White paper
Fujitsu Software openUTM Go2Cluster

System downtimes are not acceptable for applications deployed in enterprise-critical situations. Fujitsu Software openUTM supports high availability via the UTM cluster functionality according to Fujitsu's *Dynamic Infrastructures* strategy.

Content

Introduction

Downtimes cannot be tolerated especially when applications are used in critical company departments. High-availability levels for applications as well as the option of automatically distributing workloads during peak load times have top priority and they are an important feature of dynamic infrastructures. Furthermore, an increasing number of applications are being equipped with a web site making 7x24 hr availability a necessity. As described in the White Paper *Fit4Cluster* the cluster support of openUTM is an answer to such requirements.
This White Paper *Go2Cluster* uses various examples to explain in detail how a stand-alone application can be converted to a UTM cluster application. The document is aimed at developers and administrators wishing to make such a conversion. This compact description should save you time in comparison to looking for the required steps in various manuals. The explanation of each step still refers to the respective manual pages which provide more in-depth information.
Each example begins with a description of the stand-alone application as the starting-point. This is followed by a detailed explanation of the steps required for the conversion. This is then followed by some additional variations of that particular example.
Subsequent to the examples performance considerations are discussed. Hints in particular are given for wise configuration settings in context with the XCS cluster on BS2000.

References

[1] openUTM V6.3 – Generating Applications: http://manuals.ts.fujitsu.com
[2] openUTM V6.3 – Client-Server Communication with openUTM for the UPIC Carrier System: http://manuals.ts.fujitsu.com
[3] openUTM V6.3 – Using openUTM Applications under BS2000: http://manuals.ts.fujitsu.com
[4] openUTM V6.3 - Using openUTM Applications under Unix Systems and Windows Systems: http://manuals.ts.fujitsu.com
[5] SESAM/SQL-Server V8.0A – Database operations: http://manuals.ts.fujitsu.com
[6] HIPLEX MSCF – BS2000 Processor Networks – user guide: 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 Servers via NAT http://www.linuxvirtualserver.org/VS-NAT.html

First example: UPIC clients – UTM application - SESAM

Starting Point: Stand-alone application with SESAM/SQL

The UTM application considered in this example runs on a mainframe SERV1 with BS2000 as operating system, if necessary as guest system under VM2000. It uses SESAM/SQL as a database system whereby the database handler runs on the same server as the UTM application. It is assumed that application data are distributed with respect to their contents over several databases which are controlled by the same database handler (DBH) – as this is possible and common use for SESAM/SQL. The client side has UPIC clients which run on the client PCs under MS Windows®. See Figure 1. It is assumed that the UTM application does not use data areas other than GSSB and ULS across services.

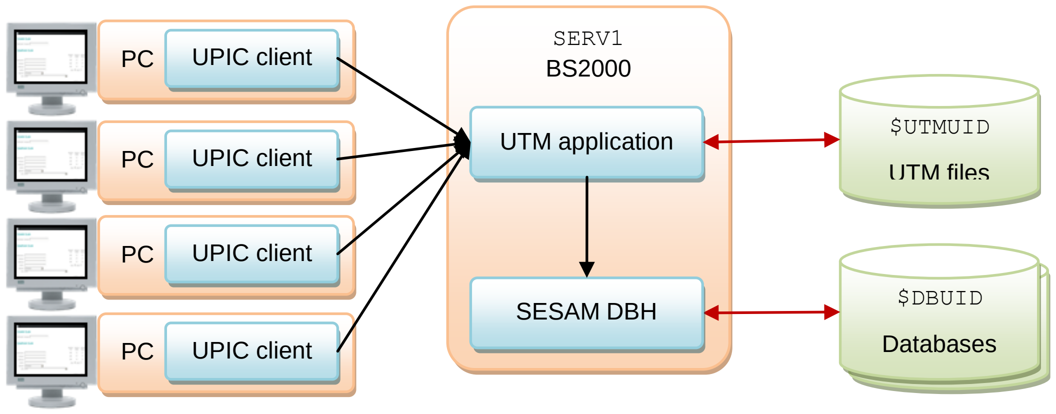

Figure 1: Stand-alone application

Generate the stand-alone application

The KDCDEF control statements listed in Figure 2 to generate this UTM application are restricted to the main elements required for this particular example and for the conversion to a UTM cluster application. For example, USERs are not generated and there is no generation of other access controls.

Only 3 TACs are generated as example: an administration TAC linked with `KDCWADMI` and used for administration via `WinAdmin`, and two action TACs which are linked with program units `ACTION1` and `ACTION2` created by the user. SESAM/SQL is used as the database system. A pool for up to 100 UPIC clients is generated. Details about these control statements are in [1], section 6.

```
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       = TPCL#,    NUMBER = 100
```

Figure 2: KDCDEF control statements

When executing KDCDEF with these control statements the UTM files are generated under the user ID `$UTMUID` with the prefix `SAMPLE1`.

Generate the database

The SESAM/SQL database is configured using the control statements listed in Figure 3 which again concentrate on the main points required for this example. Based on the name of the UTM application `SAMPLE1` `DBH-NAME = 1` is selected. V is selected as configuration name. It is assumed that there are four databases: `SQLDB1`, `SQLDB2`, `SQLDB3`, and `SQLDB4`. Four database threads are configured according to the four UTM tasks. Details about these control statements are in [5], section 3.

```
//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)
```

Figure 3: Control statements to configure the SESAM databases

When executing these control statements the database files are expected under the user ID `$DBUID`.

Start the stand-alone application

The stand-alone application can be started after the database has been configured and the database handler has been started. The start parameters specified are listed in Figure 4 . Details about the start parameters can be seen in [3], section 5.1.1.

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

Figure 4: openUTM start parameters

UPIC clients

The UPIC clients use the *Side Information file* (`upicfile`) described in Figure 5 . This example assumes that the values for `HOSTNAME` and `TSEL` in the client program are not modified via `SET` instructions. Detailed information about the structure of the entries in the `upicfile` can be seen in [2], section 6.2.

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

Figure 5: upicfile

Conversion of the first example to a UTM cluster application

Targeted UTM cluster application

A second mainframe server `SERV2` with BS2000 as operating system is to be added which, if necessary, can also be established aside of `SERV1` as a guest system under VM2000. See Figure 6. The following uses the term *system* (instead of *server*) in order to cover both situations to the same extent. The stand-alone application is to be converted to a UTM cluster application with `SERV1` and `SERV2` as nodes. The pubset which has the ID `$UTMUID` to save the UTM files is converted to an XCS pubset.
The four databases are distributed over the two systems. On each of the two systems a database handler DBH and in addition a distribution component DCN is installed so that each system can also access the databases of the other system.
The UPIC clients are to implement load balancing and set up alternating connections to `SERV1` and `SERV2` whereby `SERV1` is to be selected more frequently than `SERV2` due to the faster database accesses.

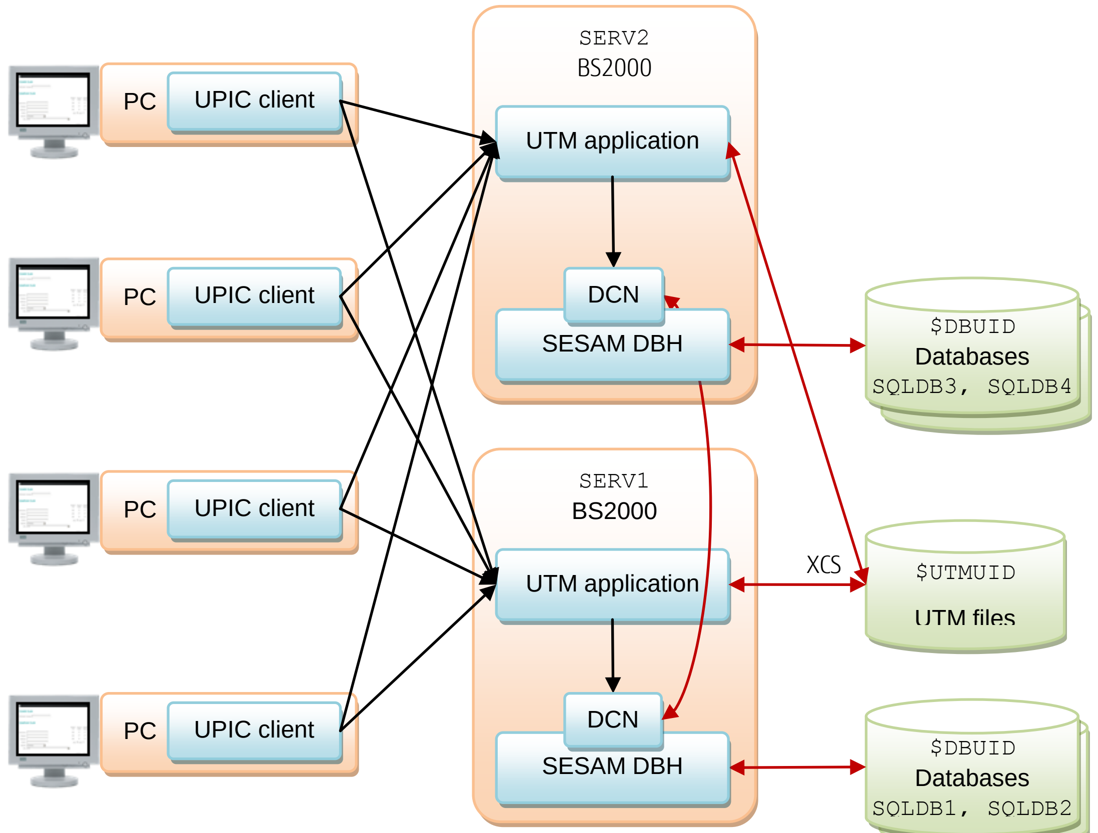

Figure 6: UTM cluster application

Structure of the hardware configuration

First of all, the second BS2000 system `SERV2` must be established, as stated earlier, either as an additional server or as a further guest system under VM2000 on the existing server.
HIPLEX-MSCF (V4.0 or higher) must now also be installed on both systems. openUTM uses the now available *Distributed Lock Management* (DLM) for the cluster functionality. More information about installing HIPLEX-MSCF can be found in [6], section 5.
An XCS cluster with the name `CLUST`, consisting of the systems `SERV1` and `SERV2` is configured using the MSCF configuration file described in Figure 7. Again, we will just concentrate on the main points regarding the operation of the UTM cluster; for example, there is no need to define passwords and the default value is selected for many parameters. More information about the configuration commands can be found in [6], sections 5.2.4 and 9.

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

Figure 7: MSCF configuration file

The XCS pubset is now configured with the command listed in Figure 8 . The `UTMC` pubset already used by the stand-alone application is still to be used. The `MODIFY-MASTER-CATALOG-ENTRY` is thus executed. Details about this command and the required setting of the BS2000 system parameter `MCXSPXCS` can be found in [6], section 5.6. General information about the XCS array is in section 6.7.

```
/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,          -
```

Figure 8: Enter the shared pubset in the MRSCAT

Generate the UTM cluster application

The conversion of the stand-alone application to a UTM cluster application is carried out according to the steps listed in [3], section 7.11.1. First of all, the control statements for KDCDEF are modified; see Figure 9 (changes compared to the control statements for the original stand-alone application are in blue).

`$UTMUID.CLUST.SAMPLE1` is set as cluster filebase and also as user filebase. In addition to the two nodes `SERV1` and `SERV2` a reserve node `RESRV` is generated so as to be ready for later upgrades. The operand `FAILURE-CMD` is explained in context with the automated warm start, the two operands `RESTART-TIMER-SEC` and `EMERGENCY-CMD` are explained later when the dynamic modification of the database configuration for fail-over is discussed. The optional fifth and sixth procedure parameters (`PROC-PAR`) were introduced with openUTM V6.2.

The `CLUSTER-NODE` statements define logical node names, as possible with openUTM V6.2 or later versions. This simplifies later replacement of servers since the logical name can thus be maintained while the `HOSTNAME` may change. If the optional definition of logical node names is omitted, the control statements can also be used with openUTM V6.1.

```
OPTION     GEN = (CLUSTER, KDCFILE, ROOTSRC), ROOTSRC = KDCROOT1
ROOT       KDCROOT1
CLUSTER    CLUSTER-FILEBASE  = $UTMUID.CLUST.SAMPLE1,                       -
           BCAMAPPL          = UTMCPING, LISTENER-PORT = 9009,              -
           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
```

Figure 9: Modified KDCDEF control statements

When executing KDCDEF with these control statements an initial `KDCFILE $UTMUID.CLUST.SAMPLE1.KDCA` is generated. This file must now be copied to node-specific files, e.g. via the commands listed in Figure 10. As the new KDCFILES have a name different to that of the previous KDCFILE of the stand-alone application, the previous KDCFILE is retained for a KDCUPD run.

```
/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
```

Figure 10: Copy the KDCFILE

The stand-alone application is now terminated normally. We select the node `SERV1` in order to transfer from the stand-alone application to the cluster application – via a KDCUPD run – the user data (e.g. passwords) and any background jobs not yet executed as well as asynchronous services. The control statements for this KDCUPD run are specified in Figure 11 . More information about the KDCUPD control statements can be found in [1], section 8.3.

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

Figure 11: Control statements for KDCUPD

Automated warm start of a failed node application
To assure high availability warm start of a failed node application should be automated. This can be achieved by configuring a failure procedure `$UTMUID.FAILURE`, see Figure 9, which will be called automatically by cluster ring monitoring when it detects the break-down of a node application.
An example of such a failure procedure is given in Figure 12. The procedure assumes that the application has been started with job variable monitoring. Parameters `NODENAME` and `TRMAREASON` are available from openUTM V6.2 on.

```
/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)'
```

Figure 12: Failure Procedure

Adapt the database configuration
In order to access the database from node `SERV2`, database handlers as well as distribution components SESDCN must be started on both nodes `SERV1` and `SERV2` using the SESDCN control statements listed in Figure 14 or Figure 16. Remote access to the databases on the opposite server is thus established on each server. No modifications for the application program are necessary. The control statements listed in Figure 3 for configuring the database are distributed over both servers, whereby on `SERV1` the databases `SQLDB1` and `SQLDB2` and on `SERV2` the databases `SQLDB3` and `SQLDB4` are established, see Figure 13 or Figure 15. More information about the SESDCN control statements can be found in [5], section 4.4.

```
//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
```

Figure 13: Control statements to configure the SESAM databases on **`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,                      -
```

Figure 14: DCN control statements for **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
```

Figure 15: Control statements to configure the SESAM databases on **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,                      -
```

Figure 16: DCN control statements for **SERV2**

Dynamic modification of database configuration for fail-over
To guarantee high availability not only openUTM needs to react on failure of a node, but also the databases. When a node has failed, all databases should be processed from the still running node. This is achieved by configuring a procedure `$UTMUID.EMERGENCY`, see Figure 9,

which is in such a case automatically called by UTM ring monitoring if no successful warm start was possible within the 3 minutes specified by `RESTART-TIMER-SEC = 180`.
Such an emergency procedure is in Figure 17. It switches the database configuration dependent on the failing node. For instance, assume `SERV2` has failed (then part of the `IF` command). Then the first SESADM call arranges that `SERV1` also handles the databases `SQLDB3` and `SQLDB4`. In the second call DCN on `SERV1` is reconfigured so that all databases are local to it. As provision for a possible future restart of `SERV2` the third call determines all databases remote to `SERV2`.
The case where `SERV1` has failed is treated symmetrically (`ELSE` part of the `IF` command). More information about SESAM administration statements can be found in [5], section 5.1.3.4.
Parameters `NODENAME` and `TRMAREASON` are available from openUTM V6.2 on. They are necessary for the node recovery explained later. `/SET-PROCEDURE-OPTIONS` is only required for node recovery, too.

```
/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
```

Figure 17: Emergency procedure

Start the node applications on `SERV1` and `SERV2`
The KDCUPD run was executed on the node `SERV1`. The node application on this node must be started first using the start parameters listed in Figure 18 . As soon as the node application has been successfully started on `SERV1` (e.g. determined via WinAdmin "*Availability Check*"), the node application can also be started on `SERV2` using the same start parameters.

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

Figure 18: Modified start parameters for the node applications

Adapt the UPIC client
For UPIC clients it is sufficient to adapt the `upicfile` as described in Figure 19.
The HD entries are replaced by CD entries whereby the CD entries for `SERV1` are listed twice in order to bias weighting for `SERV1`.
`CONVERSION=IMPLICIT` has the same effect as with the stand-alone application and prefix HD (instead of SD). More information about the CD entries can be found in [2], section 6.2.2.
The client programs can be retained unchanged and do not have to be re-compiled or re-installed.

```
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
```

Figure 19: Modified upicfile

Improvement of high availability

Automatic node recovery
Provisions have already been taken to allow continuous availability of the UTM application on remaining cluster nodes should a node break down and a warm start fail. However, in rare cases it is possible that, when a cluster node broke down, it held some locks on global resources. As of openUTM V6.2 these locks can be released via node recovery, see [3] section 7.5.6. Node recovery can be invoked manually or automatically.
For automatic node recovery in the present example the emergency procedure in Figure 17 can be extended as shown in Figure 20. This extension starts the UTM application with start parameters for node recovery where it is assumed that the bound application is the element `APPL` in library `LIB`. The start parameters are the same as for normal start of the application (Figure 18), extended by the specification `NODE-TO-RECOVER` the value of which is received from the parameter `NODENAME` of the emergency procedure. Configuration settings for SESAM are assumed in file `SESAM.CONFIG`. Optionally `RESET-PTC = YES` can be specified to also reset transactions in state *prepare to commit* (PTC). In case of node recovery the specification `STARTNAME` is not needed.

```
…
/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
```

Figure 20: Extension of the emergency procedure

If a start procedure exists for the start of the UTM application, this can also be extended by an optional parameter which initiates node recovery instead of the application start. The start procedure thus modified can be called via its additional parameter at the end of the emergency procedure. It is thus easier to ensure consistency between the start parameters for the normal application start and the start parameters for the node recovery.
If the UTM application interacts with database systems which should be involved in node recovery then SESAM V8, UDS V2.7, or XA database systems are required.

Putting the reserve node into operation
When the failed server has been repaired and operation has been resumed, the node application on this server can simply be restarted. In the event of previous node recovery this is usually a cold start of the node application.

However, if the failed server cannot be put back into operation (immediately) and you want to add another server instead, the reserve node as defined in Figure 9 can be reconfigured.
Let's assume that the new server is called `SERV3`. Then the initial KDCFILE `$UTMUID.CLUST.SAMPLE1.KDCA` must also be copied to `$UTMUID.SERV3.SAMPLE1.KDCA` analogously to Figure 10. Afterwards administration (e.g. via WinAdmin) can be used to change the features of the node known as `RESRV`, in particular to change the host name to `SERV3` and the file base to `$UTMUID.SERV3.SAMPLE1`. Subsequently, the node application on `SERV3` can be started. It then automatically takes over all administrative changes that have been made since the generation of the initial KDCFILE.

Variant for this example: dynamic definition of LU names via SET

In a variant of the above example it is assumed that the selection of the application is program-driven in the UPIC clients. For example, this can be necessary when the UPIC clients are to be universal and prepared to communicate with various UTM applications: in addition to the `SAMPLE1` application running on `SERV1` communicating with other applications which possibly run on other servers.

Starting-point

It is assumed that the UPIC clients dynamically define the respective LU name (as shown in Figure 21 by calling a function `determine_LU_name`) provided by the client program. The setting is then via `Set_Partner_LU_Name`. In this case a `upicfile` is not required. Likewise, the TAC is dynamically defined (shown in Figure 21 by calling a function `determine_TAC_name` provided by the client program). Figure 21 is an extract from such a client program. For clarity reasons, return code checks and corresponding error handling is omitted.

```
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 */
…
```

Figure 21: Extract from the client program

Adapt the UPIC client

`Set_Partner_LU_Name` calls must be eliminated as preparation for the conversion to load balancing for a UTM cluster application. A `upicfile` is configured in which the LU name to be converted later to cluster is pre-defined *as Symbolic_Destination_Name*; see Figure 22. This `upicfile` must then be distributed to all clients.
The client program has been adapted so that the selection of the UTM cluster application to be addressed for this LU name is via `Initialize_Conversation` referring to the *Symbolic_Destination_Name* instead of via `Set_Partner_LU_Name`. After this conversion, the UPIC client can be reconfigured to a load balancer as described above.
The `Set_TP_Name` calls can be retained. The `Set_Partner_LU_Name` calls for other than the UTM cluster application can also remain unchanged.
Of course, the client program modified in such a way must be re-compiled, linked and installed on all clients. This can be carried out and tested before the conversion to cluster.
When subsequently converting to cluster the HD statements are converted to CD statements in the `upicfile` as described above under "Adapt the UPIC client".

```
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 */
…
```

Figure 22: `upicfile` and extract from the modified client program

Second example: MT9750 clients – UTM application - Oracle

Starting Point: Stand-alone application with Oracle database

The UTM application considered in this example runs on a mainframe SERV1 with BS2000 as operating system, if necessary as guest system under VM2000. It uses an exported Oracle® database system (Oracle® instance) on a separate Linux server SERVDB whereby the following assumes Version 10g. PCs with terminal emulation MT9750 are used on the client side; see Figure 23. The clients are connected via a Local Area Network (LAN) with SERV1. The last router is shown explicitly in Figure 23 in front of SERV1 as this later plays a role in load balancing when converting to a UTM cluster application. This time it is assumed that the UTM application uses global storage areas GSSB.

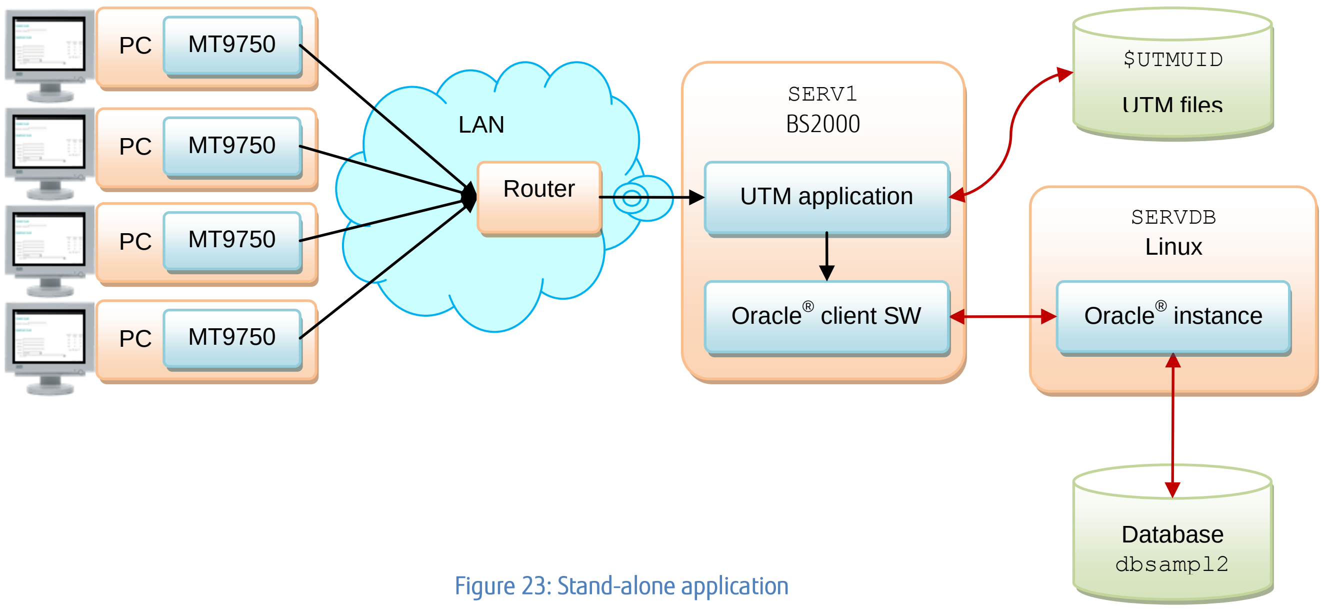

Figure 23: Stand-alone application

Generate the stand-alone application

The KDCDEF control statements listed in Figure 24 to generate this UTM application concentrate on the main elements – as in the first example. Oracle® version 10g has been selected as the database system and a separate Linux-based database server SERVDB is used. The Oracle® Client software is installed on the mainframe SERV1 under the ID $ORAC1020 and communicates with the Oracle® instance installed on the SERVDB. The DATABASE control statement names the Oracle® client software installed on the mainframe which sets up the connection to the Oracle® instance; see also [7], section 8. Database access (user and password) will be defined by generating information for the database. It is assumed that the Oracle® client software is statically linked.

A pool for up to 100 terminals is generated which are emulated using MT9750. Details about these control statements are in [1], section 6. It is assumed that the two program units ACTION1 and ACTION2 access up to 10 common GSSBs.

```
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
```

Figure 24: KDCDEF control statements

Configure the database system

To configure the Oracle® client software on the mainframe the configuration file tnsnames.ora shown in Figure 25 is selected which specifies the connection to the Oracle® instance on SERVDB as SERVICE1. Details about these configuration parameters can be found in [7], section 9 or [8], section 6.

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

Figure 25: Oracle® configuration file tnsnames.ora

Start the stand-alone application

The stand-alone application can be started after the database has been configured and started. The start parameters specified are listed in Figure 26 . Access to database on `SERVDB` was defined while generating, i.e. only proxies are given here. Details about the `.UTM` start parameters can be found in [3], section 5.1.1 and details about the `.RMXA` start parameters can be read in [7], section 8.

```
.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
```

Figure 26: openUTM start parameters

Having started the application clients can log-in via the terminal emulation MT9750 on `SERV1` with the application `SAMPLE2`.

Conversion of the second example on a UTM cluster application with Oracle® RAC

Cluster architecture planned

Analogously to the first example a second mainframe system `SERV12` is added with BS2000 as operating system. The previous system `SERV1` is renamed this time as `SERV11`, and `SERV1` is retained as virtual system name for the load balancing, see Figure 27. This ensures that clients can still log into SERV1 on `SAMPLE2` in order to reach the UTM cluster application. The last router in front of the former `SERV1` is now reconfigured as a load balancer.

The stand-alone application is to be converted to a UTM cluster application with `SERV11` and `SERV12` as nodes. To do so, the pubset with ID `$UTMUID` to store UTM files is – as explained in the first example (cf. Figure 6) – turned into an XCS pubset.

A second database server `SERVDB2` is added with Linux as operating system so that the Oracle® database can be operated in a RAC configuration. The database `dbsampl2` must be on a Network File System (NFS), Oracle Cluster Files System (OCFS) or Oracle ASM so as to be accessed by both database servers.

Generate the UTM cluster application

The stand-alone application is converted to a UTM cluster application (similarly as in the first example) according to the steps described in [3], section 7.11.1. The adapted control statements are listed in Figure 28 , whereby the changes in contrast to generation of the stand-alone application are again shown in blue. The operands `RESTART-TIMER-SEC` and `EMERGENCY-CMD` will be described later when looking at the load balancer.

The failure procedure specified by `FAILURE-CMD` is again used for automated warm start as explained in detail in the first example. However, you should be aware that resources (like the GSSBs used in this example), which were locked at the time of the break-down of a node application, will remain locked until warm start is finished. While it is normally sufficient for batch applications to choose `MAX RESWAIT` in an accordingly generous size, for dialogue applications it is often wiser to give the user an intermediate notice on return code "40Z" and repeat the attempt.

The initial `KDCFILE $UTMUID.CLUST.SAMPLE2.KDCA` that arises when executing KDCDEF must be copied to node-specific files. This has been explained in detail in the first example. There is thus no need to reconsider details here.

Since the two program units `ACTION1` and `ACTION 2` access common GSSBs, we must avoid all tasks attempting at the same time to access the GSSBs and thus provoke a task deadlock; see [3] section 7.2.2. Therefore, the appropriate TACs are put in a TAC class 1 and the tasks for this TAC class are limited to 3 while the node application is started with 4 tasks.

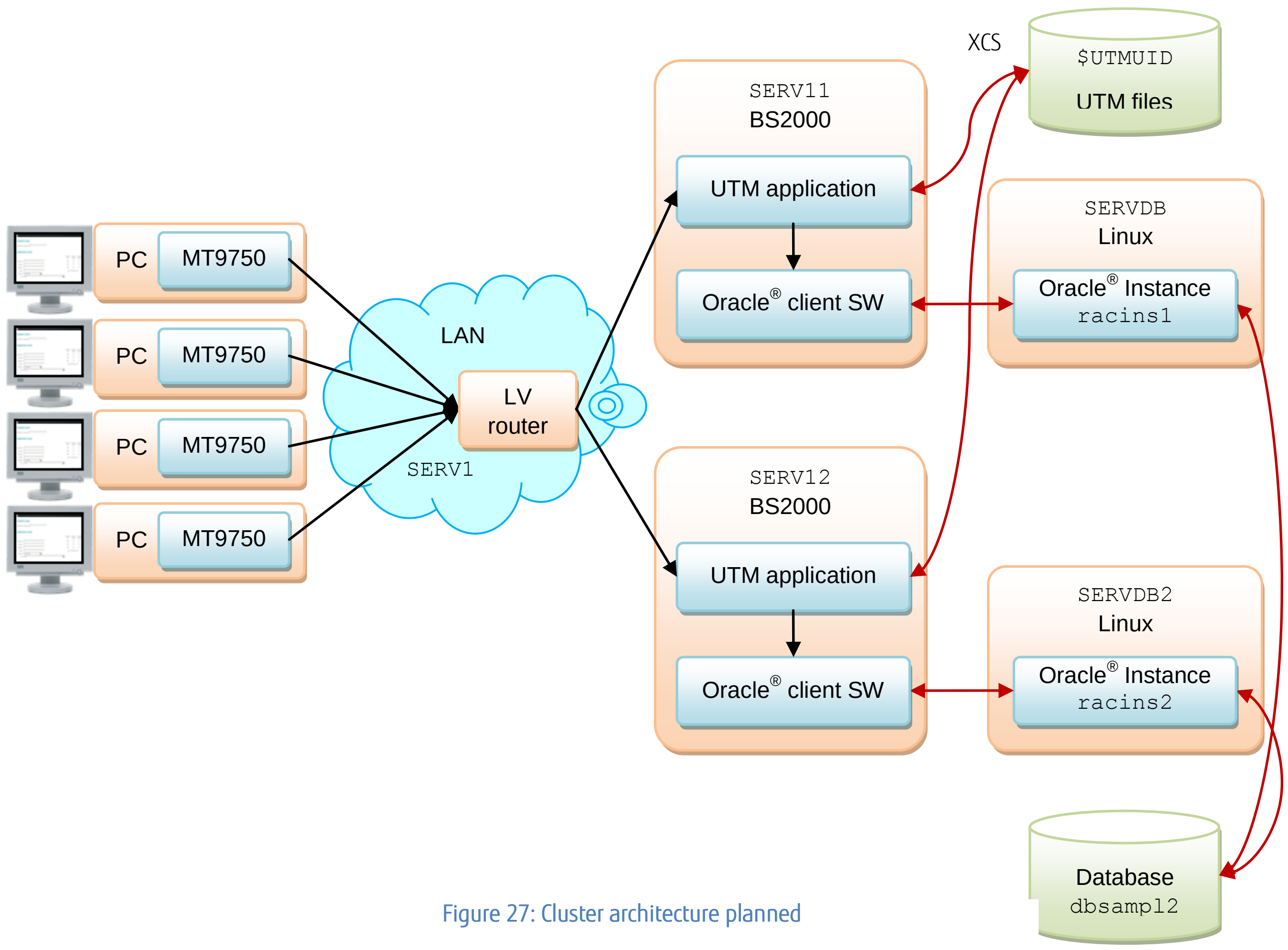

Figure 27: Cluster architecture planned

```
OPTION     GEN = (CLUSTER, KDCFILE, ROOTSRC), ROOTSRC = KDCROOT2
ROOT       KDCROOT2
CLUSTER    CLUSTER-FILEBASE  = $UTMUID.CLUST.SAMPLE2,
           BCAMAPPL          = UTMCPING, LISTENER-PORT = 9009,          -
           USER-FILEBASE     = $UTMUID.CLUST.SAMPLE2,                   -
           RESTART-TIMER-SEC = 180,                                     -
           FAILURE-CMD       =                                          -
               'FROM-FILE = $UTMUID.FAILURE, PROC-PAR = (%s,%s,%s,%s)'  -
           EMERGENCY-COMMAND =
               '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 = NODE1
CLUSTER-NODE FILEBASE  = $UTMUID.RESERV.SAMPLE2, HOSTNAME = RESERV,    -
             NODE-NAME = NODE1
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
```

Figure 28: Modified KDCDEF control statements

Adapt the database configuration

The configuration of the Oracle® client software on the mainframe is selected as described in the Figure 29 configuration file `tnsnames.ora` in which the connections to the Oracle® instances on `SERVDB` and `SERVDB2` are specified. Details about these configuration parameters are in [8], section 6. Two symmetric services `service12` and `service21` are defined. The file `tnsnames.ora` in Figure 29 is on both BS2000 servers, `SERV11` and `SERV12` or is provided in a shared pubset that can be accessed by both servers.

```
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)
```

Figure 29: Oracle® configuration file tnsnames.ora

Furthermore, the configurations of the Oracle® instances on `SERVDB` and `SERVDB2` must be modified. `racins1` and `racins2` are the respective Oracle instances on the two RAC nodes and `dbsampl2` is the database taken from the stand-alone application. The commands listed in Figure 30 are executed on one of the two Linux servers, e.g. on `SERVDB`.

```
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);
```

Figure 30: Configuring the Oracle RAC nodes

Load balancing

The last router before the former `SERV1` is now reconfigured as a load balancer whereby the IP address associated with `SERV1` is turned into the virtual IP address for load balancing. The former server `SERV1` is renamed to `SERV11` and is given a new real IP address.
Normally, the router to be changed is a normal commercially available router which offers load balancing functionality (if necessary as an additional function to be ordered separately). As we do not wish to prefer a certain make, we are demonstrating here the load balancing configuration based on the *Linux Virtual Servers* (LVS) which also offers this functionality, see Figure 31.
In the first `ipvsadm` command in Figure 31 `SERV1` is configured as a virtual service whereby round robin (rr) is selected as an example load balancing strategy. The next two `ipvsadm` commands configure the real servers belonging to this service. In all situations 102 is selected as

port. The option `-m` (or `--masquerading`) decides which packet forwarding method is used as well as which associated address modification; details can be found in [10]. Details about the `ipvsadm` commands can be found in [9] .

```
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
```

Figure 31: Load balancing configuration control commands for LVS

If a node fails and an automatic warm start is unsuccessful, this node should no longer be taken into account with load balancing. This is ensured by configuring a procedure `$UTMUID.EMERGENCY`, see Figure 28, which is automatically called by the cluster ring monitoring if a warm start is not successful within the 3 minutes specified by `RESTART-TIMER-SEC = 180`.
Such an emergency procedure is shown in Figure 32 . It sets the weighting of the failed node to 0 in the load balancing configuration which means that this node is temporarily not considered for new connection requests. As soon as the failed node has been repaired and powered up again, the administrator should reset the weight of the affected node to 1.
The emergency procedure shown in Figure 32 assumes that `LV` is the name of the load balancing processor and that administrator access has been set up via ID `lvsadm.` There are now several options to execute an action on the load balancing processor: use `rsh` or `ssh` under POSIX or use openFT. Figure 32 has selected the way `rsh` and POSIX as its method. The corresponding appropriate `rsh` commands are exported to separate command files `LVSADM-SERV11-WEIGHT-0` and `LVSADM-SERV12-WEIGHT-0` for simplification

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'
```

Figure 32: Emergency procedure with rsh command files

Start the UTM cluster application

It is assumed that a KDCUPD run has been executed on the node `SERV11`. The node application on this node must be started first using the start parameters listed in Figure 33 . As soon as the node application has been successfully started on `SERV11` (e.g. determined via WinAdmin "Availability *Check*"), the node application on `SERV12` can be started with the analogue start parameters (with `SERVICE12` replaced by `SERVICE21`). Details about the `.RMXA` start parameters for the Oracle® RAC configuration can be found in [4], section 7.4.2.

```
.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
```

Figure 33: Modified start parameters for the node applications

Node-by-node software update of a running application (online update)

It is now possible with UTM cluster applications to make an online system software update while the application is running and, for example, to install correction versions for system components. This is done node-by-node. For example, we assume that an upgrade from openUTM V6.3A00 to correction version V6.3A10 is to be done.
First of all, we start with one of the two nodes, e.g. with node `SERV11`. The weight for this node is set at 0 in the load balancer, e.g. with a load balancer command as shown in Figure 32. Termination of node application running on `SERV11` is invoked with `KDCSHUT WARN, TIME = 60, SCOPE = LOCAL`. This means that the users are notified that the node application will be shut down in one hour and that they should logoff by then. However, they can immediately login again after logging off; the load balancer then automatically connects them with the node application still active on `SERV12`. This ensures almost 100% non-interrupted operation of the cluster application.
As soon as the node application on `SERV11` is terminated, the software update can be carried out on this node. The node application on `SERV11` is then restarted and the weight for `SERV11` is reset to 1 in the load balancer. `SERV11` is thus taken into account when setting up new connections. Various UTM correction versions are now running temporarily on the two nodes which does not cause any problems if they are sufficiently compatible which is usually the case with a correction version.
The software update can then be carried out in the same way on the second node `SERV12`. The update is then implemented for the entire cluster application and all nodes are then running the same correction version V6.3A10 for openUTM.

Third example: UTM partner application – UTM application on BS2000

Starting Point: Stand-alone partner applications

This third example looks at distributed transaction processing with two UTM applications. It is based on the example on a travel booking system described in detail in the manual [1] section 6.7, whereby we will concentrate here on the main points. It is assumed that a travel agency operates an application called `TRAVEL` which is coupled via OSI TP with a reservation management system (`RMS`). `RMS` is a UTM application on a BS2000 server, `TRAVEL` is a UTM application on a UNIX system. The associated partner applications specified in [1] and the integrated database systems are not considered here in order to simplify matters. This simplified situation is seen in Figure 34.

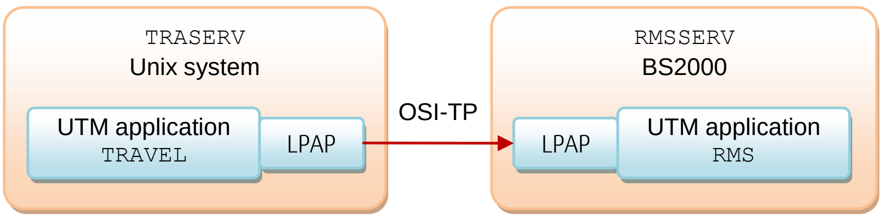

Figure 34: Stand-alone partner applications

Generate the stand-alone application TRAVEL

The main KDCDEF control statements to generate the UTM application `TRAVEL` are listed in Figure 35.

```
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 = ACTION1
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,                  -
```

Figure 35: KDCDEF control statements for TRAVEL

The main KDCDEF control statements for `RMS` are listed in Figure 36. For details, see the respective example in [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,                -
```

Figure 36: KDCDEF control statements for RMS

Convert the third example to cluster applications

Cluster architecture planned

To convert to a cluster architecture the UTM application `RMS` is turned into a UTM cluster application with two servers `RMSSERV1` and `RMSSERV2`. Likewise, the UTM application `TRAVEL` is turned into a cluster application with two servers `TRASERV1` and `TRASERV2`. Each of the two `TRAVEL` node applications communicates via an LPAP bundle with the two RMS node applications, see Figure 37.

Details about the conversion of associated databases and terminal pools are not provided here; the previous examples explained how these are converted to clusters.

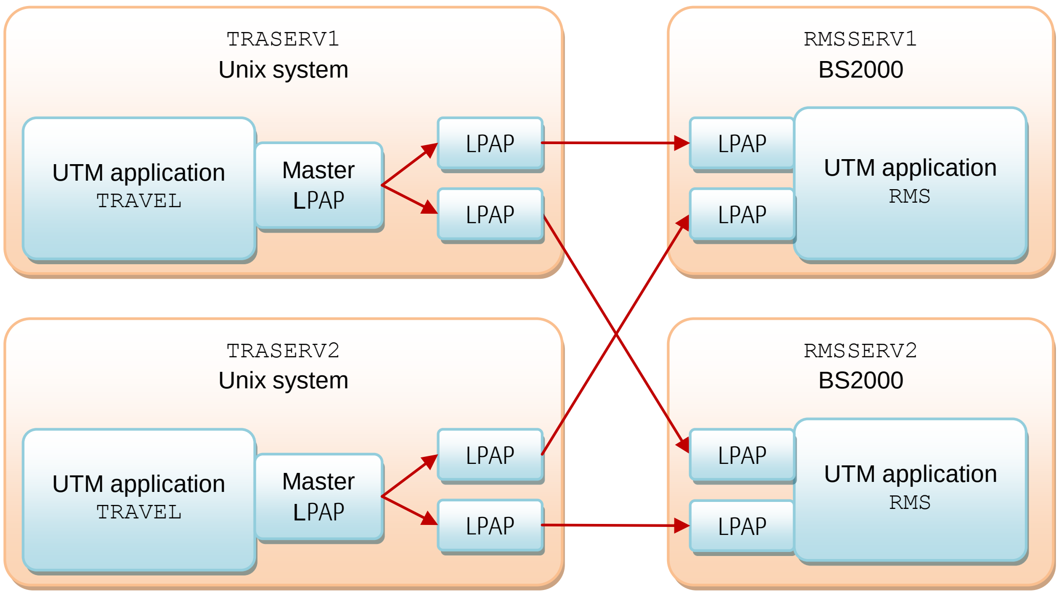

Figure 37: Cluster architecture planned

Generate the UTM cluster application TRAVEL

The stand-alone application is converted to a UTM cluster application (analogously to the first example) according to the steps described in [3], section 7.11.1. We concentrate on the conversion of the `TRAVEL` application equipped with LPAP bundle. If we assume one-sided communication where `TRAVEL` issues requests to `RMS`, no LPAP bundle needs to be generated on the RMS side, see Figure 37.

```
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-SYSTAX   = (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,          -
```

Figure 38: Modified KDCDEF control statements for TRAVEL

The KDCDEF control statements modified for the application TRAVEL are listed in Figure 38, the control statements modified for RMS in Figure 39, whereby the changes in contrast to the generation of the stand-alone application are again shown in blue.

```
OPTION     GEN = (CLUSTER, KDCFILE, ROOTSRC), ROOTSRC = SRC.RMSROOT
ROOT       RMSROOT
CLUSTER    CLUSTER-FILEBASE   = CLUST.RMSFILE,                          -
           BCAMAPPL           = UTMCPING, LISTENER-PORT = 9009,         -
           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    ACTION1,    COMP = ILCS
PROGRAM    ACTION2,    COMP = ILCS
PROGRAM    RMS,        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              = (C'STRV', ASCII), -
                       PRESENTATION-SELECTOR         = (C'PTRV', ASCII)
OSI-LPAP TRAVEL1,      ASSOCIATION-NAMES             = TRAVEL1,         -
                       ASSOCIATIONS                  = 4,               -
                       CONTWIN                       = 0,               -
                       APPLICATION-CONTEXT           = EUOSICCR,        -
                       APPLICATION-PROCESS-TITLE     = (1, 2, 3, 21)    -
```

Figure 39: Modified KDCDEF control statements for RMS

Variant for this example: LU6.1 instead of OSI-TP

It is assumed in a variant of the third example that the applications TRAVEL and RMS communicate with each other via LU6.1 and not via OSI-TP. Then all arranged parallel connections and sessions need to be explicitly listed. Regarding cluster applications, this means that connections and sessions for each node of one cluster application with each node of the other application must be defined. We can see in Figure 37 that our case has 4 groups of sessions. The example shows 4 parallel connections in each group between each node of the cluster application TRAVEL and each node of the cluster application RMS.
Node TRASERV1 can thus only use half of the connections associated with these sessions, while node TRASERV2 uses the other half. As of openUTM V6.2 this can be defined by specifying NODE-NAME in the LSES statement. NODE-NAME references the logical name of the node which is defined in the CLUSTER-NODE statement.

Figure 40 and Figure 41 show the `KDCDEF` control statements necessary for LU6.1 communication. Other control statements, as e.g. `CLUSTER, MAX, TAC, PROGRAM`, remain unchanged as in Figure 38 and Figure 39, resp., and are not repeated here.

```
...
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
*
```

Figure 40: KDCDEF control statements for `TRAVEL` in case of 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
*
```

Figure 41: KDCDEF control statements for `RMS` in case of LU6.1

Performance considerations

The cluster-wide effect of global storage areas ensured with openUTM can lead to more I/O per transaction when using these storage areas than was the case with the original stand-alone UTM application. This can also reduce the maximum achievable throughput (transactions per second). The performance of the network file system (NFS) is relevant, too. However, in most cases the throughput is not limited by openUTM itself but by the application and the database accesses. Since the specific effect greatly depends on the application architecture, it is hard to give precise figures here. Instead, it is recommended to perform load measurements with the specific application when a throughput of more than 50 transactions per second is expected.
On BS2000 openUTM makes intensive use of the *distributed lock manager* (DLM) provided with the XCS cluster. This DLM is based on token ring communication. To avoid delays during lock requests the NSM configuration setting for the token delay time should be adapted, e.g. to min/max = 0/1 ms, 0/2 ms, or 0/3 ms (command for min/max = 0/1 ms: `/MODIFY-NSM-ENVIRONMENT TOKEN-DELAY-TIME = *BY-PARAMETER (MIN-DELAY-TIME = 0, MAX-DELAY-TIME = 1`, see [6], chapter 9). These settings bring about an immediate token transfer during lock requests, and only slight transfer delay of an empty token, which, however, will cause higher CPU utilization in idle state. Which settings are optimal in a specific case very much depends on the application architecture. It is wise to find by experiment the optimal setting under heavy loads as a compromise between CPU utilization and throughput. Figure 42 schematically shows the effect of the delay time setting on idle CPU utilization and on maximum transaction throughput.

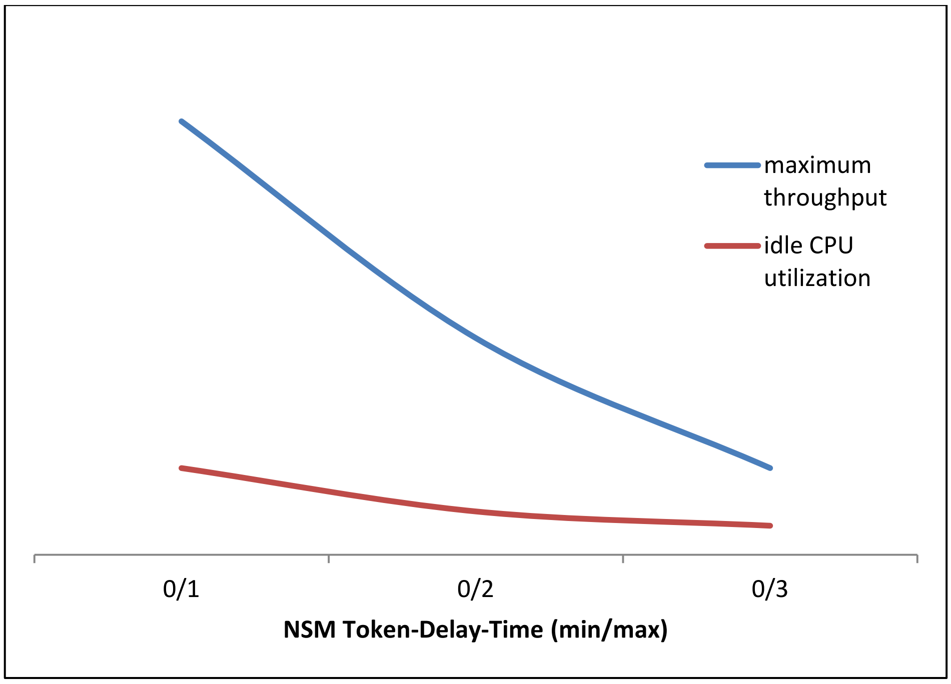

Figure 42: Effect of the NSM delay timer setting

Similarly as the overall performance of a multi processor server does not scale linearly with the number of processors, the overall performance of an UTM cluster application does not linearly scale with the number of nodes in the cluster. Precise figures cannot be given here, since they would not only be influenced by openUTM but would heavily depend on the application architecture and the database.

Summary

The various examples have shown how an existing stand-alone UTM application can be easily converted to a UTM cluster application. Specific customer situations will of course deviate from these examples that are of a general nature and which have been kept simple. However, the examples should be a good basis to understand specific conversions.

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