



**IBM HACMP による
Micro Focus Enterprise Server
クラスタリング検証報告**

平成 18 年 10 月 23 日
マイクロフォーカス株式会社

目次

はじめに.....	3
使用環境.....	4
使用したハードウェア.....	4
使用したソフトウェア構成.....	4
クラスタ構成.....	5
概要.....	6
Enterprise Server とテストシステム.....	6
クラスタリング.....	7
環境設定.....	8
手順.....	8
環境設定モデル.....	8
1. 事前準備.....	9
2. Micro Focus 製品の導入と Enterprise Server 環境の準備.....	9
3. COBOL コンポーネントの準備.....	9
4. Enterprise Server の設定.....	9
5. クライアントの作成.....	11
6. HACMP ユーザースクリプト.....	12
検証.....	15
検証方法.....	15
検証テストパターン.....	15
パターン① 手動切り替え.....	16
パターン② 電源障害による切り替え.....	18
パターン③ データ LAN アダプタ障害.....	19
パターン④ 業務障害検出による切り替え.....	21
パターン⑤ アプリケーション実行中の自動切り替え.....	23
パターン⑥ ES サーバ起動で待機.....	25
パターン⑦ サーバごとに別の業務を提供しながら待機.....	27
パターン⑧ デプロイパッケージの共有.....	31
パターン⑨ ログファイルの共有.....	35
テスト結果のまとめ.....	39
付録.....	40
1. サーバアプリケーションサンプルプログラム.....	40
2. クライアントアプリケーションサンプルプログラム.....	41
3. クライアントシェルスクリプト.....	42
4. アプリケーションデータ作成サンプルプログラム.....	42
5. killallcas.ksh シェルスクリプト.....	43
6. forceStopServer.ksh シェルスクリプト.....	48

はじめに

本書では、Micro Focus Enterprise Server (以下、Enterprise Server)で提供するアプリケーションサービスを IBM HACMP(以下、HACMP)により共有ディスクを利用してクラスタ上で動作・検証させた結果を報告します。

Enterprise Server は Windows、Unix、Linux 環境で COBOL サービスをトランザクショナルに管理・実行するアプリケーション サーバです。Enterprise Server は Micro Focus Net Express および Server Express COBOL コンパイラ ツール群の運用部分です。Enterprise Server はメインフレームのトランザクション環境と同等の OLTP 機能を、J2EE アプリケーションサーバと協調実行できるよう設計されています。ユーザーにとっては、ハイエンドのトランザクション システムで、機能およびパフォーマンスと同様に重要なことは、その環境がきわめて可用性、信頼性、保守性に優れていることです。

IBM の HACMP は、お客様のクリティカルなビジネスアプリケーションを障害から守るための製品です。多くのアプリケーション停止は、ハードウェア障害ではなく、ネットワーク、アプリケーション、その他の外部要因などが原因です。10 年以上にわたり、お客様の IBM eServer pSeries サーバやネットワーク接続の監視、バックアップサーバへのアプリケーションのフェールオーバーなど、信頼できる高可用性サービスを提供し続けています。

HACMPとEnterprise Serverを組み合わせ、如何に信頼性・可用性の高いCOBOLサービスシステムを提供していくか、その方法について検証します。

注意事項

本書に記載されている内容は、検証環境での評価システムでの情報であり、実際のシステム環境では、それぞれ個別に評価をし、固有の設定が必要になります。本書に記載されている内容は、参考値であり、実際のシステムでの動作を保証するものではありません。

このドキュメントの内容は予告なしに変更される場合があります。このドキュメントの内容の保証や、商品性又は特定目的への適合性の黙示的な保証や条件を含め明示的又は黙示的な保証や条件は一切無いものとします。マイクロフォーカス株式会社および日本アイ・ビー・エム株式会社は、このドキュメントについていかなる責任も負いません。また、このドキュメントによって直接又は間接にいかなる契約上の義務も負うものではありません。このドキュメントを形式、手段(電子的又は機械的)、目的に関係なく、マイクロフォーカス社および日本アイ・ビー・エム株式会社の書面による事前の承諾なく、複製又は転載することはできません。

Copyright © 2006 Micro Focus. All Rights Reserved.

Micro Focus は Micro Focus 社の登録商標、Enterprise Server, Server Express は同社の商標です。

HACMPは、IBM Corporationの商標です。

その他記載の会社名、製品名は、各社の商標または登録商標です。

使用環境

使用したハードウェア

[サーバハードウェア]

System p5 550, POWER5 1.65GHz 4core, 8GB memory

[host1 (LPAR)]

CPU: 2core

メモリ: 3.5GB

内蔵ディスク: 36GB x2

アダプタ: 1Gbps Ethernet x3 (1 つは管理用、2 つが業務用), シリアル, SCSI

AIX 5L V5.3 TL04 SP03 (2006 年 5 月)

HACMP V5.3 (2006 年 5 月 PTF)

[host2 (LPAR)]

CPU: 2core

メモリ: 3.5GB

内蔵ディスク: 36GB x2

アダプタ: 1Gbps Ethernet x3 (1 つは管理用、2 つが業務用), シリアル, SCSI

AIX 5L V5.3 TL04 SP03 (2006 年 5 月)

HACMP V5.3 (2006 年 5 月 PTF)

*: System p5 の LPAR はファームウェアレベルで完全に分離されており、カーネル間の依存関係もありませんので、1 台の独立したサーバとみなすことができます。

[共有ディスク]

2104-TS4 (Ultra320 SCSI 36GB HDD x10)

使用したソフトウェア構成

[host1]

AIX 5L V5.3 TL04 SP03 (2006 年 5 月)

HACMP V5.3 (2006 年 5 月 PTF)

Micro Focus Server Express 4.0J for AIX SP2

(開発環境,Fixpack40.02_59)

Micro Focus Enterprise Server 4.0J for AIX SP2

(実行環境,Fixpack40.02_59)

Java™ 2 SDK, Standard Edition 1.4.2_8

j2ee.jar

[host2]

AIX 5L V5.3 TL04 SP03 (2006 年 5 月)

HACMP V5.3 (2006 年 5 月 PTF)

Micro Focus Enterprise Server 4.0J for AIX SP2

(実行環境,Fixpack40.02_59)

Java™ 2 SDK, Standard Edition 1.4.2_8

j2ee.jar

[クライアント]

WindowsXP

Java™ 2 SDK, Standard Edition 1.4.2_8

Net Express4.0 fix:ALL07N40 + jpn07n40

j2ee.jar

クラスタ構成

[HACMP 設定]

2 ネットワークアダプタ構成
非コンカレント VG
自動フェイルバック(切り戻し)なし

host1:

リソースグループ 1 の優先ノード, リソースグループ 2 のスタンバイノード

host2:

リソースグループ 2 の優先ノード, リソースグループ 1 のスタンバイノード

リソースグループ 1 (ESSRV01):

再配置可能 IP アドレス: 10.0.0.1

共有ディスク: /u01

リソースグループ 2 (ESSRV02):

再配置可能 IP アドレス: 10.0.0.2

共有ディスク: /u02

共有ディスク

sharevg1 : /u01

アプリケーションデータ#1、ディプロイパッケージ、
ログディレクトリ

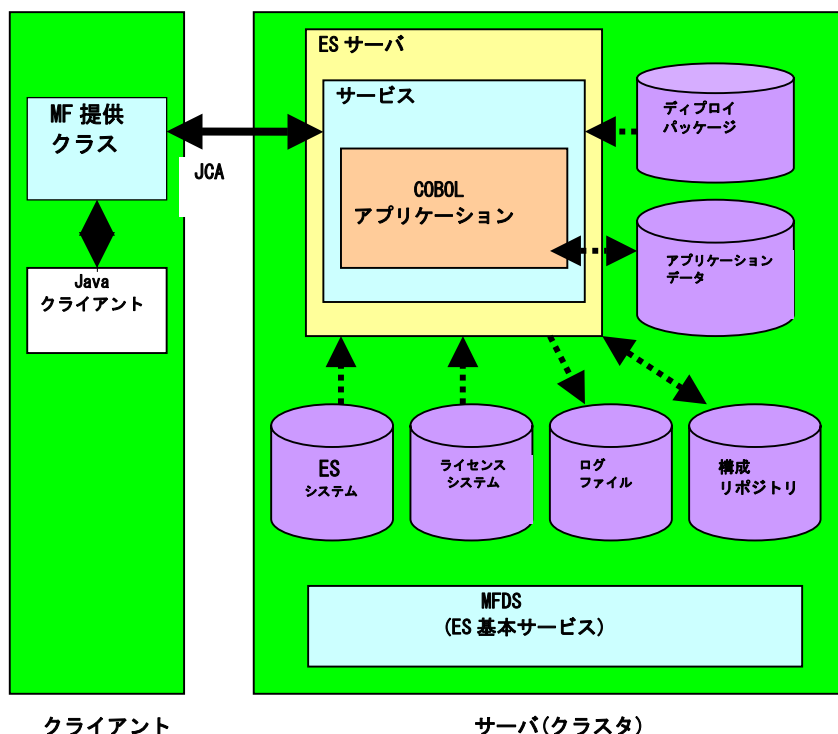
sharevg2 : /u02

アプリケーションデータ#2

概要

Enterprise Server とテストシステム

今回のテストで Enterprise Server に実装するサンプルモデルを用意しました。



Java クライアントから要求されたリクエストは Enterprise Server 上にディプロイされたサービスの COBOL アプリケーションに引き渡されます。この COBOL アプリケーションは索引ファイルであるアプリケーションデータから連番を取得し、この連番をカウントアップして書き込みします。そして日時、プログラム名、実行ホスト名、連番を返します。Java クライアントはこれを表示して終了します。

以下に関連する Enterprise Server の構成要素について説明します。

(プロセス関連)

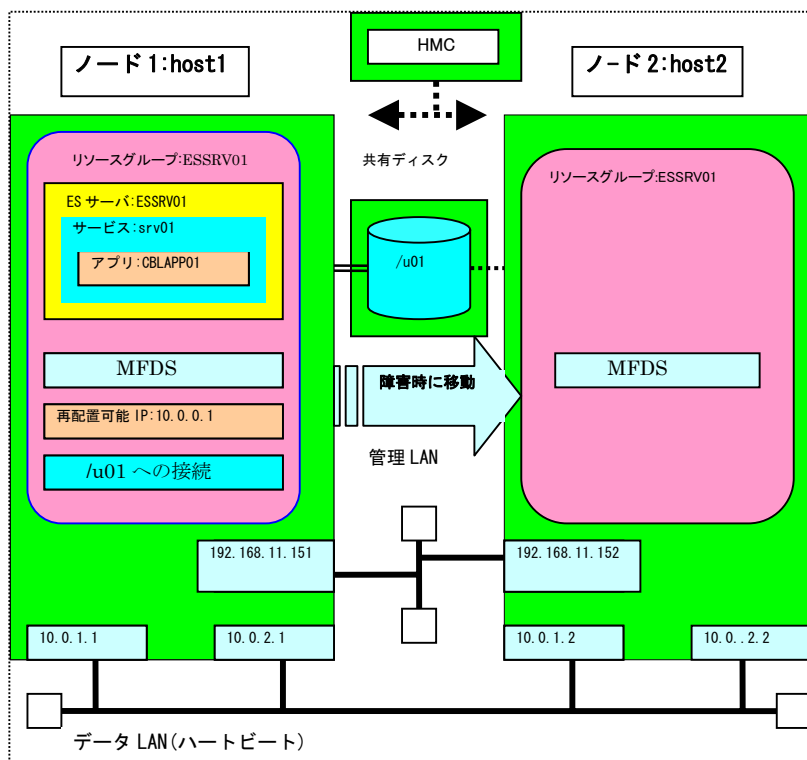
- ・MFDS Enterprise Server の基本サービスの一つ。1 ノードに1つだけ起動が必要
- ・ES サーバ Enterprise Server の管理単位。1 つのノードに複数定義でき、複数のサービスを含むことができる。
- ・サービス Enterprise Server で提供する COBOL サービスであり、ディプロイの単位。
- ・COBOL アプリケーション COBOL プログラムが提供する業務プロセス。
- ・MF 提供クラス クライアントから Enterprise Server に接続するために利用
- ・Java クライアント コマンドライン起動で Enterprise Server にリクエストを出し受け取った結果を表示する。

(ファイル関連)

- ・ES システム Enterprise Server のシステムコマンド、ライブラリなどのファイルを含む。
\$COBDIR 環境変数がポイントするディレクトリパス。共有可。
- ・ライセンスシステム Enterprise Server のライセンスデータベースを管理する。このディレクトリをディスクコピーなどで移動・復元することはできない。共有不可。
- ・ログファイル Enterprise Server のログ。ES サーバ単位で管理され、再起動によって自動的に新しいファイルに書き換えられる。共有可。
- ・構成リポジトリ Enterprise Server の設定情報を格納したファイル群。共有可だが更新モードでオープンされるため同時アクセスはできない。

- ・ディプロイパッケージ .car ファイル。ディプロイに必要なファイル群。実装する際のディプロイパッケージは、Interface Mapping Tool Kit(IMTK)により COBOL ソースから容易に生成することが出来る。Server Express によってコンパイルされた実行形式のファイルを含む。(使用したソースサンプルは付録に記載) 共有可。
- ・アプリケーションデータ COBOL プログラムが利用するユーザーファイル。共有可。
- ・Java クライアント Java プログラム。(ソースサンプルは付録に記載)
- ・MF 提供クラス クライアントプログラムが利用する Enterprise Server 接続用の Java クラス。

クラスタリング



(注 HMC: ハードウェア管理コンソール)

HACMP ではテークオーバーする一連の操作をリソースグループの単位で管理します。リソースグループには、再配置可能IPアドレス(仮想IPアドレス)や共有ディスクの有効化、アプリケーションの起動・停止・監視の処理を設定します。再配置可能IPアドレスや共有ディスクはリソースグループが立ち上がると自動的に有効化され停止すると無効化します。

MFDSやESサーバなどの起動・停止はそれぞれのシェルにコマンドで処理を記述します。本運用ではこの起動・停止のシェルには障害時の状態回復・状態保存を行なう処理なども組み込む必要があります。またアプリケーションの監視は、障害とする事象が発生するまで監視を繰り返す業務監視用のシェルをユーザーが準備しこれを組み込む必要があります。

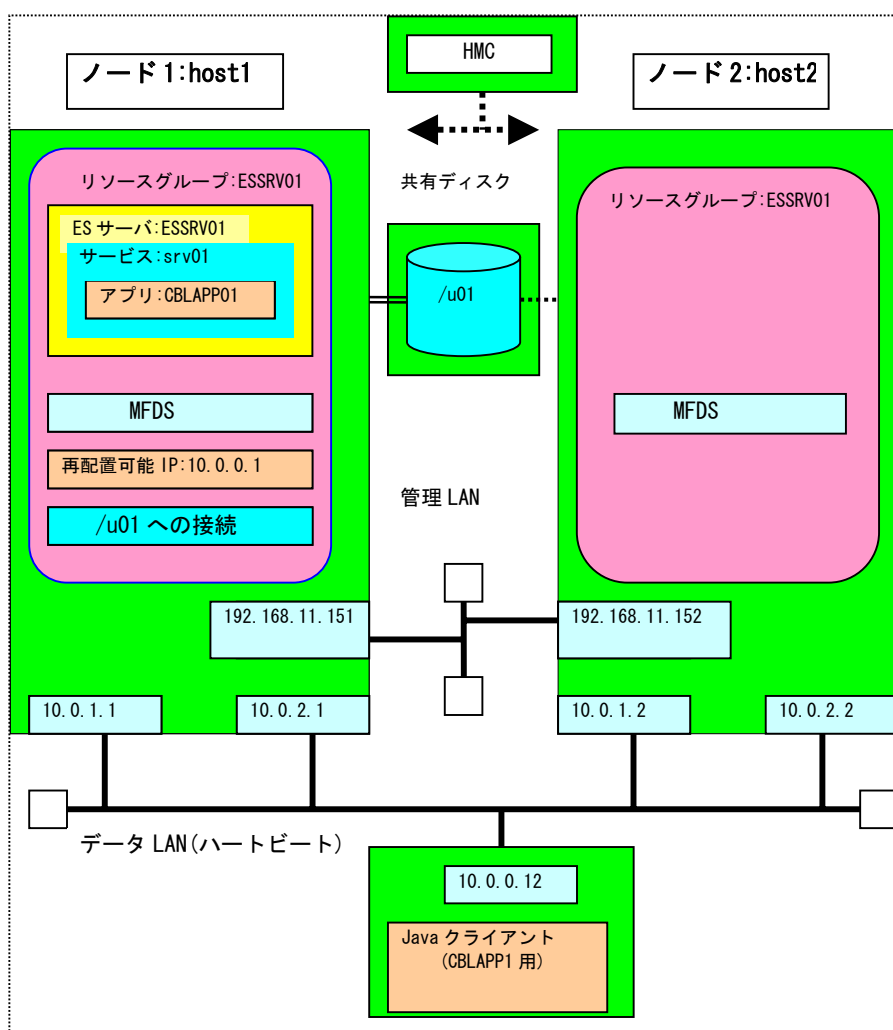
環境設定

下図のモデルを想定した場合の環境の HACMP、MF 製品の設定例を記述します。

手順

1. 事前準備
2. Micro Focus 製品の導入と Enterprise Server 環境の準備
3. COBOL コンポーネントの準備
4. Enterprise Server の設定
5. クライアントの作成
6. HACMP ユーザースクリプト

環境設定モデル



1. 事前準備

各ノードに搭載されたネットワークアダプタの設定、共有ファイルの接続に関する事前の設定やその他の OS レベルの設定を事前に済ませておきます。また、各ノードごとに HACMP V5.3 を導入しておきます。

2. Micro Focus 製品の導入と Enterprise Server 環境の準備

1) 開発環境の準備

• Server Express4.0 SP2 及び Fixpack40.02_59

ノード 1 へ導入

インストールディレクトリ: /opt/mf/SE40SP2

2) 実行環境の準備

• Enterprise Server4.0 SP2 及び Fixpack40.02_59

ノード 1、ノード 2 それぞれに導入。

インストールディレクトリ: /opt/mf/ES40SP2

3. COBOL コンポーネントの準備

Enterprise Server に実装するコンポーネントの準備を行いません。

作業は開発環境にて行いません。本検証では Server Express が提供する Interface Mapping Tool Kit を使って一連の作業を簡略化しています。

初めに COBOL ソースをもとにサービスに対するデフォルトのマッピングを作成します。サービス名は任意に命名します。付録にサンプルを掲載しています。

```
imtkmake -defmap src=cblapp01.cbl service=srv01 type=ejb
```

これで "srv01.xml" の名称でデフォルトのマッピング情報が生成されます。

デフォルトではこのサービスのパラメータはプログラム中の連絡節のパラメータ名で全て "io" (入出力) として生成されています。本サンプルでは出力パラメータのみであるため実態に合わせ全て "out" に手修正しました。そして次のコマンドでディプロイパッケージを生成します。

```
imtkmake -generate service=srv01 type=ejb
```

これで "srv01.car" ファイルが生成されます。

更にここで同じ COBOL ソースを cob コマンドでコンパイルし、実行形式ファイル(本検証では gnt 形式を選択)を生成しておきます。

またこの実行形式ファイルが使用する索引ファイルを (APPFILE01, APPFILE01.idx)、initfile1.cbl をコンパイル・実行させて作成・準備します。

また両ノード間での双方向テークオーバーテスト用に同じ内容の別名のファイル、

srv02.car, cblapp02.gnt, APPFILE02.idx, APPFILE02 も上記手順で準備(パターン⑦で使用)しました。

4. Enterprise Server の設定

ノード 1、ノード 2 で Enterprise Server の環境を設定し入り下記の作業を行います。

1) /etc/services へ次の 2 行を追加します

```
mfcobol      86/tcp
```

```
mfcobol      86/udp
```

2) mfdss コマンドでディレクトリサーバを起動します

```
mfdss &
```

3) Enterprise Server Administration 画面のオープン

Web ブラウザより http://<IP アドレス>:86/ を指定すると Enterprise Server Administration 画面が開きます。ここでサーバに対して多くの操作を実行できるようになります。

4) サーバ定義

両ノード共デフォルトの ESDEMO を削除し新しく ESSRV01 を定義します。サーバ名称以外はデフォルト設定で作成します。

5) サービスのディプロイ

次にディプロイパッケージを使ってサービスの登録を行ないます。

<Enterprise Server インストールディレクトリ>/deploy の下に任意のディプロイ用ディレクトリ (例えば srv01) を作成し、その下に 3 で作成した car ファイル、実行形式ファイル、アプリケーションデータをコピーします。

<Enterprise Server インストールディレクトリ>/deploy/.mfdeploy を以下の様に書き換えます。

```
MFDS=localhost
MFUS_SERVER=ESSRV01  <<-- 修正
MFUS_LISTENER=Web Services
```

<Enterprise Server インストールディレクトリ>/deploy の下のディプロイ用ディレクトリに移動して下記のコマンドを実行します。

```
cd srv0
mfdepinst srv01.car
```

すると mfdeploy に記載した ES サーバ上に car ファイル作成時に指定した名称でサービスが作成されます。ディプロイ処理の結果は deploy.txt というファイルで確認することができます。登録された ES サーバは開始(Start)ボタンで手動で開始できます。

<ノード 1>

ESSRV01 を起動すると下記のように表示されパッケージにサービスがひとつ追加されたことがわかります

タイプ	名前	現ステータス	通信プロセス	ライセンス	ステータスログ	オブジェクト	説明
MFES	ESSRV01	開始	1.1.1.10.0.0.1*33175* (host1*)	* / 10	MD53800E Server started successfully 20-05-15 9 minutes 7 seconds in "開始" state since 20-05-15	3 サービス 2 ハンドラ 1 パッケージ	Communications server for Web Services

<ノード 1 パッケージの状態>

登録されたパッケージの内容が下記のように表示されます

名前	現ステータス	ステータスログ	パッケージモジュール	IDT	パッケージパス	カスタム構成	説明
srv01.CBLAPP01	Available	OK		Au01/deplo/srv01/srv01.idt	Au01/deplo/srv01		created 20-02-26 Thu 2006-08-03 from srv01/srv01.car

<サービスの内容>

下記のようにサービスが登録されます

Server ESSRV01 [開始]

サーバー... リスナー (2) サービス (3) ハンドラ (2) パッケージ (1)

サービス表示フィルタ ネームスペース: オペレーション: クラス: All ハンドラ

1 - 3 of 3 displayable namespaces from a total of 3 Show 10 service namespa

サービス ネームス ペース	オペレーショ ン	サービス クラス	探索 順序	リスナー	要求 ハンドラ	実装 パッケージ	現 ステータ ス	ステ ータ ス ログ	カスタム 構成
Deployer	Deployer 編集...	MF deployment	1	1 CP 1 Web top:172.16.0.104*:47055* (host1 +)			Available	OK	[MF client] scheme=http URL=c{ accept=application/x-zip-compres listener=Web Services
ES	ES 編集...	MF ES	1	1 CP 1 Web Services top:172.16.0.104*:47055* (host1 +)			Available	OK	
削除	srv01	1 of 1 operations shown							
	.CBLAPP01 編集...		1	1 CP 1 Web Services top:172.16.0.104*:47055* (host1 +)	MFRHBINP	srv01.CBLAPP01	Available	OK	

追加

上記の登録はノード 2 側でも同様に確認できます

6) リスナーの設定

ESSRV01 を再度停止しリスナーの画面の編集(Edit)ボタンで編集画面を開き”Web Services”の中に”*:*”で設定されている IP アドレス:ポート番号を固定で設定します。ここで設定する IP アドレスはテークオーバーで移動される再配置可能 IP アドレスを指定します。

今回のケースでは下記のように設定しました。

ES サーバ ESSRV01 のリスナー: 10.0.0.1:9003

ESSRV02 のリスナー(パターン⑦のみ): 10.0.0.2:9003

Server ESSRV01 [開始]

サーバー... リスナー (2) サービス (3) ハンドラ (2) パッケージ (1) 画面更新 自動更新間隔 (秒)

リスナー表示フィルタ プロセス: All 会話タイプ: All ステータス: All

通信プロセス 1 自動起動 統計 Stop コピー...

リスナー	プロセスID	コントロールチャンネルアドレス	ステータス	前回のステータス変更	ステータスログ	パ
編集... 2 追加	process 848040	tcp:10.0.0.1*:34621* (host1 +)	開始	07/03/06-19:15:17	OK	1.1

名前	アドレス	ステータス	前回のステータス変更	ステータスログ	会話タイプ	カスタム構成	説明
編集... Web Services	tcp:10.0.0.1:9003 (host1 +)	開始	07/03/06-19:15:16	OK	Web Services and J2EE		Wel via HTT
編集... Web	tcp:10.0.0.1:34130* (host1 +)	開始	07/03/06-19:15:17	OK	Web	[virtual paths] cgi=<ES>/bin uploads=<ES>/deploy docs=<ES>/docs/html	Bas web

7) ESSRV01 の起動

Enterprise Server Administration 画面で”開始”ボタンを押し立ち上げます。

5. クライアントの作成

本検証で Enterprise Server 上にデプロイしたサービスにリクエストを出すのは Java のクライアントプログラム (cbltest.java)です。(ソースサンプルは付録に記載)

java クライアントプログラム及び Net Express を PC に導入・準備し java の CLASSPATH に下記のファイルを指定します。

J2ee.jar
%COBDIR%\lib\mfconnector.jar
%COBDIR%\lib\mfcbolpure.jar

次に java クライアントを javac コマンドで準備します。

Javac cbltest.java

クライアントプログラムは J2EE API を介して、デプロイされている COBOL サービスを呼び出し、返される日付、時間、実行プログラム名、ホスト名、連続番号を画面に表示します。
これによって接続先の確認、トランザクションの連続性を検証することができます。

6. HACMP ユーザースクリプト

テークオーバーの自動化のため HACMP 制御下で呼び出される三つのスクリプトを準備しました。
それぞれ、起動、停止、モニタリングスクリプトで下記のような処理を行います。通常これらはアプリケーションの要件に応じてカスタマイズする必要があります。

各ノード共 mfdss を立ち上げた状態でテークオーバーテストを実施しました。

1) mfstart.ksh

テークオーバー時に新しいノードでの起動処理をおこないます。その内容は下記の通りです。

- Enterprise Server 環境の設定
- Enterprise Server の残プロセスのクリーニング処理(kill)
(注: パターン⑦での他サーバのプロセスの不用意な kill 終了を避けるため
kill 処理は対象サーバ名を指定して行う)
- サーバ ESSRV01 の開始
casstart /rESSRV01

```
#!/bin/ksh
. /home/user2/ES40SP2
/home/user2/killallcas.ksh ESSRV01 >> ESSRV01.cntl.log
sleep 3
/opt/mf/ES40SP2/bin/casstart /rESSRV0
```

注: killallcas.ksh については付録を参照

2) mfhalt.ksh

テークオーバー時に停止するノードでの終了処理を行います。

- Enterprise Server 環境の設定
- サーバ ESSRV01 の停止
forceStopServer.ksh シェルスクリプトを呼び出し立ち上がっているサーバを Enterprise Server
の mfdump ユーティリティを使用してチェックして casstop コマンドで停止します。
サーバプロセス終了を確認しサーバプロセス残が生じたら casstop /force /rESSRV01 を発行します。

```
#!/bin/ksh
. /home/user2/ES40SP2
/home/user2/forceStopServer.ksh> forceStopServer.msg 2>> forceStopServer.msg
```

注: forceStopServer.ksh については付録を参照

3) mfmonitor.ksh

Enterprise Server 環境をモニターしアプリケーション実行環境に異常が生じた時
ゼロ以外のリターンコードを HACMP に返しテークオーバーを発生させます。

- Enterprise Server 環境の設定
- Enterprise Server の mdump ユーティリティによりサーバの状態を採取してサーバダウンやリスナーの
停止を監視し障害時にゼロ以外のリターンコードを返す。

```

#!/bin/ksh
. /home/user2/ES40SP2
PATH=$PATH:/usr/bin

TMPFILE=/tmp/$$tmp.txt

mdump_func()
{
$COBDIR/bin/mdump localhost ESSRV01 > $TMPFILE 2> /dev/null
web_srv_flg=0
/usr/bin/cat $TMPFILE | while read line
do
    # if mlds not started "SERVER_DOWN" found
    # actual text is 'm_ldap_bind: 0x51 SERVER_DOWN'
    tmp=`echo $line | fgrep "SERVER_DOWN"`
    if [ ".$tmp" != "." ] ; then
        # echo $line
        return 3
    fi

    tmp=`echo $line | fgrep "CN:"`
    if [ ".$tmp" != "." ] ; then
        tmp=`echo $line | fgrep "Web Services"`
        if [ ".$tmp" != "." ] ; then
            # echo $line
            web_srv_flg=1
        else
            # echo "no " $line
            web_srv_flg=0
        fi
    fi

    if [ $web_srv_flg -eq 1 ] ; then
        lisnr=`echo $line | fgrep "mListenerStatus:"`
        if [ ".$lisnr" != "." ] ; then
            lisnr=`echo $line | fgrep "Started"`
            if [ ".$lisnr" != "." ] ; then
                return 0
            else
                # echo $line
                return 1
            fi
        fi
    fi
done
}

```

```
mdump_func
res=$?
/usr/bin/rm -f $TMPFILE
if [ "$res" = "1" ]; then
    echo "**** MF Service Failed ****" >> /tmp/ESSRV01.cntl.log
    exit 1
fi
if [ "$res" = "3" ]; then
    echo "**** MF mfd's Down ****" >> /tmp/ESSRV01.cntl.log
    exit 1
fi
echo $res
```

検証

検証方法

ここでは相互テークオーバーの条件化で2ノード間でのテークオーバーを確認します。

- ① スタンバイ側の Enterprise Server Administration 画面によるステータスの遷移の確認を行います。
テークオーバーの発生により制御はスタンバイ側のノードに移ります。
スタンバイ側の Enterprise Server Administration 画面のステータスが“停止”から“開始”移ることでノードの移動を確認します。また、テークオーバーされるノードは制御を失い“停止”または“応答無し”の状態に変わります。
- ② java クライアントが受け取り表示するホスト名及び連続番号による確認
java クライアントから COBOL を呼び出すたびに COBOL サービスから java に対して日付、時間、実行プログラム名、ホスト名、連続番号を返し java クライアントはこの情報を画面に表示するようにしています。
これにより接続先の確認と実行トランザクションの連続性を確認することができます。テークオーバー時の HACMP による一時的な中断をはさみテークオーバー先での接続の再開と実行の継続を確認することができます。

今回上記の2点を確認することで各テストパターンが正しく行われたことを検証しました。

検証テストパターン

パターン	タイトル	説明
①	手動切り替え	HACMPコマンド機能で切り替えを確認
②	電源障害による切り替え	LPARの電源オフ機能による切り替えを確認
③	データLANアダプタ障害 (二重)	2本のデータLANケーブルを抜くことによるLAN障害での切り替えを確認
④	業務障害検出による 切り替え	業務監視用シェル(mfmonitor.ksh)での異常検出による切り替えを確認
⑤	アプリケーション実行中 の自動切り替え	テークオーバーによる切り替えに伴う業務の継続確認
⑥	ESサーバ起動で待機	待機状態のノードではMFDS、ESサーバを起動状態で切り替えを確認
⑦	サーバ毎に別の業務を 提供しながら待機 (相互スタンバイ)	待機状態のノードではMFDS起動、切り替えるESサーバとは別のESサーバを動作させながら切り替えを確認。ノード1⇄ノード2の双方向で確認
⑧	ディプロイパッケージの 共有	ディプロイするパッケージを共有化し、切り替え後に同じパッケージを引き継ぐことを確認
⑨	ログ出力先の共有	ログファイルの出力先を共有化し、切り替え後のログ照会ができる事を確認

①～④は、テークオーバーのトリガーのテストです。代表的なトリガー4種類のパターンで確認します。

⑤～⑦は、待機側のノードの状態を変えるテストです。本検証ではEnterprise Serverのプロセスの実行状態を変えて確認します。Javaクライアントから繰り返しリクエストを出し続け、テークオーバー後もサービスが提供され続ける事を確認します。

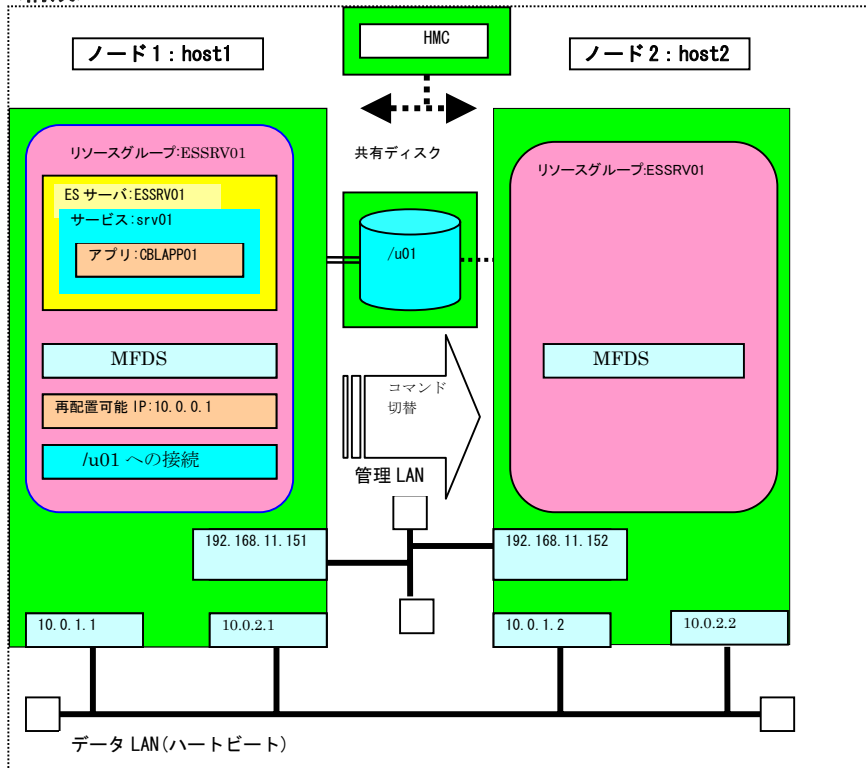
⑧～⑨は、Enterprise Serverシステムが利用する情報を共有するテストです。

パターン① 手動切り替え

<目的>

ノード1を ONLINE、ノード2を OFFLINE にしておいて HACMP コマンドによってノード1側からノード2側に手動テークオーバーできることを検証します。このテストは計画停止などで一般に行われるテークオーバーを検証します。

<構成>



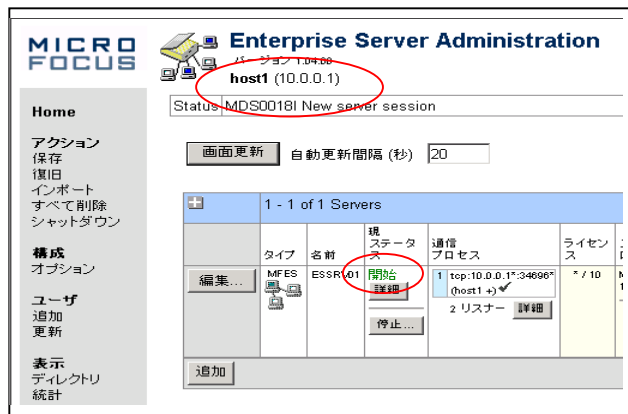
<検証>

各ノードに接続した Enterprise Server Administration 画面ノード1側(画面1)ノード2側(画面2)の画面変化によってテークオーバーを確認します。

画面1: 開始ステータス、ホスト名: host1(テークオーバー前)

この画面はノード1側の Enterprise Server の状態を示しています。

HACMP コマンドによりノード1->ノード2のテークオーバーが発生するとノード1は制御を失います。




画面 2: 停止ステータス、ホスト名:host2(テークオーバー前)




<中断>

HACMP によるノード切り替えに伴いノード 1 側のサービスが停止します。

 ページを表示できません

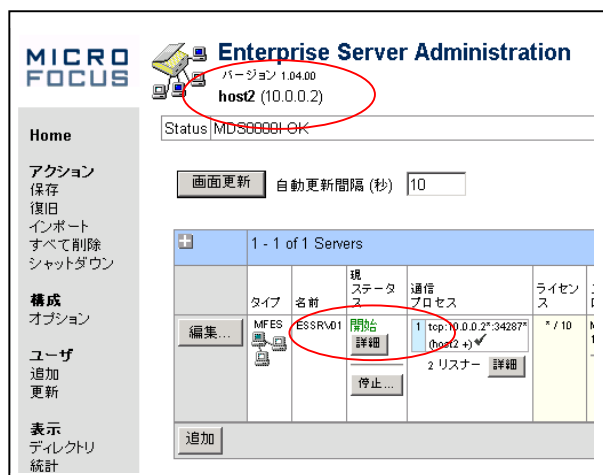
検索中のページは現在、利用できません。Web サイトに技術的な問題が発生しているか、ブラウザの設定を調整する必要があります。

次のことを試してください。

-  [更新] ボタンをクリックするか、後でやり直してください。
- アドレスバーにページアドレスを入力した場合は、ページアドレスを正しく入力したかどうかを確認してください。
- 接続の設定を確認するには、[ツール]メニューの [インターネット オプション] をクリックします。[接続] タブで [ダイヤルアップの設定] グループの [設定] ボタン、または [LAN の設定] グループの [LAN の設定] ボタンをクリックしてください。設定情報は、LAN (ローカルエリアネットワーク) の管理者か、ISP (インターネット サービス プロバイダ) が提供する情報と一致する必要があります。
- ネットワーク管理者がネットワークの接続の設定を使用可能にしている場合は、Microsoft Windows を使用して、ネットワークの接続試験を行ったり、自動的にネットワークの接続の設定を見つけることができます。

画面 2: 開始ステータス、ホスト名:host2(テークオーバー後)

テークオーバーにより制御がノード 2 側に移り開始ステータスに変わります。



<結果>

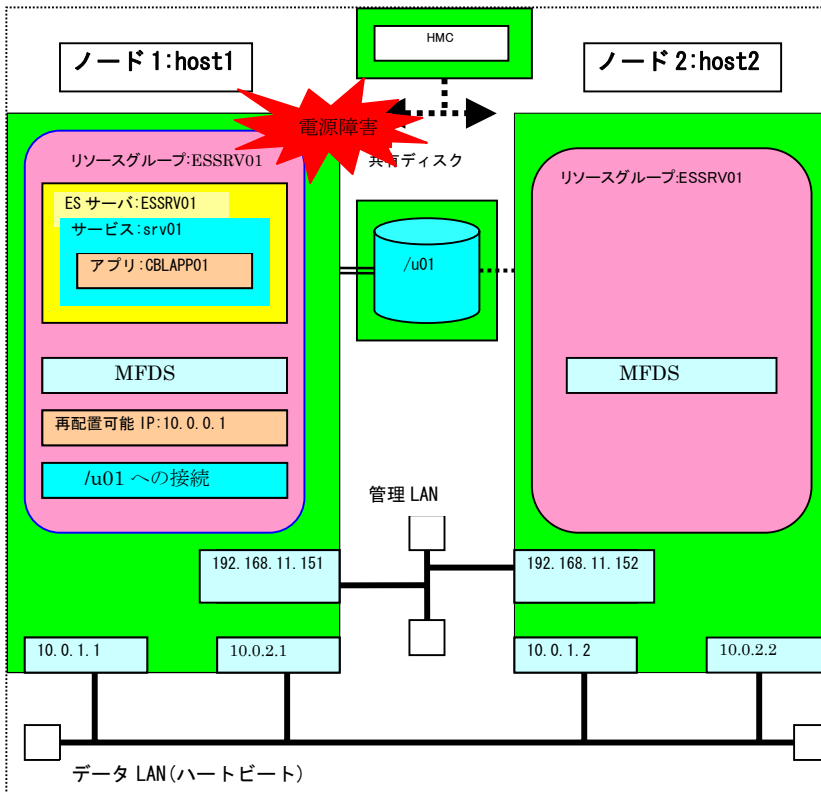
テークオーバーの結果ノード 1->2 への制御移動が発生したことが Enterprise Server Administration 画面で確認できました。

パターン② 電源障害による切り替え

<目的>

ここでは電源障害が生じたときのテークオーバー機能を検証します。
ここではテークオーバーのトリガーを LPAR の HMC(ハードウェア管理コンソール)のパワーオフ機能でおこないます。

<構成>



<検証>


HMC から LPAR の強制パワーオフ (模擬電源障害) を実行するとノード 1 側が制御を失いノード 2 に制御が移ることを確認します。

<テークオーバー前(開始状態)>




<中断>

パワーオフを HACMP が検知しノード 1 のサービスが停止します。

 ページを表示できません

検索中のページは現在、利用できません。Web サイトに技術的な問題が発生しているか、ブラウザの設定を調整する必要があります。

次のことを試してください。

-  [更新] ボタンをクリックするか、後でやり直してください。
- アドレス バーにページ アドレスを入力した場合は、ページ アドレスを正しく入力したかどうかを確認してください。
- 接続の設定を確認するには、[ツール] メニューの [インターネット オプション] をクリックします。[接続] タブで [ダイヤルアップの設定] グループの [設定] ボタン、または [LAN の設定] グループの [LAN の設定] ボタンをクリックしてください。設定情報は、LAN (ローカル エリアネットワーク) の管理者か、ISP (インターネット サービス プロバイダ) が提供する情報と一致する必要があります。
- ネットワーク管理者がネットワークの接続の設定を使用可能にしているか、Microsoft Windows を使用して、ネットワークの接続試験を行ったり、自動的にネットワークの接続の設定を見つけることができます。

<テークオーバー後>

ノード 2 に制御が移り開始状態に変わります。



The screenshot shows the 'Enterprise Server Administration' window. At the top, the server name 'host2 (10.0.0.2)' is circled in red. Below, a table lists server details. The first row is highlighted, and the '開始' (Start) button in the '操作' (Action) column is also circled in red.

タイプ	名前	現ステータス	通信プロセス	ライセンス
MFES	ESSR01	開始	1 top:10.0.0.2*:34424* (host2 +)	* / 10
		停止...	2 リスナー	詳細

<結果>

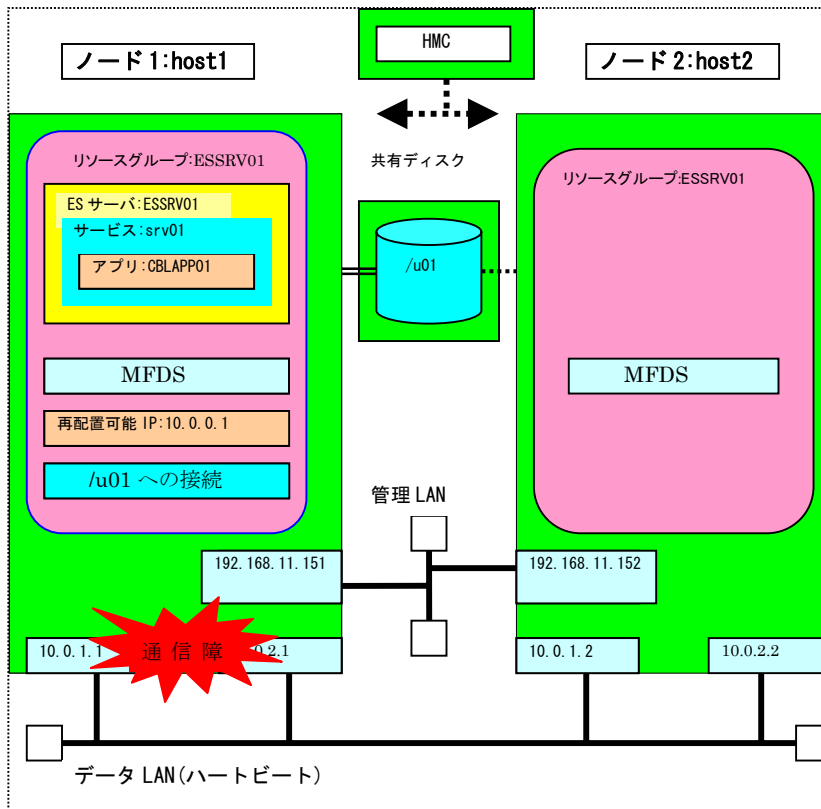
電源をオフにすることによりテークオーバーが発生しノード 1->2 に制御が移ることが確認できました。

パターン③ データ LAN アダプタ障害

<目的>

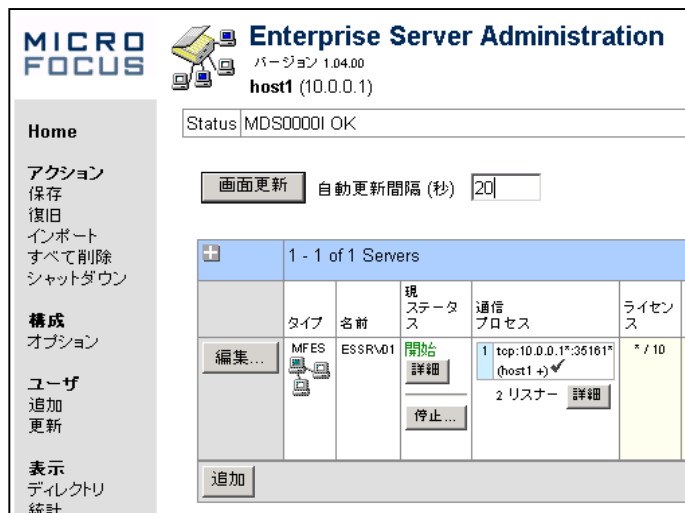
ここでは LAN 障害に伴うテークオーバー動作を確認します。LAN 障害は二本の LAN ケーブルを抜き取ることによって発生させます。

<構成>




<検証>

Host1(ノード 1)の二本の LAN ケーブルを抜きます。




<テークオーバー時の中断>

HACMP がネットワークアダプタ障害を検知しノード 1 のサービスが停止になります。

 ページを表示できません


検索中のページは現在、利用できません。Web サイトに技術的な問題が発生しているか、ブラウザの設定を調整する必要があります。

次のことを試してください:

-  [更新] ボタンをクリックするか、後でやり直してください。
- アドレスバーにページ アドレスを入力した場合は、ページ アドレスを正しく入力したかどうかを確認してください。
- 接続の設定を確認するには、[ツール] メニューの [インターネット オプション] をクリックします。[接続] タブで [ダイヤルアップの設定] グループの [設定] ボタン、または [LAN の設定] グループの [LAN の設定] ボタンをクリックしてください。設定情報は、LAN (ローカルエリアネットワーク) の管理者か、ISP (インターネット サービス プロバイダ) が提供する情報と一致する必要があります。
- ネットワーク管理者がネットワークの接続の設定を使用可能にしているか、Microsoft Windows を使用して、ネットワークの接続試験を行ったり、自動的にネットワークの接続の設定を見つけることができます。

<テークオーバー後>

ノード 2 に制御が移り開始状態に変わります。



The screenshot shows the Enterprise Server Administration interface. At the top, it says 'Enterprise Server Administration' and 'バージョン 1.04.00'. Below that, the server name 'host2 (10.0.0.2)' is circled in red. The status is 'MDS00001 OK'. There is a '画面更新' button and an '自動更新間隔 (秒)' input field set to '10'. Below this is a table titled '1 - 1 of 1 Servers'.

	タイプ	名前	現ステータス	通信プロセス	ライセンス
	MFES	ESSRV01	開始	1 top:10.0.0.2:34424* (host2 +)	* / 10
			詳細	2 リスナー	詳細
			停止...		

The '開始' (Start) status in the table is circled in red. There is also a '追加' (Add) button at the bottom left of the table area.

<結果>

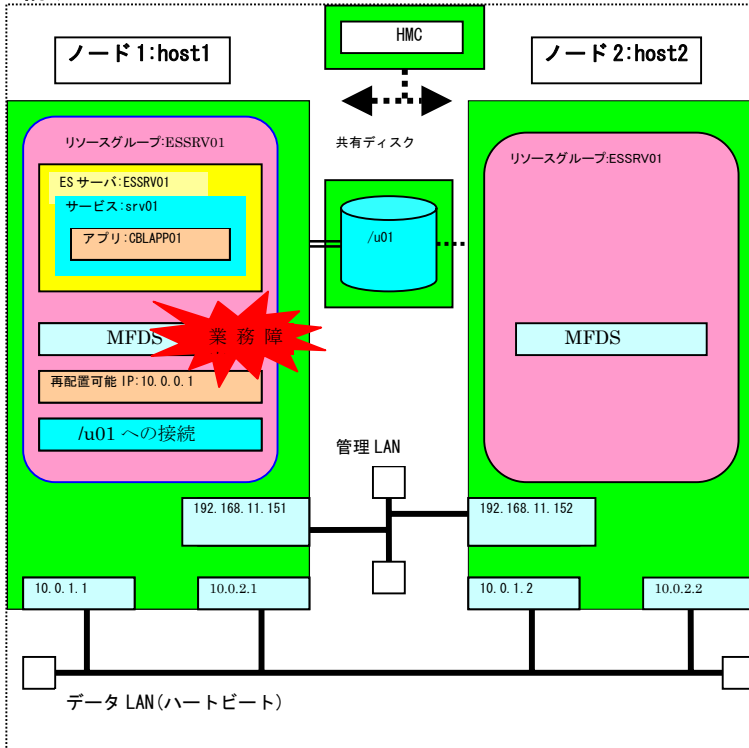
HACMP が LAN アダプタの障害を検知しノード 1→2 に制御が移ることが確認できました。

パターン④ 業務障害検出による切り替え

<目的>

Enterprise Server のディレクトリサーバ(mfds)を kill コマンドで終了させモニター用シェル(mfmonitor.ksh)でサーバダウンを検出し HACMP によるテークオーバー処理のトリガーを発生させます。

<構成>




<検証>

MFDS の kill 終了を HACMP のもとでシェルスクリプト(monitor.ksh)に検知させ、ゼロ以外のリターンコードを返すことによりテークオーバーをトリガーします。




<中断>

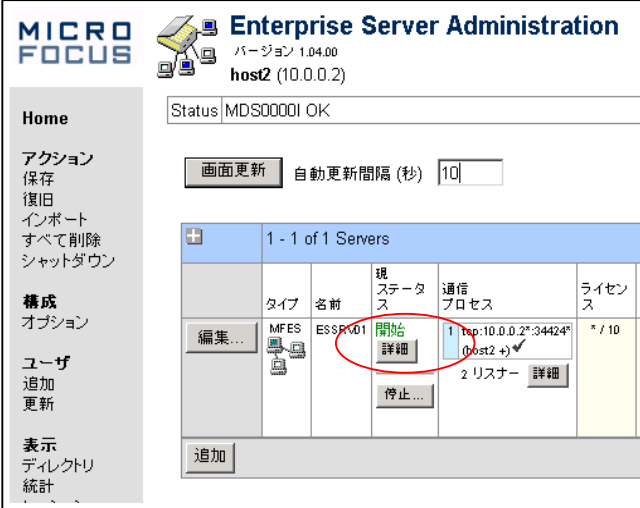
 ページを表示できません

検索中のページは現在、利用できません。Web サイトに技術的な問題が発生しているか、ブラウザの設定を調整する必要があります。

次のことを試してください:

-  [更新] ボタンをクリックするか、後でやり直してください。
- アドレス バーにページ アドレスを入力した場合は、ページ アドレスを正しく入力したかどうかを確認してください。
- 接続の設定を確認するには、[ツール] メニューの [インターネット オプション] をクリックします。[接続] タブで [ダイヤルアップの設定] グループの [設定] ボタン、または [LAN の設定] グループの [LAN の設定] ボタンをクリックしてください。設定情報は、LAN (ローカル エリアネットワーク) の管理者か、ISP (インターネット サービス プロバイダ) が提供する情報と一致する必要があります。
- ネットワーク管理者がネットワークの接続の設定を使用可能にしているか、Microsoft Windows を使用して、ネットワークの接続試験を行ったり、自動的にネットワークの接続の設定を見つけることができません。

<テークオーバー後>



The screenshot shows the Enterprise Server Administration interface. The main content area displays a table with the following data:

タイプ	名前	現ステータス	通信プロセス	ライセンス
MFES	ESSP001	開始	1 tcp:10.0.0.25:34424* (host2 +)	* / 10

The '開始' button in the '現ステータス' column is circled in red. Below the table, there are buttons for '編集...', '停止...', and '追加'.

<結果>

テークオーバーの結果 host1→2 への制御の移動が発生したことが Administration 画面で確認できました。

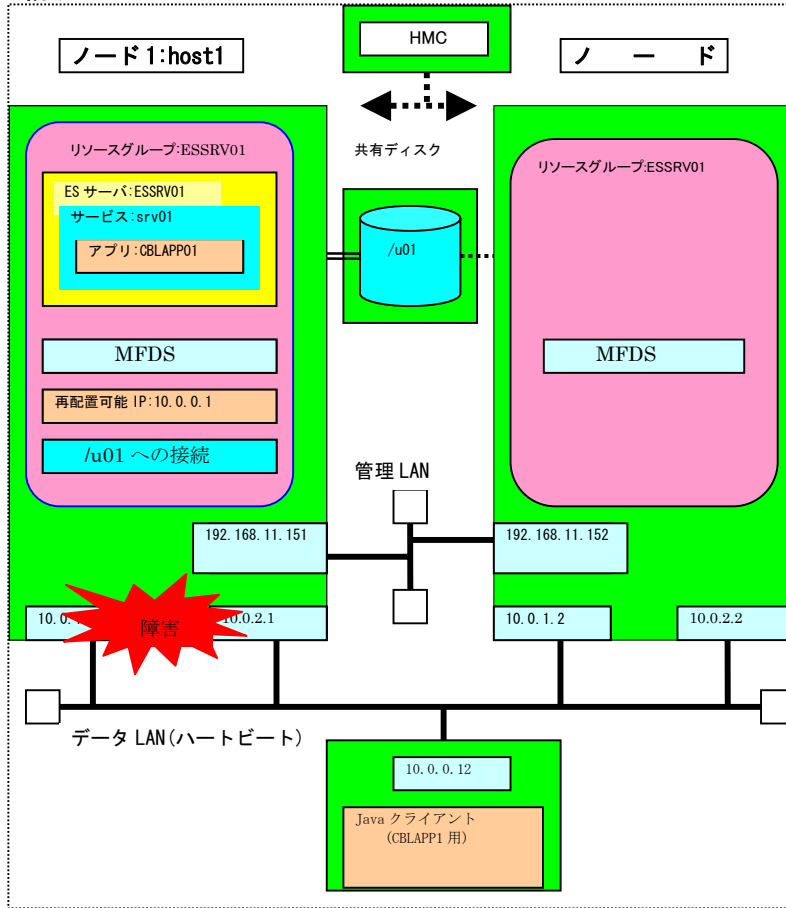
パターン⑤ アプリケーション実行中の自動切り替え

<目的>

java クライアントからノード 1 に接続した状態でテークオーバーを発生させ、テークオーバーによってクライアント・サーバセッションがノード 2 に引き継がれることを確認する。

二本の LAN ケーブルの抜き取りによるサーバステータスの異常をシェル(mfmonitor.ksh)で検出させ HACMP による自動テークオーバーの方法をとります。テークオーバー前の画面の状態はパターン①に同じです。さらに java クライアントが受け取ったメッセージから接続先のホスト名がノード2のホスト名を示し、かつ順序番号が中断をへてノード 2 で続行されたことを確認します。

<構成>



<検証>

ノード 1 は開始状態です



ノード 2 は待機状態です。



<クライアントプログラム実行結果>

```

06/08/03 17:08:50.03 CBLAPP01 host1 330
06/08/03 17:08:51.20 CBLAPP01 host1 331
06/08/03 17:08:52.59 CBLAPP01 host1 332
06/08/03 17:08:53.67 CBLAPP01 host1 333
06/08/03 17:08:54.88 CBLAPP01 host1 334
06/08/03 17:08:56.11 CBLAPP01 host1 335
06/08/03 17:08:57.27 CBLAPP01 host1 336
06/08/03 17:08:58.35 CBLAPP01 host1 337
06/08/03 17:08:59.55 CBLAPP01 host1 338
    
```

host1 でサービス実行

サービス中断し
1メッセージ書き込み
エラーでロス

```

javax.resource.spi.EISSystemException: IOException Connection reset by
peer: socket write error occurred executing srv01.CBLAPP01
DEBUG: can retry? true
    
```

```

06/08/03 17:10:36.09 CBLAPP01 host2 340
06/08/03 17:10:37.23 CBLAPP01 host2 341
06/08/03 17:10:38.50 CBLAPP01 host2 342
06/08/03 17:10:40.57 CBLAPP01 host2 343
06/08/03 17:10:41.69 CBLAPP01 host2 344
06/08/03 17:10:43.70 CBLAPP01 host2 345
06/08/03 17:10:45.12 CBLAPP01 host2 346
06/08/03 17:10:46.64 CBLAPP01 host2 347
06/08/03 17:10:48.62 CBLAPP01 host2 348
    
```

新規メッセージ No で
接続再開

テークオーバーにより
接続先が host2 に変更

<結果>

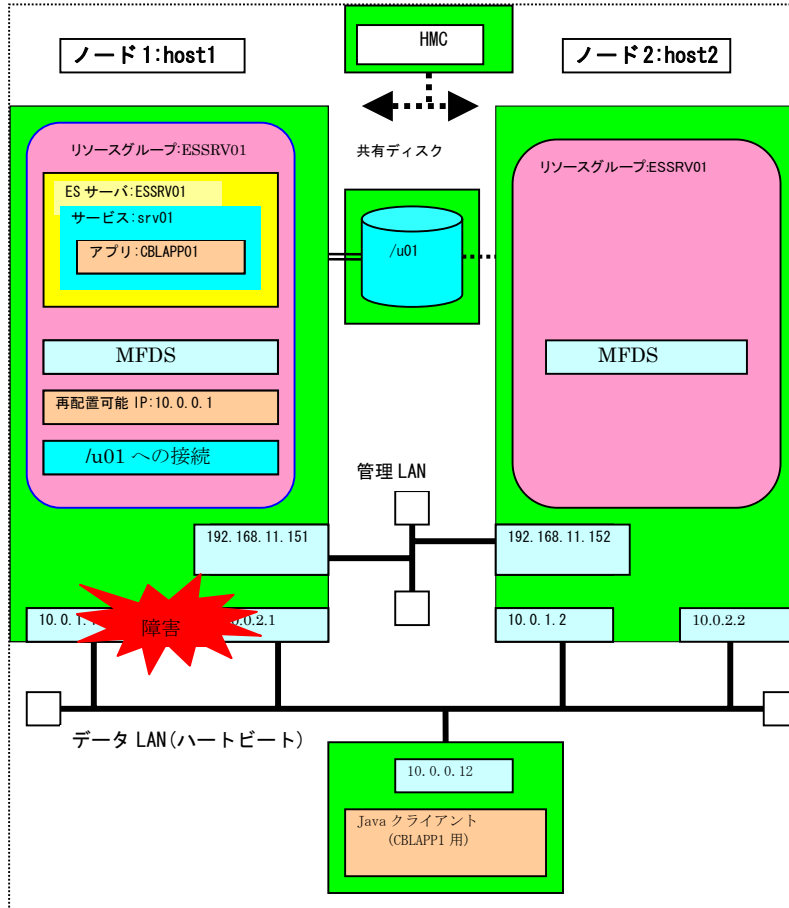
MFDS の Kill によりテークオーバーが発生しノード 1->2 に制御が移ります。接続は中断するが IP アドレスが host2(ノード 2)に引き継がれ処理が続行されることが確認できました。

パターン⑥ ES サーバ起動で待機

<目的>

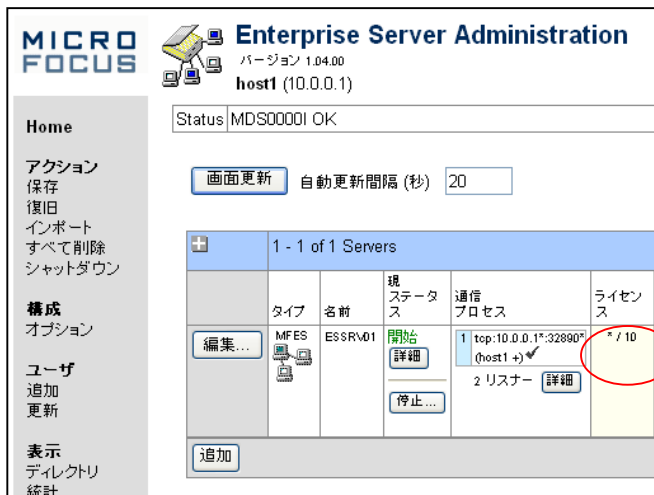
ES サーバを事前起動させての切り替えテストケースを実施・確認します。

<構成>



<検証>

ノード 1 で Administration 画面を手動開始しておきます。



ところがノード 2 で ESSRV01 を立ち上げるとリスナーがエラーで立ち上がらない状態になりました。



理由: Enterprise Server のリスナーは与えられた IP アドレスが事前に有効になっている必要があるがこの IP アドレスは再配置可能 IP アドレスのためテークオーバー前の時点ではまだ HACMP から渡されていません。このため”開始”状態になりません。

<結果>

Enterprise Server の制限事項により待機側の ES サーバを正常に立ち上げることができません。このため本パターンは前提条件を満たせないため動作検証できないことが判明しました。

パターン⑦ サーバごとに別の業務を提供しながら待機

<目的>

これはノード 1、ノード 2 に 2 つのサーバ(ESSRV01/ESSRV02)をそれぞれ定義しノード 1 は ESSRV01 を起動させノード 2 は ESSRV02 を起動させそれぞれ反対側のサーバでスタンバイさせておき、クライアントからの同時接続をさせてテークオーバーを確認するケースです。

クライアント機から 2 つのノードに同時アクセスをしながら検証します。

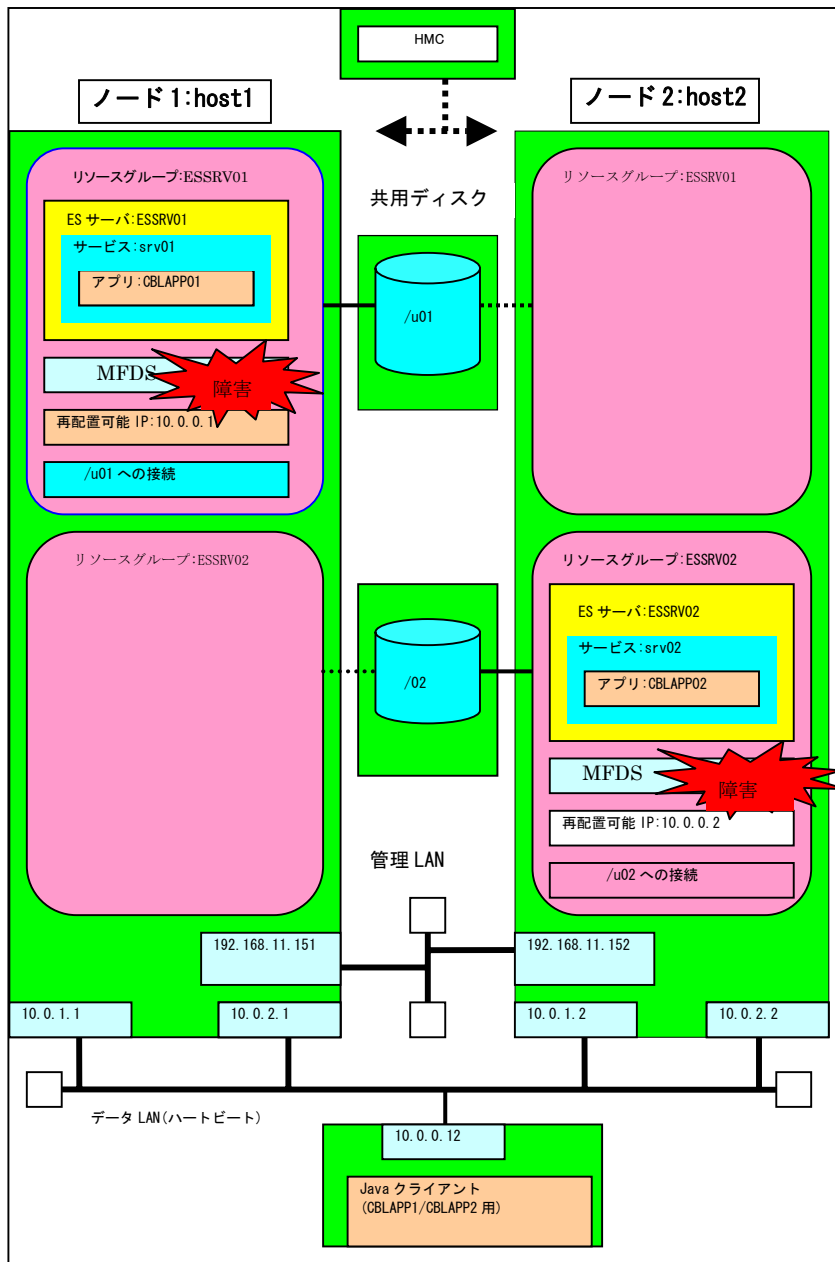
- ・ノード 1->2 へのテークオーバーはノード 1 側の MFDS を kill 終了させて実施。
- ・ノード 2->1 へのテークオーバーはノード 2 側の MFDS を kill 終了させて実施。

ノード 1 の MFDS kill 終了によりノード 1->2 に制御が移動します。IP アドレスが host2 に引き継がれ処理が継続することを確認します。

ノード 2 の MFDS の kill 終了においても同じ逆の動作ができることを確認します。

ノード 2 の ESSRV02 のサービス登録は、前出の環境設定の 3. COBOL コンポーネントの準備、4. Enterprise Server の設定の手順を参照して srv02.car,cblapp02.gnt,APPFILE02.idx,APPFILE02 を配置します。

<構成>



<検証>

それぞれのノードであらかじめ MFDS を立ち上げてサーバを起動しておきます。

Enterprise Server Administration
バージョン 1.04.00
host1 (10.0.0.1)

Status MDS00001 OK

自動更新間隔 (秒) 20

タイプ	名前	現ステータス	通信プロセス	ライセンス
MFES	ESSRV01	開始	1 top:10.0.0.1*:33246* (host1 +) 2 リスナー	* / 10
MFES	ESSRV02	停止	1 top:10.0.0.1*:33200* (host1 +) 2 リスナー	* / 10

Enterprise Server Administration
バージョン 1.04.00
host2 (10.0.0.2)

Status MDS00001 OK

自動更新間隔 (秒) 20

タイプ	名前	現ステータス	通信プロセス	ライセンス
MFES	ESSRV01	停止	1 top:10.0.0.2*:32904* (host2 +) 2 リスナー	* / 10
MFES	ESSRV02	開始	1 top:10.0.0.2*:32837* (host2 +) 2 リスナー	* / 10

クライアントマシンで各サービスに対する Java クライアントプログラムによるサービス呼び出しを繰り返し実行させておきます。

次に MFDS のプロセスを kill します。直後にノード 1 の管理画面の表示が停止します。すると HACMP にセットしたモニター用シェルが障害を検知してテークオーバーが発生します。

この結果ノード 1->2 に制御が移ります。

ページを表示できません

検索中のページは現在、利用できません。Web サイトに技術的な問題が発生しているか、ブラウザの設定を調整する必要があります。

次のことをお試しください:

- 「更新」ボタンをクリックするか、後でやり直してください。
- アドレスバーにページアドレスを入力した場合は、ページアドレスを正しく入力したかどうかを確認してください。
- 接続の設定を確認するときは、「ツール」メニューの「インターネット オプション」をクリックします。「接続」タブで「ダイヤルアップの設定」グループの「設定」ボタン、または「LAN の設定」グループの「LAN の設定」ボタンをクリックしてください。設定情報は、LAN (ローカルエリアネットワーク) の管理者か、ISP (インターネット サービス プロバイダ) が提供する情報と一致する必要があります。
- ネットワーク管理者がネットワークの接続の設定を使用可能にしていれば、Microsoft Windows を使用して、ネットワークの接続試験を行ったり、自動的にネットワークの接続の設定を見つけることができます。

Enterprise Server Administration
バージョン 1.04.00
host2 (10.0.0.2)

Status MDS00001 OK

自動更新間隔 (秒) 20

タイプ	名前	現ステータス	通信プロセス	ライセンス
MFES	ESSRV01	開始	1 top:10.0.0.2*:32989* (host2 +) 2 リスナー	* / 10
MFES	ESSRV02	開始	1 top:10.0.0.2*:32837* (host2 +) 2 リスナー	* / 10

<クライアントプログラム実行結果(CBLAPP01)>

```
06/08/03 13:08:01.19 CBLAPP01 host1 26
06/08/03 13:08:03.48 CBLAPP01 host1 27
06/08/03 13:08:06.21 CBLAPP01 host1 28
06/08/03 13:08:08.98 CBLAPP01 host1 29
06/08/03 13:08:11.78 CBLAPP01 host1 30
06/08/03 13:08:15.60 CBLAPP01 host1 31
06/08/03 13:08:17.90 CBLAPP01 host1 32
06/08/03 13:08:19.87 CBLAPP01 host1 33
java.net.ConnectException: Connection refused: connect
  at java.net.PlainSocketImpl.socketConnect(Native Method)
  at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:305)
  at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:171)
  at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:158)
  at java.net.Socket.connect(Socket.java:452)
  at java.net.Socket.connect(Socket.java:402)
  at java.net.Socket.<init>(Socket.java:309)
  at java.net.Socket.<init>(Socket.java:153)
  at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:894)
  at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:825)
  at com.microfocus.cobol.connector.transport.BINPStream.<init>(BINPStream.java:97)
  at com.microfocus.cobol.connector.transport.BINPRemoteCall.open(BINPRemoteCall.java:176)
  at com.microfocus.cobol.connector.transport.BINPRemoteCall.connect(BINPRemoteCall.java:73)
  at com.microfocus.cobol.connector.transport.BINPRemoteCall.reconnect(BINPRemoteCall.java:145)
  at com.microfocus.cobol.connector.PureCobolBean.cobcall(PureCobolBean.java:292)
  at com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:247)
  at com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:174)
  at clptest.main(clptest.java:48)
javax.resource.spi.EISSystemException: CobolException Exception throw during open (see getCause() for more
information) (elapsed time=1022.0)for 10.0.0.1:9003 (cause class:java.net.ConnectException,
cause:Connection refused: connect) executing srv01.CBLAPP01
06/08/03 13:09:09.25 CBLAPP01 host2 34
06/08/03 13:09:12.13 CBLAPP01 host2 35
06/08/03 13:09:14.59 CBLAPP01 host2 36
06/08/03 13:09:17.24 CBLAPP01 host2 37
06/08/03 13:09:19.99 CBLAPP01 host2 38
06/08/03 13:09:22.73 CBLAPP01 host2 39
06/08/03 13:09:25.66 CBLAPP01 host2 40
06/08/03 13:09:28.44 CBLAPP01 host2 41
06/08/03 13:09:31.49 CBLAPP01 host2 42
```

host1 でサービスを実行中

テークオーバー時のサービス中断

host2 でサービスを引き継ぐ。データファイルの連番も継続される

<クライアントプログラム実行結果(CBLAPP02)>
ノード 2->1 への逆のテークオーバー

```
06/08/03 16:34:31.98 CBLAPP02 host2 308
06/08/03 16:34:34.61 CBLAPP02 host2 309
06/08/03 16:34:36.98 CBLAPP02 host2 310
06/08/03 16:34:39.13 CBLAPP02 host2 311
06/08/03 16:34:41.18 CBLAPP02 host2 312
06/08/03 16:34:42.76 CBLAPP02 host2 313
06/08/03 16:34:44.66 CBLAPP02 host2 314
06/08/03 16:34:45.89 CBLAPP02 host2 315
06/08/03 16:34:48.14 CBLAPP02 host2 316
06/08/03 16:34:50.77 CBLAPP02 host2 317
06/08/03 16:34:53.70 CBLAPP02 host2 318
06/08/03 16:34:56.38 CBLAPP02 host2 319
06/08/03 16:34:59.10 CBLAPP02 host2 320
06/08/03 16:35:01.17 CBLAPP02 host2 321
06/08/03 16:35:04.07 CBLAPP02 host2 322
06/08/03 16:35:51.44 CBLAPP02 host1 323 ← 同時にテークオーバーにより
06/08/03 16:35:53.58 CBLAPP02 host1 324 ← ノード 2 から 1 に接続先が変わった
06/08/03 16:35:56.10 CBLAPP02 host1 325
06/08/03 16:35:57.81 CBLAPP02 host1 326
```

<結果>

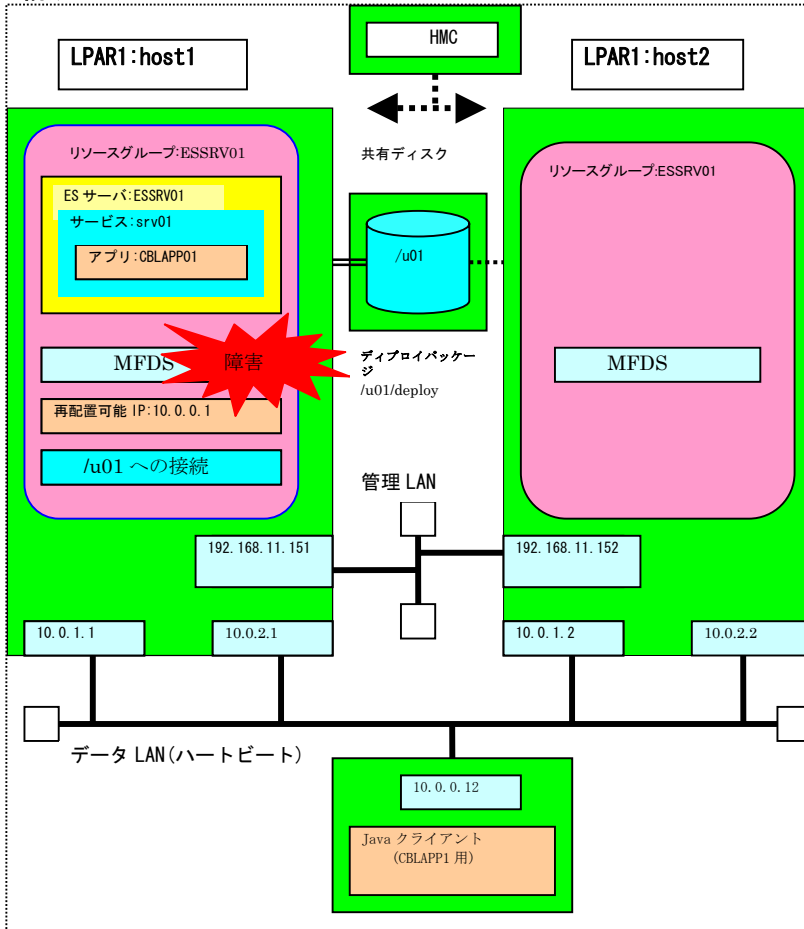
上記のログおよび Administration 画面を確認することで双方向へのテークオーバーを確認できました。

パターン⑧ ディプロイパッケージの共有

<目的>

ディプロイパッケージを共有ディスク上に配置しテークオーバーによりノード 2 側からのディプロイパッケージのアクセスができることを確認します。

<構成>



<準備>

準備作業としてそれぞれのノードの ES サーバから一度ディプロイ済みのサービスをパッケージとともに削除します。ノード 1 の \$COBDIR/ deploy/ srv01 に作成した srv01.car, cblapp01.gnt, APPFILE01、APPFILE01.idx をノード1の共有ディプロイパッケージ域(/u01/ deploy/ srv01)にコピー。また \$COBDIR/ deploy/ .mfdeploy も /u01/ deploy へコピーします(または手修正します)。

```
cd /u01/ deploy/ srv01
mfdepinst srv01.car でサービスディプロイを実施します。
```

同じコピー操作及びディプロイ操作をノード 2 でも実行しノード 2 をスタンバイ待機させます。

本検証では /u01/ deploy にディプロイパッケージを配置しました。



この作業を両ノード共に実施します。

Enterprise Server Administration > ESSRV01 > パッケージ
バージョン 1.04.00
host2 (10.0.0.2)

Status: MDS0000I OK

Server ESSRV01 [停止]

サーバー... リスナー (2) サービス (3) ハンドラ (2) **パッケージ (1)**

1 - 1 of 1 packages Show 10 packages

	名前	現ステータス	ステータスログ	パッケージモジュール	IDT	パッケージパス	カスタム構成
編集...	srv01.CBLAPP01	Available	OK		A01/Deploy/srv01/srv01.id	A01/Deploy/srv01	

追加

ノード1でリソースグループ ESSRV01を開始しておきます。

Enterprise Server Administration
バージョン 1.04.00
host1 (10.0.0.1)

Status: MDS0000I OK

画面更新 自動更新間隔 (秒) 20

1 - 1 of 1 Servers

	タイプ	名前	現ステータス	通信プロセス	ライセンス
編集...	MFES	ESSRV01	開始	1 top:10.0.0.1*:33175* (host1 *)	* / 10

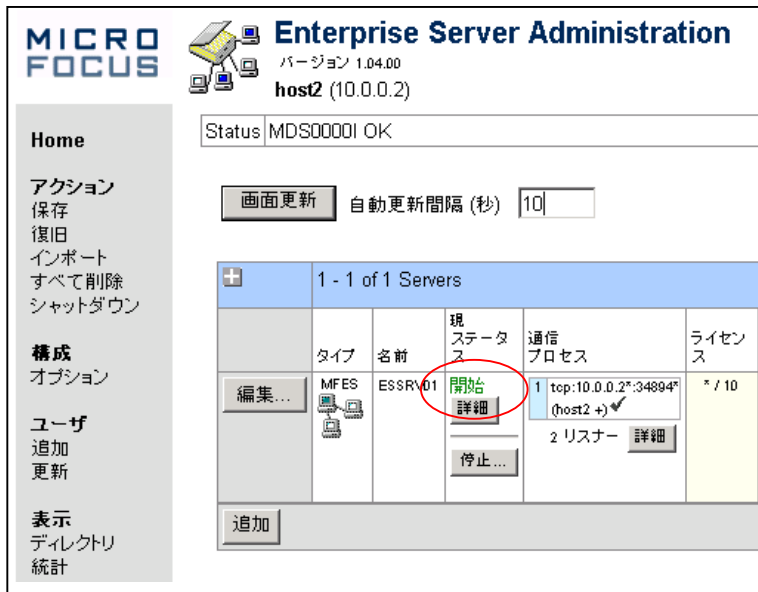
2 リスナー 詳細

追加

<検証>

ノード1にjavaクライアント接続させておき、ノード1のMFDSをkill終了させテークオーバーを起こします。テークオーバーによりノード1→2へ制御が移ります。

テークオーバーによりノード 2 でリソースグループ ESSRV01 が開始されます。



アプリケーションは接続・実行し途中中断をはさみノード 2 にテークオーバーされます。

```

06/08/03 20:17:05.90 CBLAPP01 host1 425
06/08/03 20:17:07.56 CBLAPP01 host1 426
06/08/03 20:17:08.86 CBLAPP01 host1 427
06/08/03 20:17:10.51 CBLAPP01 host1 428
06/08/03 20:17:12.05 CBLAPP01 host1 429
06/08/03 20:17:13.78 CBLAPP01 host1 430
06/08/03 20:17:15.23 CBLAPP01 host1 431
C:\¥HACMPVerify>javacplptest.bat
C:\¥HACMPVerify>java clptest | tee -a Case1Log
java.net.ConnectException: Connection refused: connect
    at java.net.PlainSocketImpl.socketConnect(Native Method)
    at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:305)
    at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:171)
.....
(途中省略)
.....
javax.resource.spi.EISSystemException: CobolException Exception throw during open (see
getCause() for more information) (elapsed time=1021.0)for 10.0.0.1:9003 (cause
class:java.net.ConnectException, cause:Connection refused: connect) executing
srv01.CBLAPP01
06/08/03 20:18:04.07 CBLAPP01 host2 432
06/08/03 20:18:05.08 CBLAPP01 host2 433
06/08/03 20:18:06.08 CBLAPP01 host2 434
06/08/03 20:18:07.07 CBLAPP01 host2 435
06/08/03 20:18:08.19 CBLAPP01 host2 436

```

<結果>

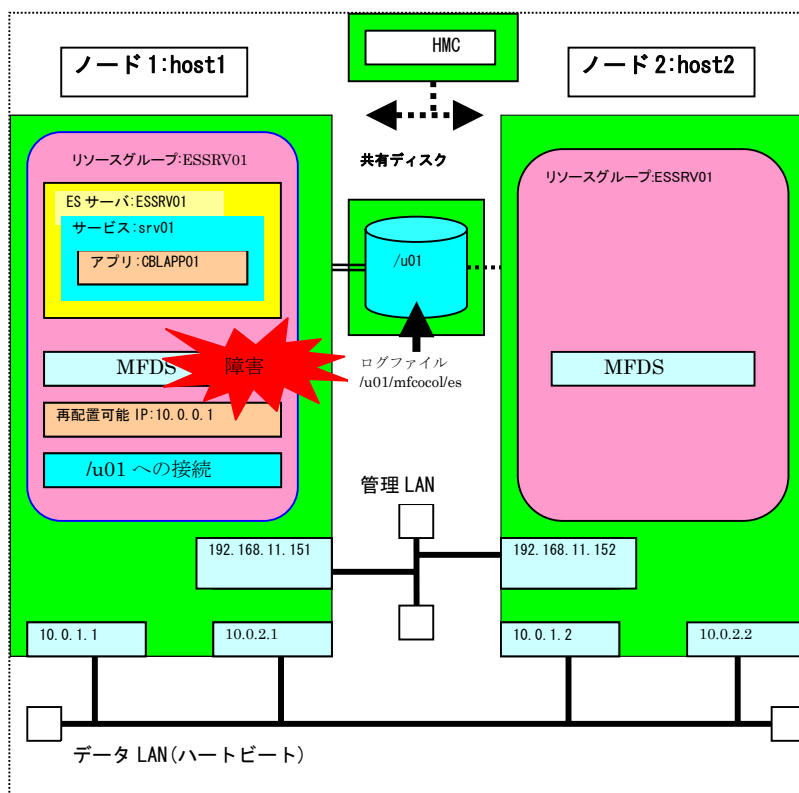
クライアントのログの host 名と連番の確認によって共有ディスクのディプロイパッケージがノード間で正しく引き継がれたことが確認できました。

パターン⑨ ログファイルの共有

<目的>

ES サーバが出力するログファイルの出力先を変更して共有化する。テークオーバー後にノード2側から、ノード1側で作成したログを参照できる事を確認します。

<構成>



<検証>

ログファイルを共有するにはシェル casperm を使用して下記のように行います。

ノード1側で

```
cd $COBDIR
```

```
sh bin/casperm
```

を実行して次ページのように共有ディレクトリのパスを指定します。

<casperm>

```
# cd $COBDIR
# sh bin/casperm
```

続行する前に、Enterprise Server のシステム管理者用の英数字のユーザー ID を手元に準備しているか確認してください。

詳細は、マニュアル「構成と管理」に記述されています。

終了するには、'q' を、続行するには、改行を押してください。:

次に続く質問に答えながら、グローバルトランザクションシステム構成ファイル(cas.cfg)を構築します。- 詳細はマニュアル「構成と管理」を参照してください。

エラーや警告メッセージを syslog デーモンから表示しますか (y/n)?

y

Enterprise Server のシステム管理者用の英数字のユーザー ID を入力してください。:root

Enterprise Server 実行時システムファイルをコピーするディレクトリを絶対パス名で入力してください。環境変数は使用できません。

/var/mfcobol/es にコピーする場合は、RETURN キーを押してください。

/u01/mfcobol/es

ファイルパーミッションを設定中です。お待ちください...

<\$COBDIR/etc/cas/cas.cfg の確認>

casperm コマンドにより CASROOT パラメータにログディレクトリパスが下記の様に設定されます。

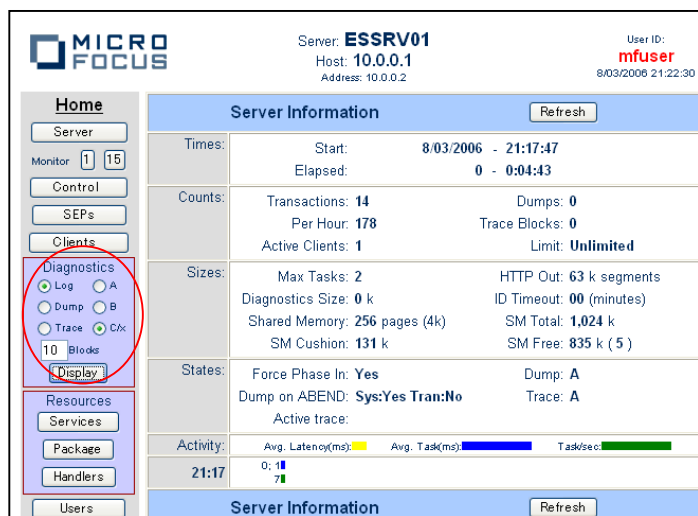
```
[root@host1:/opt/mf/ES40SP2]# cd $COBDIR/etc/cas
[root@host1:/opt/mf/ES40SP2/etc/cas]# cat cas.cfg
#####
### Transaction System Global/Region Configuration File ###
#####
###          global only directives          ###
USESYSCON=Y
AUTHPGMS=
CASROOT=/u01/mfcobol/es   <←----ログディレクトリパス
###          other directives          ###
#####
[root@host1:/opt/mf/ES40SP2/etc/cas]#
```

正しく処理されたことを\$COBDIR/etc/cas/cas.cfg の CASROOT=/u01/mfcobol/es になっていることで確認します。ノード2側でも同じ操作を実施しスタンバイさせます。

<テスト方法>

ノード1のES ESSRV01を立ち上げサービス要求を数回実施。

次に ES モニター & コントロール画面に入ります。



Server Information 画面から Log と C/x のラジオボタンを押し現行ログの内容の日付、時間及び内容を確認します。

```

ESSRV01
System Log: $TXRFDIR/console.log
060803 21114183          CASCD0100I ES Daemon Initialized (Ver CAS 3.0.14) 21:11:41
060803 21114286          CASCD0120I Server manager created for ES ESSRV01, process-id = 679950 21:11:42
060803 21114290          679950 ESSRV01  CASSI0000I Server manager initialization started 21:11:42
060803 21114490          679950 ESSRV01  CASSI4005I Retrieving ES configuration from MFDS (127.0.0.1:86) 21:11:44
060803 21114537          CASCD1020I JCP service process created for region ESSRV01, process-id = 712816
060803 21114538          CASCD0127I SEP 0001 created for ES ESSRV01, process-id = 737480 21:11:45
060803 21114600          712816 ESSRV01  CASJC0001I Journal control initialized 21:11:46
060803 21114601          737480 ESSRV01  CASSI1500I SEP initialization started 21:11:46
060803 21114603          737480 ESSRV01  CASSI1600I SEP initialization completed successfully 21:11:46
060803 21114605          CASCD0127I SEP 0002 created for ES ESSRV01, process-id = 229400 21:11:46
060803 21114607          229400 ESSRV01  CASSI1500I SEP initialization started 21:11:46
060803 21114608          229400 ESSRV01  CASSI1600I SEP initialization completed successfully 21:11:46
060803 21114611          CASCD1038I ES Communications Server created, ES ESSRV01, process-id = 524532 2
060803 21114611          679950 ESSRV01  CASKC1000I ES concurrent request limit: (unlimited) 21:11:46
060803 21114611          679950 ESSRV01  CASSI1000I Server Manager initialization completed successfully 21:11:46
060803 21114815          524532 ESSRV01  CASCS5001I Communications interface 01 initialization started 21:11:48
060803 21114815          524532 ESSRV01  CASCS5003I Communications interface 01 initialization complete 21:11:48
    
```

次にノード 1 の MFDS を kill 終了させノード 2 にテークオーバーさせます。
 ノード 2 の ES モニター & コントロール画面に入り Log と B のラジオボタンを押して共有ディスクの前のログ console.bak を確認し、その日付時間がノード1で見たものと同じ内容のままであることをチェックし、ログディレクトリの共有が正しく行われたことを確認します。

```

ESSRV01
System Log: $TXRFDIR/console.bak
060803 21114189          CASCD01001 ES Daemon Initialized (Ver CAS 3.0.14) 21:11:41
060803 21114288          CASCD01201 Server manager created for ES ESSRV01, process-id = 679950 21:11:42
060803 21114290      679950 ESSRV01  CASSI00001 Server manager initialization started 21:11:42
060803 21114490      679950 ESSRV01  CASSI40051 Retrieving ES configuration from MFDS (127.0.0.1:86) 21:11:44
060803 21114597          CASCD10201 JCP service process created for region ESSRV01, process-id = 712816
060803 21114598          CASCD01271 SEP 0001 created for ES ESSRV01, process-id = 737480 21:11:45
060803 21114600      712816 ESSRV01  CASJC00011 Journal control initialized 21:11:46
060803 21114601      737480 ESSRV01  CASSI15001 SEP initialization started 21:11:46
060803 21114603      737480 ESSRV01  CASSI16001 SEP initialization completed successfully 21:11:46
060803 21114605          CASCD01271 SEP 0002 created for ES ESSRV01, process-id = 229400 21:11:46
060803 21114607      229400 ESSRV01  CASSI15001 SEP initialization started 21:11:46
060803 21114608      229400 ESSRV01  CASSI16001 SEP initialization completed successfully 21:11:46
060803 21114611          CASCD10381 ES Communications Server created, ES ESSRV01, process-id = 524532 21:11:46
060803 21114611      679950 ESSRV01  CASKC10001 ES concurrent request limit: (unlimited) 21:11:46
060803 21114611      679950 ESSRV01  CASSI10001 Server Manager initialization completed successfully 21:11:46
060803 21114815      524532 ESSRV01  CASC50001 Communications interface 01 initialization started 21:11:48
060803 21114815      524532 ESSRV01  CASC50031 Communications interface 01 initialization complete 21:11:48
060803 21170422      684184 ESSRV01  CASST00051 Shutdown of ES ESSRV01 starting 21:17:04
060803 21170424      679950 ESSRV01  CASKC00031 SEP 0000229400 shutdown complete. 21:17:04
060803 21170424      679950 ESSRV01  CASKC00031 SEP 0000737480 shutdown complete. 21:17:04
060803 21170424      679950 ESSRV01  CASKC10021 ES allocated license limit: (unlimited) units, peak usage: 000000000
060803 21170425          CASCD01311 SEP 0002 for server ESSRV01 has terminated normally 21:17:04
060803 21170426          CASCD01311 SEP 0001 for server ESSRV01 has terminated normally 21:17:04
060803 21170426      679950 ESSRV01  CASST00001 Server manager termination started 21:17:04
060803 21170426      524532 ESSRV01  CASC50071 Communications interface 01 termination started 21:17:04

```

<結果>

console.log の内容がテークオーバー後 console.bak と名前を変更されて残っていてその内容が保存されていることを確認できました。

テスト結果のまとめ

パターン	タイトル	テスト結果	確認事項
①	手動切り替え	○	HACMP コマンド機能で切り替え
②	電源障害による切り替え	○	LPAR の電源オフ機能による切り替え
③	データ LAN アダプタ(二重)障害	○	2 本のデータ LAN を抜くことによる LAN アダプタ障害切り替え
④	業務障害検出による切り替え	○	MDUMP ユーティリティ情報を使用した業務系プロセス障害検出による切り替え
⑤	接続業務の自動切り替え	○	テークオーバーによる切り替えに伴う業務の自動継続の確認
⑥	MFDS サーバ起動で待機	×	Enterprise Server のリスナーは定義する IP が立ち上げ時に実在しなければ立ち上がらないため、このパターンは実装不可
⑦	サーバ毎に別の業務を提供しながら待機(相互スタンバイ)	○	ノード毎に異なるサーバを立ち上げてクライアント接続を並行実施し相互にテークオーバー可能なことを確認
⑧	ディプロイパッケージの共有	○	共有ディスクにディプロイパッケージを配置して切り替え
⑨	ログ出力先の共有	○	共有ディスクにログファイルの出力先を指定して切り替え

- ・ 切り替え要因としてパターン①～④までのものを用意したが HACMP の切り替えが作動して設定したシェルが起動され待機ノードへの切り替えが行われたことを確認した。
- ・ パターン③で業務系プロセス異常を検知するためマイクロフォーカス提供の mdump ユーティリティで情報収集し異常を検知するシェルをモニターシェルとして HACMP に組み込み、想定した Enterprise Server ステータス異常をとらえて切り替えることができた。
- ・ お客様の本運用においては、ユーザープログラム又は関連ソフトウェアの監視なども含め、監視方法や監視対象を検討しシェルを作りこむ必要があります。
- ・ サーバを事前に立ち上げておくパターン⑥のケースでは、待機系のノードの Enterprise Server を立ち上げたとき再配置可能 IP アドレスが割り当てられていない状態のためリスナーが”停止”状態になる。このためテークオーバーができないことが判明。待機側は MFDS の立ち上げで待機させる必要がある。
- ・ Enterprise Server システムの情報の共有化のテストではディプロイパッケージ、ログファイルを共有してテークオーバーすることを確認。
- ・ Enterprise Server 構成リポジトリは MFDS によって一元的に管理される構成データベース。このリポジトリは Enterprise Server 運用時に随時そのステータスを保存するため常に更新モードでオープンされるためホットスタンバイでは共有ディレクトリに配置することはできません。シンプルな運用を考慮してそれぞれローカルに配置し管理することをお勧めします。

付録

1. サーバアプリケーションサンプルプログラム (cbl app01.cbl)

```
FILE-CONTROL.
  SELECT APPFILE ASSIGN TO EXTERNAL APPFILE01
    ORGANIZATION INDEXED RECORD KEY RKEY
    ACCESS MODE RANDOM
    FILE STATUS IS AP-STATUS.

DATA          DIVISION.
FILE          SECTION.
FD APPFILE.
01 AF-REC.
  05 RKEY     PIC X(6).
  05 AF-COUNTER PIC 9(10).
WORKING-STORAGE SECTION.
01 WK-DATE   PIC X(6).
01 WK-TIME   PIC X(8).
01 AP-STATUS PIC X(2).
LINKAGE      SECTION.
01 LK-DATETIME PIC X(20).
01 LK-PGMNAME  PIC X(10).
01 LK-HOSTNAME PIC X(20).
01 LK-COUNTER  PIC 9(10).
PROCEDURE    DIVISION
  USING LK-DATETIME LK-PGMNAME LK-HOSTNAME LK-COUNTER.

1.
  PERFORM GET-DATETIME.
  MOVE 'CBLAPP01 ' TO LK-PGMNAME.
  CALL 'gethostname' USING LK-HOSTNAME BY VALUE 20.
  OPEN I-O APPFILE.
  MOVE SPACE TO RKEY.
  READ APPFILE INVALID KEY CONTINUE.
  MOVE AF-COUNTER TO LK-COUNTER.
  ADD 1 TO AF-COUNTER.
  REWRITE AF-REC INVALID KEY CONTINUE.
  CLOSE APPFILE.
  EXIT PROGRAM.

GET-DATETIME.
  ACCEPT WK-DATE FROM DATE.
  ACCEPT WK-TIME FROM TIME.
  STRING WK-DATE(1:2) '/' WK-DATE(3:2) '/' WK-DATE(5:2) ' '
    WK-TIME(1:2) ':' WK-TIME(3:2) ':' WK-TIME(5:2) '.'
    WK-TIME(7:2) DELIMITED BY SIZE INTO LK-DATETIME.
```


2. クライアントアプリケーションサンプルプログラム (clptest.java)

```
import com.microfocus.cobol.connector.spi.*;
import com.microfocus.cobol.connector.cci.*;
import com.microfocus.cobol.lang.*;

public class clptest
{
    public static void main(String[] args)
    {
        CobolNoTxManagedConnectionFactory mcf;
        javax.resource.cci.ConnectionFactory cxf;
        javax.resource.cci.Connection connection=null;
        javax.resource.cci.Interaction interaction=null;

        try{
            mcf = new CobolNoTxManagedConnectionFactory();
            //mcf.setServerHost("172.16.1.183");
            mcf.setServerHost("10.0.0.1");
            mcf.setServerPort("9003");
            mcf.setTrace(new Boolean(false));
            cxf = (javax.resource.cci.ConnectionFactory) mcf.createConnectionFactory();
            connection = cxf.getConnection();

            {
                javax.resource.cci.Interaction ix =
                    connection.createInteraction();
                com.microfocus.cobol.connector.cci.CobolInteractionSpec
                    iSpec = new
                        com.microfocus.cobol.connector.cci.CobolInteractionSpec();
                iSpec.setFunctionName("initialize");
                javax.resource.cci.RecordFactory rf = cxf.getRecordFactory();
                javax.resource.cci.IndexedRecord irec =
                    rf.createIndexedRecord("beanArgs");
                irec.add(new Boolean(true));
                javax.resource.cci.Record orec = ix.execute(iSpec, irec);
                ix.close();
            }

            javax.resource.cci.Interaction ix = connection.createInteraction();
            CobolInteractionSpec iSpec = new CobolInteractionSpec();
            iSpec.setFunctionName("srv01.CBLAPP01");
            iSpec.setArgument(0, java.lang.String.class, com.microfocus.cobol.RuntimeProperties.OUTPUT_ONLY);
            iSpec.setArgument(1, java.lang.String.class, com.microfocus.cobol.RuntimeProperties.OUTPUT_ONLY);
            iSpec.setArgument(2, java.lang.String.class, com.microfocus.cobol.RuntimeProperties.OUTPUT_ONLY);
            iSpec.setArgument(3, Long.class, com.microfocus.cobol.RuntimeProperties.OUTPUT_ONLY);
            javax.resource.cci.RecordFactory rf = cxf.getRecordFactory();
            javax.resource.cci.IndexedRecord iRec = rf.createIndexedRecord("CBLAPP01In");
            javax.resource.cci.IndexedRecord oRec = rf.createIndexedRecord("CBLAPP01Out");
            ix.execute(iSpec, iRec, oRec);
            ix.close();
            System.out.println(oRec.get(0) + " " + oRec.get(1) + " " + oRec.get(2) + " " + oRec.get(3));
        }
        catch(Exception e) {
            System.err.println(e);
        }
        return;
    }
}
```

3. クライアントシェルスクリプト (XPbatRun.bat)

```
set
classpath=. ;C:\NetX4\Base\BIN\mfcobol.jar;C:\NetX4\Base\BIN\mfconnector.jar;C:\j2ee1.4.2\lib\j2ee.jar
for /L %%A IN (1,1,10000) do javac\ptest.bat
```

4. アプリケーションデータ作成サンプルプログラム (initfile1.cbl)

```
FILE-CONTROL.
  SELECT APPFILE ASSIGN TO EXTERNAL APPFILE01
  ORGANIZATION INDEXED RECORD KEY RKEY
  ACCESS MODE RANDOM.

DATA          DIVISION.
FILE          SECTION.

FD APPFILE.
01 AF-REC.
  05 RKEY     PIC X(6).
  05 AF-COUNTER PIC 9(10).

PROCEDURE     DIVISION.
1.
  OPEN        OUTPUT    APPFILE.
  MOVE        SPACE     TO RKEY.
  MOVE        ZERO      TO AF-COUNTER.
  WRITE       AF-REC.
  CLOSE       APPFILE.
  STOP        RUN.
```

5. killallcas.ksh シェルスクリプト

```
#!/bin/ksh
#set -x
#
#      ===== All MFES CAS processes kill Tool =====
#
# Requirements:
#      1. Have a correct $COBDIR environment
#      2. Have the same logon user who ran "mfds" command
#
#*****
#
# Script Start here .....
#
echoCmd="echo"
runUser=`whoami`
WaitTime1=1
WaitTime2=2

#####
# kill all cascd32 processes (Start)
###
$echoCmd "¥n***** Kill all cascd32 processes start *****"
cascdid=`ps -fu $runUser | fgrep "$COBDIR/bin/casc32 /r" | fgrep -v fgrep |
fgrep -v awk | awk '{print $2}'`
if test "x$cascdid" != "x"
then
    $echoCmd "casc32 process are(is) :¥n$cascdid¥n"
    for i in $cascdid
    do
        kill -9 $i
        $echoCmd "$i cascd32 process has been killed"
        sleep "$WaitTime1"
    done
fi
sleep "$WaitTime2"
cascdid=`ps -fu $runUser | fgrep "$COBDIR/bin/casc32 /r" | fgrep -v fgrep | fgrep -v awk | awk '{print
$2}'`
if test "x$cascdid" != "x"
then
    $echoCmd "Surviving cascd32 process are(is) :$cascdid¥n"
else
    $echoCmd "***** Kill all cascd32 processes complete *****"
fi
```

```

###
#####
# kill all casjcp32 processes (Start)
###
$echoCmd "¥n***** Kill all casjcp32 processes start *****"
$echoCmd "Please Wait...¥n"

casjcpid=`ps -fu $runUser | fgrep "casjcp32 /r" | fgrep -v fgrep | fgrep -v awk | awk ' {print $2}' `
if test "x$casjcpid" != "x"
then
    $echoCmd "All casjcp32 process are :¥n$casjcpid¥n"
    for i in $casjcpid
    do
        kill -9 $i
        $echoCmd "$i casjcp32 process has been killed"
        sleep "$WaitTime1"
    done
fi
sleep "$WaitTime2"
casjcpid=`ps -fu $runUser | fgrep "casjcp32 /r" | fgrep -v fgrep | fgrep -v awk | awk ' {print $2}' `
if test "x$casjcpid" != "x"
    $echoCmd "Surviving casjcp32 process are :$casjcpid¥n"
else
    $echoCmd "***** Kill all casjcp32 processes complete *****"
fi
###
# kill all casjcp32 processes (End)
#####

#####
# kill all casmgr32 processes (Start)
###
$echoCmd "¥n***** Kill all casmgr32 processes start *****"
$echoCmd "Please Wait...¥n"

casmgrid=`ps -fu $runUser | fgrep "casmgr32 /r" | fgrep -v fgrep | fgrep -v awk | awk ' {print $2}' `
if test "x$casmgrid" != "x"
then
    $echoCmd "All casmgr32 process are :¥n$casmgrid¥n"
    for i in $casmgrid
    do
        kill -9 $i

```

```

        sleep "$WaitTime1"
    done
fi
sleep "$WaitTime2"
casmgrid=`ps -fu $runUser | fgrep "casmgr32 /r" | fgrep -v fgrep | fgrep -v awk | awk '{print $2}'`
if test "x$casmgrid" != "x"
then
    $echoCmd "Surviving casmgr32 process are :$casmgrid¥n"
else
    $echoCmd "***** Kill all casmgr32 processes complete *****"
fi

###
# kill all casmgr32 processes (End)
#####

#####
# kill all cassi32 processes (Start)
###
$echoCmd "¥n***** Kill all cassi32 processes start *****"
$echoCmd "Please Wait...¥n"
if test "x$cassiid" != "x"
then
    $echoCmd "All cassi32 process are : ¥n$cassiid¥n"
    for i in $cassiid
    do
        kill -9 $i
        $echoCmd "$i cassi32 process has been killed"
        sleep "$WaitTime1"
    done
fi
sleep "$WaitTime2"
cassiid=`ps -fu $runUser | fgrep "cassi32 /r" | fgrep -v fgrep | fgrep -v awk | awk '{print $2}'`
if test "x$cassiid" != "x"
then
    $echoCmd "Surviving cassi32 process are...$cassiid¥n"
else
    $echoCmd "***** Kill all cassi32 processes complete *****"
fi

###
# kill all cassi32 processes (End)
#####

```

```

# kill all mfcs32 processes (Start)
###
$echocmd "¥n***** Kill all mfcs32 processes start *****"
$echocmd "Please Wait...¥n"

mfcsid=`ps -fu $runUser | fgrep "$COBDIR/bin/mfcs32 -n" | fgrep -v fgrep | fgrep -v awk | awk ' {print $2}'`
if test "x$mfcsid" != "x"
then
$echocmd "All mfcs32 process are :¥n$mfcsid¥n"
for i in $mfcsid
do
kill -9 $i
$echocmd "$i mfcs32 process has been killed"
sleep "$WaitTime1"
done
fi
sleep "$WaitTime2"
mfcsid=`ps -fu $runUser | fgrep "$COBDIR/bin/mfcs32 -n" | fgrep -v fgrep | fgrep -v awk | awk ' {print $2}'`
if test "x$mfcsid" != "x"
then
$echocmd "Surviving mfcs32 process are :$mfcsid¥n"
$echocmd "***** Kill all mfcs32 processes complete *****"
fi

###
# kill all mfcs32 processes (End)
#####

#####
# kill all casstop processes (Start)
###
$echocmd "¥n***** Kill all casstop processes start *****"
$echocmd "Please Wait...¥n"

casstopid=`ps -fu $runUser | fgrep casstop | fgrep -v fgrep | fgrep -v awk |
awk ' {print $2}'`
if test "x$casstopid" != "x"
then
$echocmd "All casstop process are :¥n$casstopid¥n"
for i in $casstopid
do
kill -9 $i
$echocmd "$i casstop process has been killed"
sleep "$WaitTime1"

```

```
done
fi
sleep "$WaitTime2"
casstopid=`ps -fu $runUser | fgrep casstop | fgrep -v fgrep | fgrep -v awk |awk '{print $2}'`
if test "x$casstopid" != "x"
then
    $echocmd "Surviving casstop process are :$casstopid¥n"
else
    $echocmd "***** Kill all casstop processes complete *****"
fi

###
# kill all casstop processes (End)
#####

#
# Script End
#
#*****
```

6. forceStopServer.ksh シェルスクリプト

```
#!/bin/ksh
#
#      ===== Force Stop All MFES Server Tool =====
#
# Requirements:
#      1. Have a correct $COBDIR environment.
#      2. Have the same logon user who ran "mfds" command.
#      3. Have a correct repository back up direcotory.

Usage()
{
    $echocmd "¥nUsage : ksh ./forceStopServer.ksh ¥n"
    exit 1
}

yorn ()
{
    YN=b
    while [ "$YN" != "n" -a "$YN" != "no" -a ¥
           "$YN" != "N" -a "$YN" != "NO" -a ¥
           "$YN" != "y" -a "$YN" != "yes" -a ¥
           "$YN" != "Y" -a "$YN" != "YES" -a ¥
           "$YN" != "Yes" -a "$YN" != "No" ]
    do
        $echocmd $1
        read YN
        if [ "$YN" = "y" -o "$YN" = "yes" -o "$YN" = "Yes" -o ¥
            "$YN" = "Y" -o "$YN" = "YES" ]; then
            return 1
        fi
        if [ "$YN" = "n" -o "$YN" = "no" -o "$YN" = "No" -o ¥
            "$YN" = "N" -o "$YN" = "NO" ]; then
            return 0
        fi
        $echocmd "Please input y or n "
    done
}

StopAllESSer ()
{
    $echocmd "¥n***** Force Stop All MFES Server *****¥n"
    SerLIST=`mdump $machinename | grep "^Server "|sed -e "s/Server //"`
    $echocmd "ES Server are(is): ¥n$SerLIST"
```



```

if [ ".x" != "$SerLIST.x" ]; then
  for i in $SerLIST
  do
    RegionPID=`ps -eaf | grep "cascd32 ¥/r$i" | awk '{print$2}'`
    if [ "$RegionPID.x" != ".x" ]; then
      $echocmd "¥nThe RegionPID of $i is $RegionPID¥n"
      $echocmd "casstop -r$i....."
      casstop -r$i

      x=1
      y=60
      while test $x -le $y
      do
#Check if there are cas processes (e.g. cassi32..) by check root process $RegionPID
        RRegionPID=`ps -eaf | grep -v grep | grep $RegionPID`
        if [ "$RRegionPID.x" = ".x" ]
        then
#Have no CAS processes
          x=61
          fi
          if [ $x = 60 ]; then
            $echocmd "casstop -f -r$i..."
            casstop -f -r$i
          fi
          x=`expr $x + 1`
          $echocmd ".¥c"
          sleep $WaitTime1
          done
        else
          $echocmd "$i Server already stopped."
        fi
      done
    else
      $echocmd " No MFES Servers defined or running."
    fi
  }

killallCAS ()
{
#####
# kill all cascd32 processes (Start)
###
  cascdid=`ps -fu $runUser | fgrep "$COBDIR/bin/cascd32 /r" | fgrep -v fgrep | fgrep -v awk |
  awk '{print $2}'`

```

```

if test "x$cascdid" != "x"
then
    $echo "¥n***** Kill all cascd32 processes start *****"
    $echo "Please Wait...¥n"
    $echo "cascd32 process are(is) :¥n$cascdid¥n"
    for i in $cascdid
    do
        kill -9 $i
        $echo "$i cascd32 process has been killed"
        sleep "$WaitTime1"
    done
    $echo "***** Kill all cascd32 processes complete *****"
fi
sleep "$WaitTime2"
cascdid=`ps -fu $runUser | fgrep "$COBDIR/bin/cascd32 /r" | fgrep -v fgrep | fgrep -v awk | awk
' {print $2}' `
if test "x$cascdid" != "x"
then
    $echo "Surviving cascd32 process are(is) :$cascdid¥n"
fi

###
# kill all cascd32 processes (End)
#####

#####
# kill all casjcp32 processes (Start)
###
casjcpid=`ps -fu $runUser | fgrep "casjcp32 /r" | fgrep -v fgrep | fgrep -v awk | awk ' {print $2}' `
if test "x$casjcpid" != "x"
then
    $echo "¥n***** Kill all casjcp32 processes start *****"
    $echo "Please Wait...¥n"
    $echo "All casjcp32 process are :¥n$casjcpid¥n"
    for i in $casjcpid
    do
        kill -9 $i
        $echo "$i casjcp32 process has been killed"
        sleep "$WaitTime1"
    done
    $echo "***** Kill all casjcp32 processes complete *****"
fi

```

```

sleep "$WaitTime2"
casjcpid=`ps -fu $runUser | fgrep "casjcp32 /r" | fgrep -v fgrep | fgrep -v awk | awk '{print $2}'`
if test "x$casjcpid" != "x"
then
    $echo "Surviving casjcp32 process are :$casjcpid\n"
fi

###
# kill all casjcp32 processes (End)
#####

#####
# kill all casmgr32 processes (Start)
###
casmgrid=`ps -fu $runUser | fgrep "casmgr32 /r" | fgrep -v fgrep | fgrep -v awk | awk '{print $2}'`
if test "x$casmgrid" != "x"
then
    $echo "***** Kill all casmgr32 processes start *****"
    $echo "Please Wait... \n"
    $echo "All casmgr32 process are : \n$casmgrid\n"
    for i in $casmgrid
    do
        kill -9 $i
        $echo "$i casmgr32 process has been killed"
        sleep "$WaitTime1"
    done
    $echo "***** Kill all casmgr32 processes complete *****"
fi
sleep "$WaitTime2"
casmgrid=`ps -fu $runUser | fgrep "casmgr32 /r" | fgrep -v fgrep | fgrep -v awk | awk '{print $2}'`
if test "x$casmgrid" != "x"
then
    $echo "Surviving casmgr32 process are :$casmgrid\n"
fi

###
# kill all casmgr32 processes (End)
#####

#####

```

```

# kill all cassi32 processes (Start)
###
cassiid=`ps -fu $runUser | fgrep "cassi32 /r" | fgrep -v fgrep | fgrep -v awk | awk '{print $2}'`
if test "x$cassiid" != "x"
then
    $echo "***** Kill all cassi32 processes start *****"
    $echo "Please Wait...¥n"
    $echo "All cassi32 process are : ¥n$cassiid¥n"
    for i in $cassiid
    do
        kill -9 $i
        $echo "$i cassi32 process has been killed"
        sleep "$WaitTime1"
    done
    $echo "***** Kill all cassi32 processes complete *****"
fi
sleep "$WaitTime2"
cassiid=`ps -fu $runUser | fgrep "cassi32 /r" | fgrep -v fgrep | fgrep -v awk | awk '{print $2}'`
if test "x$cassiid" != "x"
then
    $echo "Surviving cassi32 process are...$cassiid¥n"
fi

#####
# kill all cassi32 processes (End)
#####

#####
# kill all mfcs32 processes (Start)
###
mfcsid=`ps -fu $runUser | fgrep "$COBDIR/bin/mfcs32 -n" | fgrep -v fgrep | fgrep -v awk | awk '{print $2}'`
if test "x$mfcsid" != "x"
then
    $echo "***** Kill all mfcs32 processes start *****"
    $echo "Please Wait...¥n"
    $echo "All mfcs32 process are :¥n$mfcsid¥n"
    for i in $mfcsid
    do
        kill -9 $i
        $echo "$i mfcs32 process has been killed"
        sleep "$WaitTime1"
    done

```

```

$echocmd "***** Kill all mfcs32 processes complete *****"
fi
sleep "$WaitTime2"
mfcsid=`ps -fu $runUser | fgrep "$COBDIR/bin/mfcs32 -n" | fgrep -v fgrep | fgrep -v awk | awk ' {print $2}' `
if test "x$mfcsid" != "x"
then
$echocmd "Surviving mfcs32 process are :$mfcsid¥n"
fi

###
# kill all mfcs32 processes (End)
#####

#####
# kill all casstop processes (Start)
###

casstopid=`ps -fu $runUser | fgrep casstop | fgrep -v fgrep | fgrep -v awk | awk ' {print $2}' `
if test "x$casstopid" != "x"
then
$echocmd "¥n***** Kill all casstop processes start *****"
$echocmd "Please Wait...¥n"
$echocmd "All casstop process are :¥n$casstopid¥n"
for i in $casstopid
do
kill -9 $i
$echocmd "$i casstop process has been killed"
sleep "$WaitTime1"
done
$echocmd "***** Kill all casstop processes complete *****"
fi
sleep "$WaitTime2"
casstopid=`ps -fu $runUser | fgrep casstop | fgrep -v fgrep | fgrep -v awk | awk ' {print $2}' `
if test "x$casstopid" != "x"
then
$echocmd "Surviving casstop process are :$casstopid¥n"
fi

###
# kill all casstop processes (End)
#####

}

```

```

#*****
#
# Script Start here .....
#
#Maybe you need to set LANG for getting a corrent mfdsid
#LANG=ja_JP.eucJP
#export LANG

echocmd="echo"
machinename=`uname -n`
runUser=`whoami`
WaitTime1=1
WaitTime2=2
WaitTime5=5

# If not argument given the Usage displayed
test $1.x != .x && Usage

# Check if you have a correct Server Express version and $COBDIR setting
if [ ! -f $COBDIR/bin/mdump ]; then
    $echocmd "$COBDIR/bin/mdump is not found."
    $echocmd "mdump required to force stop MFES server!"
    exit 2
fi

# You should start mfd32 process before stop All MFES Server
mfd32=`ps -fu $runUser | fgrep "$COBDIR/bin/mfd32" | fgrep -v fgrep | fgrep -v awk | awk '{print $2}'`
if test "x$mfd32" = "x"
then
    $echocmd "No mfd32 process, please run mfd32 ¥n"
    exit 3
fi

# Main script are

StopAllESSer
casid=`ps -fu $runUser | fgrep "32 /r" | fgrep -v fgrep | fgrep -v awk | awk '{print $2}'`
if test "x$casid" != "x"
then
    $echocmd "Have surviving cas32 process, run killallCAS.....¥n"
    killallCAS

```

```

else
    casid=`ps -fu $runUser | fgrep "mfcs32 -n" | fgrep -v fgrep | fgrep -v awk | awk ' {print
$2}' `
    if test "$casid" != "x"
        then
            $echo "Have surviving mfcs process, run killallCAS....."
            killallCAS
        fi
    fi

$echo "***** Force Stop All MFES Server Complete *****"

#
# Script End
#
#*****

```