
…
BCAMAPPL TRA1
BCAMAPPL TRA2
*
MASTER-LU61-LPAP RMS
LPAP RMS1, SESCHA = SESRMS, BUNDLE = RMS
LPAP RMS2, SESCHA = SESRMS, BUNDLE = RMS
*
SESCHA SESRMS, CONTWIN = NO, PLU = YES
*
* 4 Connections to RMS1
CON RMS1, PRONAM=RMSSERV1, BCAMAPPL = TRA1, LPAP = RMS1
CON RMS2, PRONAM=RMSSERV1, BCAMAPPL = TRA1, LPAP = RMS1
CON RMS1, PRONAM=RMSSERV1, BCAMAPPL = TRA2, LPAP = RMS1
CON RMS2, PRONAM=RMSSERV1, BCAMAPPL = TRA2, LPAP = RMS1
*
* 4 Connections to RMS2
CON RMS1, PRONAM=RMSSERV2, BCAMAPPL=TRA1, LPAP = RMS2
CON RMS2, PRONAM=RMSSERV2, BCAMAPPL=TRA1, LPAP = RMS2
CON RMS1, PRONAM=RMSSERV2, BCAMAPPL=TRA2, LPAP = RMS2
CON RMS2, PRONAM=RMSSERV2, BCAMAPPL=TRA2, LPAP = RMS2
*
* 4 Sessions TRAVEL1 to RMS1
LSES TR1RM1A, RSES = RM1TR1A, LPAP = RMS1, NODE-NAME = TRANODE1
LSES TR1RM1B, RSES = RM1TR1B, LPAP = RMS1, NODE-NAME = TRANODE1
LSES TR1RM1C, RSES = RM1TR1C, LPAP = RMS1, NODE-NAME = TRANODE1
LSES TR1RM1D, RSES = RM1TR1D, LPAP = RMS1, NODE-NAME = TRANODE1
*
* 4 Sessions TRAVEL1 to RMS2
LSES TR1RM2A, RSES = RM2TR1A, LPAP = RMS2, NODE-NAME = TRANODE1
LSES TR1RM2B, RSES = RM2TR1B, LPAP = RMS2, NODE-NAME = TRANODE1
LSES TR1RM2C, RSES = RM2TR1C, LPAP = RMS2, NODE-NAME = TRANODE1
LSES TR1RM2D, RSES = RM2TR1D, LPAP = RMS2, NODE-NAME = TRANODE1
*
* 4 Sessions TRAVEL2 to RMS1
LSES TR2RM1A, RSES = RM1TR2A, LPAP = RMS1, NODE-NAME = TRANODE2
LSES TR2RM1B, RSES = RM1TR2B, LPAP = RMS1, NODE-NAME = TRANODE2
LSES TR2RM1C, RSES = RM1TR2C, LPAP = RMS1, NODE-NAME = TRANODE2
LSES TR2RM1D, RSES = RM1TR2D, LPAP = RMS1, NODE-NAME = TRANODE2
*
```

Abbildung 40: KDCDEF Steueranweisungen für TRAVEL bei LU6.1

```
…
BCAMAPPL RMS1
BCAMAPPL RMS2
*
LPAP TRAVEL1, SESCHA = SESTRA
LPAP TRAVEL2, SESCHA = SESTRA
*
SESCHA SESTRA, CONTWIN = YES, PLU = NO
*
* 4 Connections to TRAVEL1
CON TRA1, PRONAM = TRASERV1, BCAMAPPL = RMS1, LPAP = TRAVEL1
CON TRA1, PRONAM = TRASERV1, BCAMAPPL = RMS2, LPAP = TRAVEL1
CON TRA2, PRONAM = TRASERV1, BCAMAPPL = RMS1, LPAP = TRAVEL1
CON TRA2, PRONAM = TRASERV1, BCAMAPPL = RMS2, LPAP = TRAVEL1
*
* 4 Connections to TRAVEL2
CON TRA1, PRONAM = TRASERV2, BCAMAPPL = RMS1, LPAP = TRAVEL2
CON TRA1, PRONAM = TRASERV2, BCAMAPPL = RMS2, LPAP = TRAVEL2
CON TRA2, PRONAM = TRASERV2, BCAMAPPL = RMS1, LPAP = TRAVEL2
CON TRA2, PRONAM = TRASERV2, BCAMAPPL = RMS2, LPAP = TRAVEL2
*
* 4 Sessions RMS1 to TRAVEL1
LSES RM1TR1A, RSES = TR1RM1A, LPAP = TRAVEL1, NODE-NAME = RMSNODE1
LSES RM1TR1B, RSES = TR1RM1B, LPAP = TRAVEL1, NODE-NAME = RMSNODE1
LSES RM1TR1C, RSES = TR1RM1C, LPAP = TRAVEL1, NODE-NAME = RMSNODE1
LSES RM1TR1D, RSES = TR1RM1D, LPAP = TRAVEL1, NODE-NAME = RMSNODE1
*
* 4 Sessions RMS2 to TRAVEL1
LSES RM2TR1A, RSES = TR1RM2A, LPAP = TRAVEL1, NODE-NAME = RMSNODE2
LSES RM2TR1B, RSES = TR1RM2B, LPAP = TRAVEL1, NODE-NAME = RMSNODE2
LSES RM2TR1C, RSES = TR1RM2C, LPAP = TRAVEL1, NODE-NAME = RMSNODE2
LSES RM2TR1D, RSES = TR1RM2D, LPAP = TRAVEL1, NODE-NAME = RMSNODE2
*
* 4 Sessions RMS1 to TRAVEL2
LSES RM1TR2A, RSES = TR2RM1A, LPAP = TRAVEL2, NODE-NAME = RMSNODE1
LSES RM1TR2B, RSES = TR2RM1B, LPAP = TRAVEL2, NODE-NAME = RMSNODE1
LSES RM1TR2C, RSES = TR2RM1C, LPAP = TRAVEL2, NODE-NAME = RMSNODE1
LSES RM1TR2D, RSES = TR2RM1D, LPAP = TRAVEL2, NODE-NAME = RMSNODE1
*
```

Abbildung 41: KDCDEF Steueranweisungen für RMS bei LU6.1

Performance Überlegungen

Die bei einer UTM-Cluster-Anwendung gewährleistete Knoten-weite Wirkung globaler Speicherbereiche kann bei Nutzung dieser Speicherbereiche zu mehr I/O pro Transaktion führen, als dies bei der ursprünglichen stand-alone UTM-Anwendung der Fall war. Dies kann auch den maximal erzielbaren Durchsatz (Transaktionen pro Sekunde) reduzieren. Relevant ist dabei auch die Performance des Network-File-Systems (NFS). Meist wird der Durchsatz aber nicht durch openUTM selbst sondern durch die Anwendung und die Datenbankzugriffe begrenzt. Da die konkrete Auswirkung stark von der Anwendungsarchitektur abhängt, können keine konkreten Werte angegeben werden. Stattdessen wird empfohlen, ab einem erwarteten Durchsatz von mehr als 50 Transaktionen pro Sekunde eine Last-Messung mit der konkreten Anwendung durchzuführen.
Bei BS2000 nutzt openUTM intensiv den im XCS-Verbund zur Verfügung stehenden *Distributed Lock Manager* (DLM). Dieser basiert auf einer Token-Ring-Kommunikation. Um Verzögerungen bei Lock-Anforderungen zu vermeiden sollte in der NSM-Konfiguration die Token-Delay-Time angepasst werden z.B. auf min/max = 0/1 ms, 0/2 ms oder 0/3 ms (Kommando für min/max = 0/1 ms: `/MODIFY-NSM-ENVIRONMENT TOKEN-DELAY-TIME = *BY-PARAMETER (MIN-DELAY-TIME = 0, MAX-DELAY-TIME = 1`, siehe [6], Kapitel 9). Diese Einstellungen bewirken ein unverzügliches Weiterleiten des Token bei Lock-Anforderungen und eine nur geringe Verzögerung beim Weiterleiten eines leeren Token, was aber auch zu einem höheren CPU-Zeit-Verbrauch im Leerlauf führt. Welche Einstellungen im konkreten Fall optimal sind, hängt stark von der Anwendungsarchitektur ab. Es ist sinnvoll, die optimale Einstellung als Kompromiss zwischen CPU-Verbrauch im Leerlauf und Durchsatz unter Hochlast experimentell zu ermitteln. Abbildung 42 zeigt schematisch den Einfluss der Delay-Time-Einstellung auf den Leerlauf-CPU-Verbrauch und den maximalen Transaktions-Durchsatz.

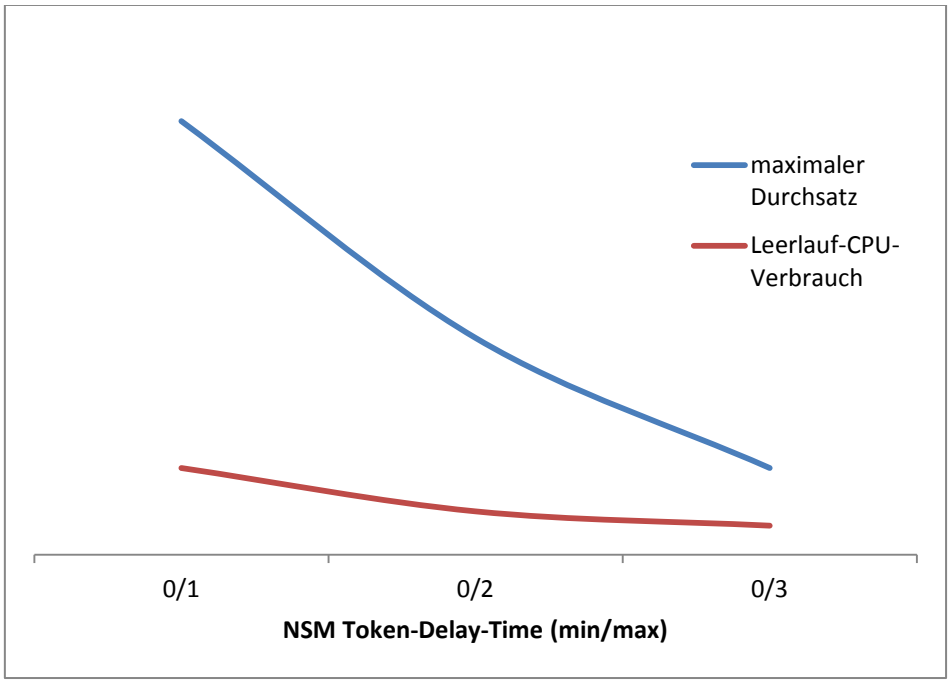

Abbildung 42: Einfluss der NSM-Delay-Timer-Einstellung

Ähnlich wie die Gesamtleistung eines Multi-Prozessor-Servers nicht linear mit der Anzahl der Prozessoren steigt, steigt die Gesamtleistung einer UTM-Cluster-Anwendung auch nicht linear mit der Anzahl der Cluster-Knoten. Konkrete Werte können nicht angegeben werden, denn diese werden nicht nur von openUTM beeinflusst sondern hängen stark von der Anwendungsarchitektur und der Datenbank ab.

Fazit

Anhand verschiedener Beispiele wurde gezeigt, wie die Umstellung einer existierenden stand-alone UTM-Anwendung auf eine UTM-Cluster-Anwendung auf einfache Weise möglich ist. Konkrete Kunden-Situationen werden naturgemäß von den allgemein und einfach gehaltenen Beispielen abweichen. Dennoch sollten die Beispiele eine gute Basis und Hilfestellung für konkrete Umstellungsvorhaben sein.

Contact
FUJITSU Technology Solutions GmbH
Address: Mies-van-der-Rohe-Str. 8, 80807 München, Germany
E-mail: openSEAS@ts.fujitsu.com
Website: www.fujitsu.com/de
2015-03-30