CLUSTERPRO® X for Linux

VMware® Infrastructure 3 Setup HOWTO

2009/02/25
第 1 版

CLUSTERPRO

目次

はじめに

本書には NEC の CLUSTERPRO を使って VMware Infrastructure 3 上にクラスタを構築する方法が記載されています。
本書が対象とする VMware ESX 及び CLUSTERPRO のバージョンは下記のとおりです。

= VMware ESX
- VMware ESX 3.5 Update 2

= CLUSTERPRO (ホスト上)
- CLUSTERPRO X 2.0 for Linux (内部バージョン : 2.0.2-1)

= CLUSTERPRO (ゲスト上)
- ゲスト上の OS をサポートしている CLUSTERPRO

免責事項

本書の内容に対して、いかなる保障も提供されません。

商標

CLUSTERPRO は日本電気株式会社の登録商標です。
VMware は米国 VMware, Inc.の登録商標です。
VMotion は米国 VMware, Inc.の商標です。
Linux は米国及びその他の国における Linus Torvalds の登録商標です。

省略

本書では以下の省略をした単語を使用します。

ESX	VMware ESX Server の動作している物理マシン
PM	VMware ESX Server 以外の OS が動作している物理マシン
VM	VMware ESX Server 上で動作している仮想マシン
CLS	CLUSTERPRO
SSS	SingleServerSafe
AP	業務アプリケーション

注意

1. VM をクラスタ化する場合に使用する CLUSTERPRO は、VM にインストールする OS をサポートするものを使用してください。
2. ESX をクラスタ化する場合は、 ”CLUSTERPRO X 2.0 for Linux (内部バージョン 2.0.2-1 以降)” を利用してください。
3. ESX 上では、Replicator、Replicator DR 及び各種 Agent は利用できません。
4. VM の (仮想) NIC は MII をサポートしていません。このため、VM に CLUSTERPRO をインストールした場合、 ”NIC Link Up/Down モニタリソース” を使用することはできません。

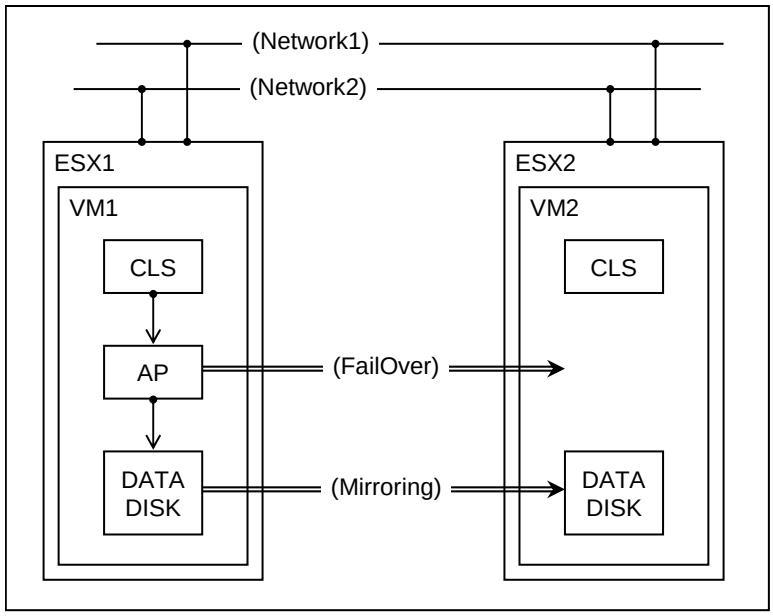
詳細

VM-VM データミラー型クラスタ

特徴

1. 各 ESX は VM を持ち、各 VM からクラスタが構成されます。
2. CLUSTERPRO は VM 上にインストールされ、AP を監視します。AP はクラスタ内でフェイルオーバします。
3. AP のデータは CLUSTERPRO によってミラーされるディスク領域に格納されます。

概要図



セットアップ手順

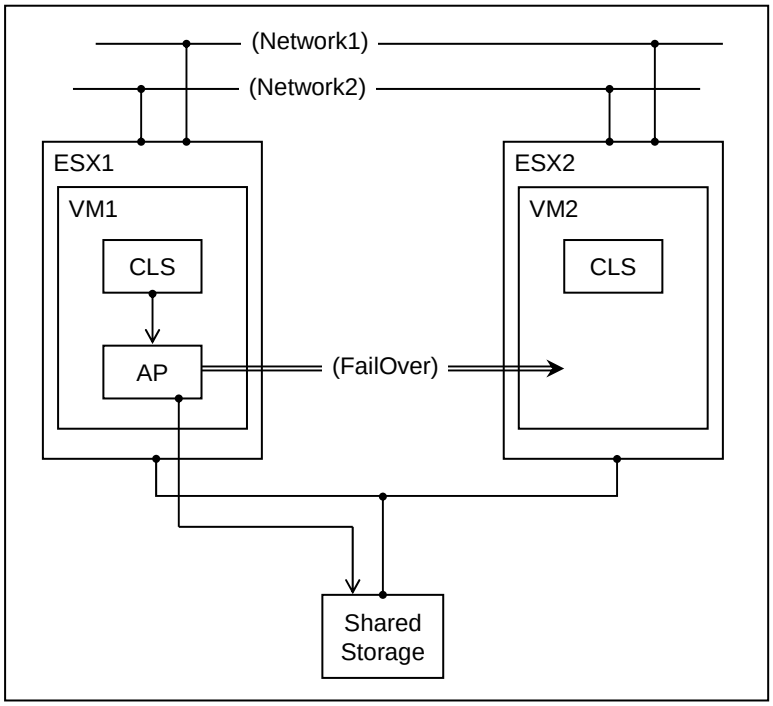
1. ミラーされるデータパーティション (とそれに対応するクラスタパーティション) を持つ VM1 を作成します。
2. VM1 から VM2 と言う名前のクローンを作ります。
3. 各 VM に OS と CLUSTERPRO をインストールします。手順は物理サーバに対する手順と同じです。(CLUSTERPRO のマニュアルを参照してください)

VM-VM 共有ディスク型クラスタ

特徴

1. 各 ESX は VM を持ち、各 VM からクラスタが構成されます。
2. CLUSTERPRO は VM 上にインストールされ、AP を監視します。AP はクラスタ内でフェイルオーバします。
3. AP のデータは共有ディスクに格納されます。
4. VM 上の CLUSTERPRO から共有ディスクを参照するためには、仮想 SCSI HBA の "SCSI Bus Sharing" 設定を有効にする必要があります。

概要図



セットアップ手順

1. NIC を 2 つ持つ VM1、VM2 を作成します。
2. VM1、2 に共有ディスクを追加します。これには 2 つの選択肢があります。環境や目的に応じて選択してください。
    1. 物理的な方法では、RDM (Raw Device Mapping) を使って物理的な共有ディスクを VM に追加します。

       VMware Virtual Console を使って RDM を作成します。 (以下はサンプルです)

```
VM1 アイコンをクリック -> [Summary] タブ -> [Edit Settings]
-[Add...]-> [Hard Disk]
-[Next]-> [Raw Device Mappings]
-[Next]-> (物理)LUN を選択します。
-[Next]-> [Store with Virtual Machine]
-[Next]-> [Physical]
-[Next]-> [Node] に [SCSI(1:0)] を指定します。(新しい仮想デバイスノードを指定します)
-[Next]-> [Finish]
[OK]
```

VM2 に対して下記の手順で RDM を作成します。

```
VM2 アイコンをクリック -> [Summary] タブ -> [Edit Settings]
-[Add...]-> [Hard Disk]
-[Next]-> [Use an existing virtual disk]
-[Next]-> [Browse...]->VM1 で作成した(RDM)仮想ディスクを選択します。デフォルトで作成されるマッピングされた仮想ディスクは、VM1 の仮想ディスクがあるディレクトリ内に"VM1 の仮想ディスク"_1.vmdk で作成されます。
-[Next]-> [Node] に [SCSI(1:0)] を指定します。(新しい仮想デバイスノードを指定します)
-[Next]-> [Finish]
[OK]
```

2. 仮想的な方法では、共有ディスク上に「仮想共有ディスク」を作成し、VM に追加します。

   ESX のコンソール OS から以下のコマンドで共有ディスク上に「仮想共有ディスク」を作成します。

   (以下はサンプルです。)

```
% vmkfstools -c 500m -d thick ¥
> /vmfs/volumes/shared-storage/vsd/vsd.vmdk
```

   VMware Virtual Console を使って仮想共有ディスクを両 VM に追加します。

```
VM1 アイコンをクリック -> [Summary] タブ -> [Edit Settings]
-[Add...]-> [Hard Disk]
-[Next]-> [Use an existing virtual disk]
-[Next]-> 上で作成した .vmdk ファイルを指定します。
-[Next]-> [Node] に [SCSI(1:0)] を指定します。(新しい仮想デバイスノードを指定します)
-[Next]-> [Finish]
[OK]
```

   仮想共有ディスクは複数の VM からアクセス可能である必要があるため、両 VM の仮想 SCSI コントローラを設定します。

```
VM1 アイコンをクリック-> [Summary] タブ -> [Edit Settings]
上で作成した .vmdk に接続されている [SCSI Controller *] を選択します。
[SCSI Bus Sharing] を [Physical] に変更します。
[OK]
```

3. 各 VM に OS と CLUSTERPRO をインストールします。手順は物理サーバに対するものと同様です。

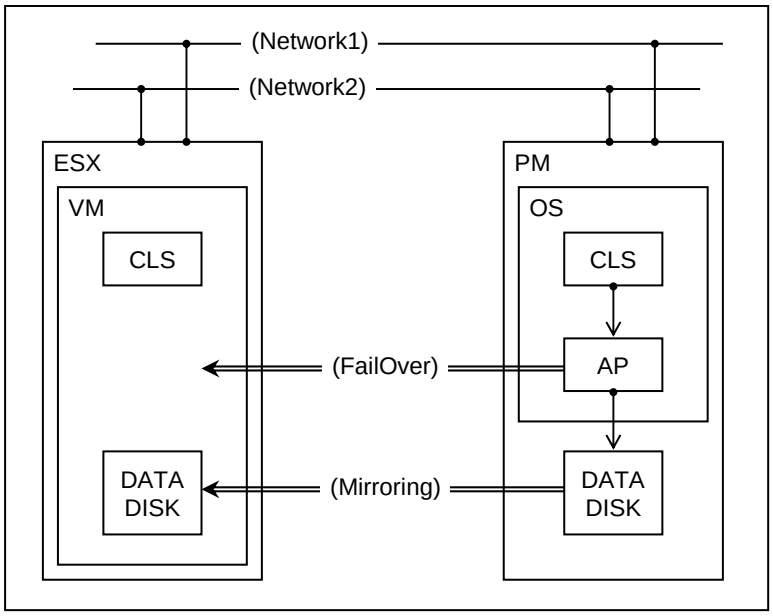
   (CLUSTERPRO のマニュアルを参照)

VM-PM データミラー型クラスタ

特徴

1. ESX は VM を持ち、VM と PM からクラスタが構成されます。
2. CLUSTERPRO は VM と PM 上にインストールされ、AP を監視します。AP はクラスタ内でフェイルオーバします。
3. AP のデータは CLUSTERPRO によってミラーされるディスク領域に格納されます。

概要図



セットアップ手順

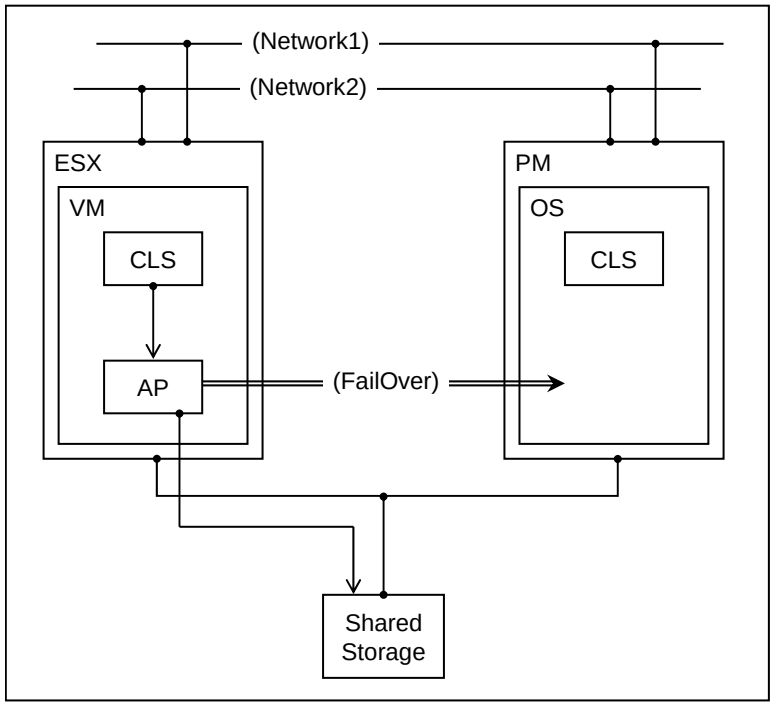
1. ミラーされるデータディスク (とそれに対応するクラスタパーティション) を持つ PM を構築します。
2. VM を作成します。
3. OS と CLUSTERPRO をインストールします。手順は物理サーバに対する手順と同じです。(CLUSTERPRO のマニュアルを参照してください)

VM-PM 共有ディスク型クラスタ

特徴

1. ESX は VM を持ち、VM と PM でクラスタを構成します。
2. CLUSTERPRO は AP を監視し、AP はクラスタ内でフェイルオーバします。
3. AP のデータは共有ディスクに格納します。

概要図



セットアップ手順

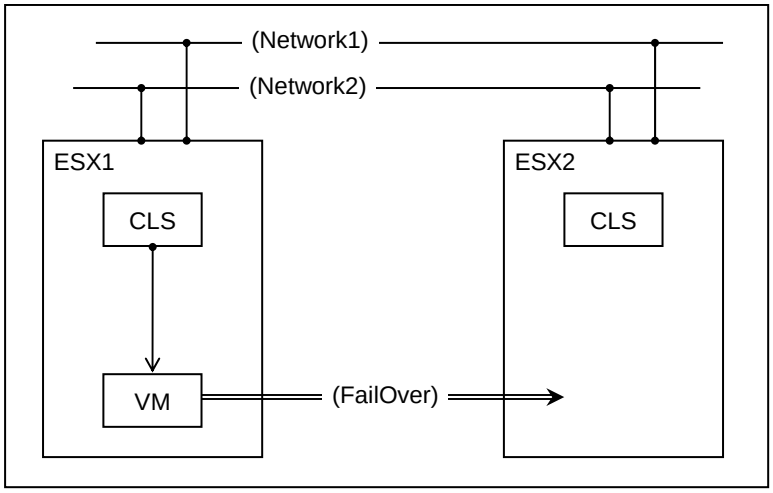
1. PM と共有ディスクを設定します。共有ディスク上には データパーティションとハートビートパーティションを用意しておきます。
2. VM を作成します。VM に RDM (Raw Device Mapping) を作成します。
3. OS と CLUSTERPRO をインストールします。手順は物理サーバに対する手順と同じです。(CLUSTERPRO のマニュアルを参照)

ESX-ESX クラスタ

特徴

1. ESX 上でクラスタが構成されます。
2. CLUSTERPRO は各 ESX のコンソール OS 上にインストールされます。
3. CLUSTERPRO は VM を監視することができます。
4. 複数の VM から構成されるクラスタではないため、VM にインストールされた OS やアプリケーションのローリングメンテナンス (現用系で業務を継続させたまま、待機系にアップデートやパッチを適用する運用) ができません。

概要図



セットアップ手順

1. VMを作成し、OSをインストールします。この際、 "CLUSTERPRO SingleServerSafe" もインストールすることを推奨します。
2. ESX コンソール OS 上で、CLUSTERPRO が使用するポート番号を開放するようファイヤーウォールの設定を変更します。開放が必要なポート番号については CLUSTERPRO マニュアルを参照してください。
3. ESX コンソール OS 上に CLUSTERPRO (clusterpro-2.0.2-1.vmware.i386.rpm) をインストールします。 "CLUSTERPRO Builder" で設定を作る際のポイントは以下の通りです。 (※スクリプトファイルの内容は付録を参照してください)
   i. クラスタを追加します。
   ii. サーバを追加します。
   iii. "vmgrp" という名前 (例) のフェイルオーバグループを追加します。
   iv. "vmgrp" に "vmpower" という名前 (例) の “execute resource” を追加します。このリソース

は対象VMの電源制御 (On/Off) を行います。

[置換] ボタンでstart.shの内容を vmpower.start.plで置き換えます。同様にstop.shもvmpower.stop.plで置き換えます。

[編集] ボタンで各スクリプト内の設定 (VM設定ファイルの格納パス) を環境に応じた値に変更します。

これらのスクリプトは以下の機能を実装しています。

1. フェイルオーバグループが起動する際、start.sh は VM の電源状態が ON ではない場合に、ON にします。
2. フェイルオーバグループが停止する際、stop.sh は VM の電源状態が OFF ではない場合に、OFF にします。 また、VM の電源状態が OFF になるまで待ちます。

v. "Monitors" 配下に "vmgenw" という名前 (例) の "custom monitor resource" を追加します。これは、上記 “vmpower” で起動した対象VMを監視するためのものです。異常検出時には再活性させず、すぐにフェイルオーバさせるように設定することを推奨します。 (再活性化閾値 0、フェイルオーバ閾値 1 以上を推奨します)

[置換] ボタンで genw.sh の内容を clpvmmon.plで置き換えます。

[編集] ボタンでスクリプト内の設定 (VM設定ファイルの格納パス) を環境に応じた値に変更します。

“監視タイプ” を非同期に変更します。

これらのスクリプトは以下の機能を実装しています。

1. genw.sh は対象 VM の電源状態と ESX-VM 間のハートビートの状態を監視します。
2. 対象 VM の電源状態が ON ではなかったり、ハートビートが止まっている事を検出した場合、genw.sh は終了します。
   “vmgenw” が異常を検出するのは、”vmgenw” の genw.sh が終了した場合、つまり、対象 VM の電源やハートビートが異常な状態になった場合です。

vi. 設定を作り終えたら 保存します。

vii. 作成した設定データを使ってクラスタを構築します。

監視処理の流れの概要は以下のようになります。

[vmgenw] -> [VM]

VM の異常を検出すると、”vmgenw” の genw.sh が終了します。

“vmgenw” がその終了を検出すると、フェイルオーバを実行します。

すると、フェイルオーバグループ "vmgrp" は他の PM で起動され、対象 VM が起動されます。

注意

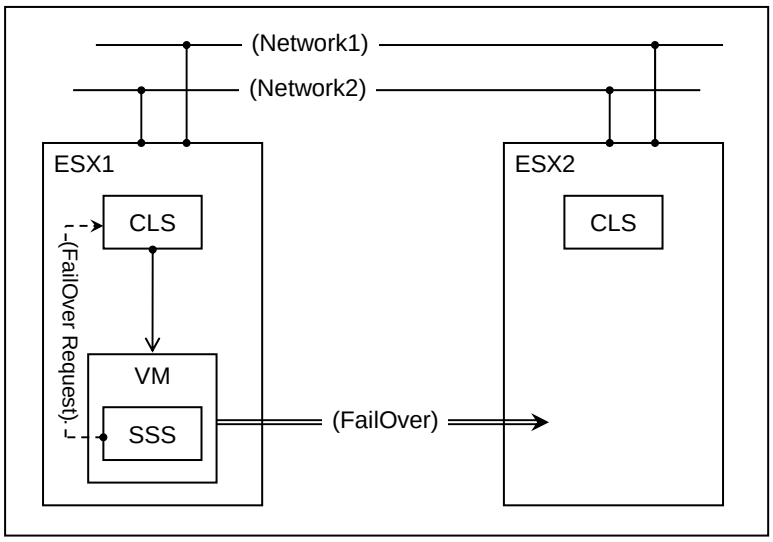
1. ESX-ESX クラスタでは、データミラー型クラスタを利用することはできません。
2. ESX-ESX クラスタでは、 “ユーザ空間監視リソース(userw)” で “softdog” は利用できません。

ESX-ESX クラスタ (VM-ESX 連携)

特徴

1. "CLUSTERPRO SingleServerSafe" を使用することで、VM 上の AP の可用性を向上させることができます。
2. VM の SSS が障害を検出した場合、ESX のクラスタへ VM のフェイルオーバ要求を発行することができます。
3. クラスタ間のフェイルオーバ要求には clptrnreq コマンドを使用します。
4. VM はフェイルオーバによってクラスタを構成する ESX 間を移動します。
5. VM の OS は Windows、Linux に対応しています。

概要図



セットアップ手順

1. ESX にクラスタを構築します。手順は前章のセットアップ手順を参照してください。
2. VM 上の OS に "CLUSTERPRO SingleServerSafe" をインストールします。
   特に通常 OS のインストール手順と変更ありません。
3. "CLUSTERPRO Builder" で VM のクラスタ構成情報を編集します。スクリプトの内容は付録を参照してください。
   i. 監視対象のモニタリソース (pid モニタリソース、oracle モニタリソースなど) を追加します。
   ii. モニタリソースの "最終動作前スクリプトを実行する" を有効にして、[設定]ボタンを選択します。
   iii. [置換] ボタンでpreaction.shの内容を vmpreaction.shで置き換えます。
   [編集] ボタンでスクリプト内の設定 (ESXが起動するVMのリソース名、ESXのIPアドレス) を環境に応じた値に変更します。

注意

1. clptrnreq コマンドの詳細に関しては、「CLUSTERPRO X2.0 for Linux リファレンスガイド」を参照してください。

VMotion について

利用時の注意

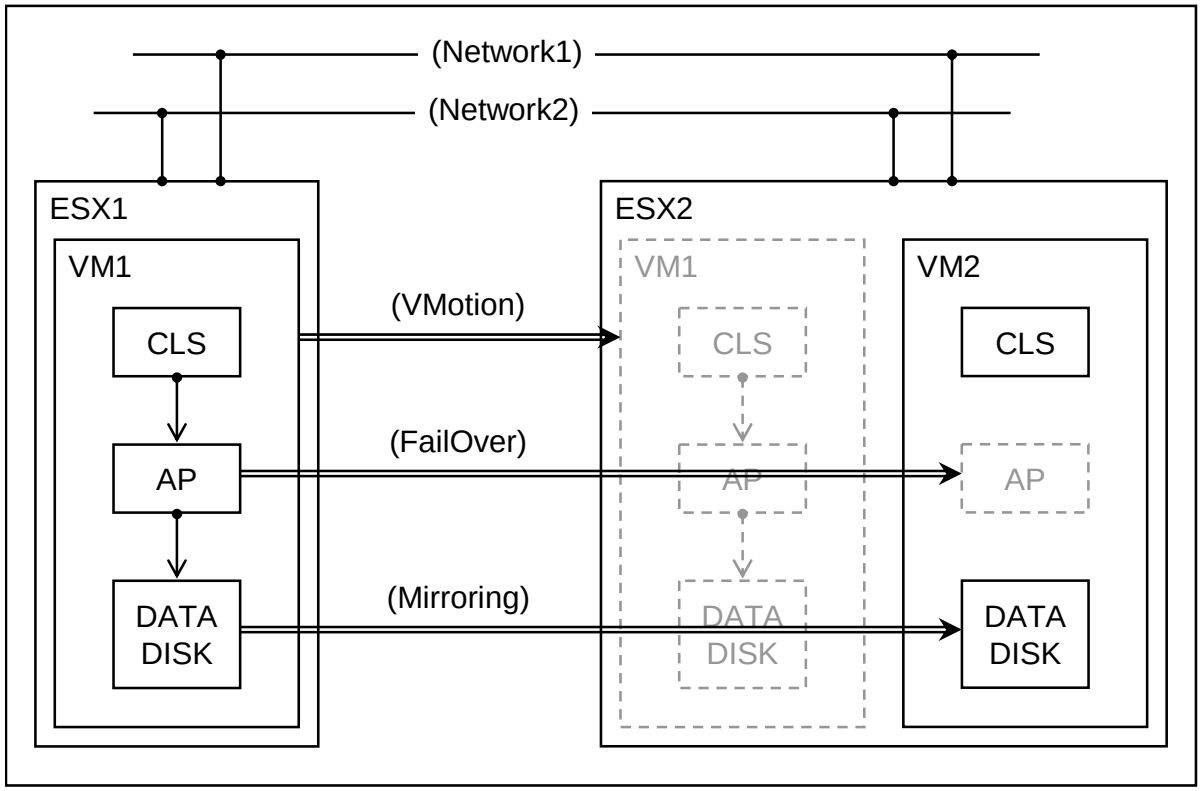
1. VM の設定ファイル及びディスクイメージは、共有ディスク上に配置する必要があります。
2. VMware の仕様により、仮想 SCSI HBA の "SCSI Bus Sharing" の設定を "None" 以外にすると VMotion が利用できなくなります。このため、”VM-VM 共有ディスク型クラスタ” 内の VM は Vmotion による他のホストへの移動はできません。

VMotion 併用可否

	共有ディスク型クラスタ	データミラー型クラスタ
VM-VM	×	○
VM-PM	○	○
ESX-ESX	○	-

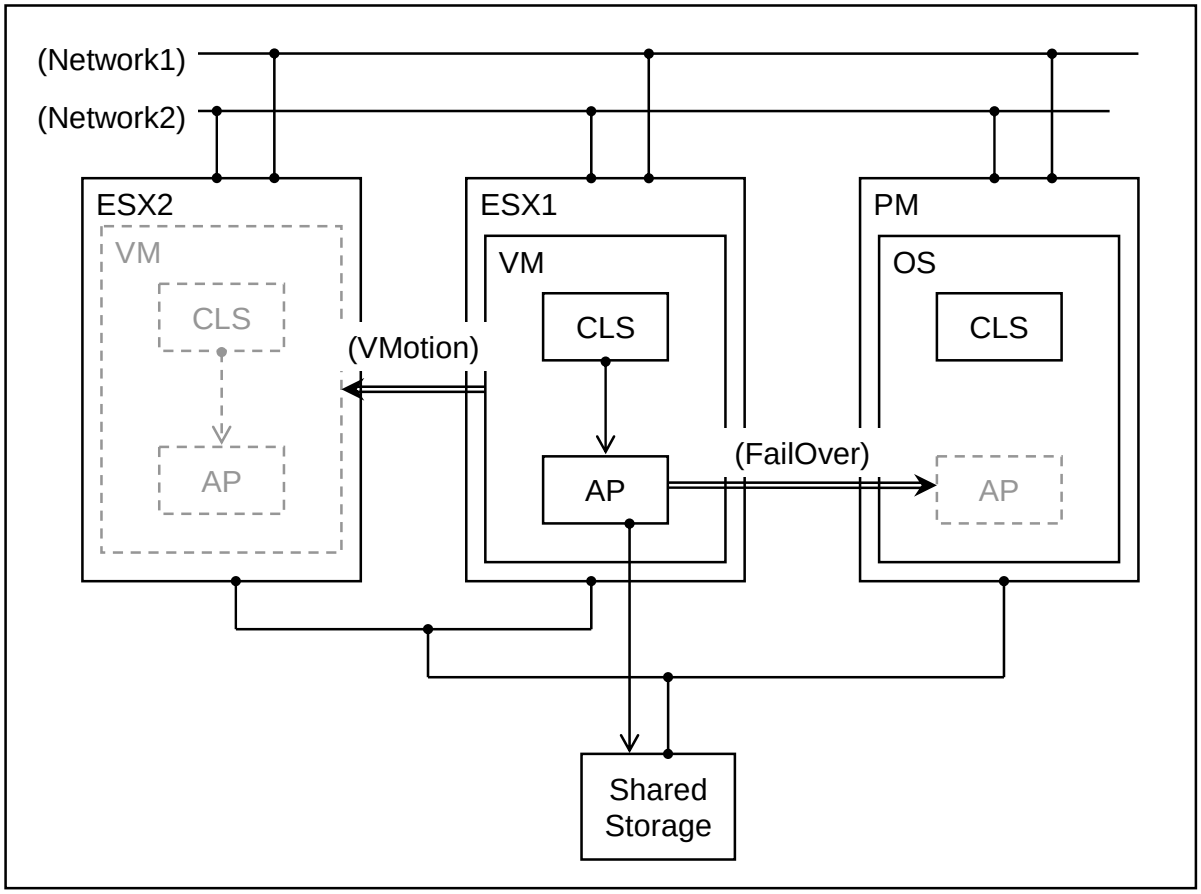
VMotion 併用時の構成例

VM-VM データミラー型クラスタ



上記の構成では、ESX1 上の VM1 及び VM2 を ESX1、ESX2 どちらへも移動できます。
VM1 および VM2 が共有ディスクに接続されていないため、SCSI Bus Sharing の設定を"none"にできます。

VM-PM 共有ディスク型クラスタ

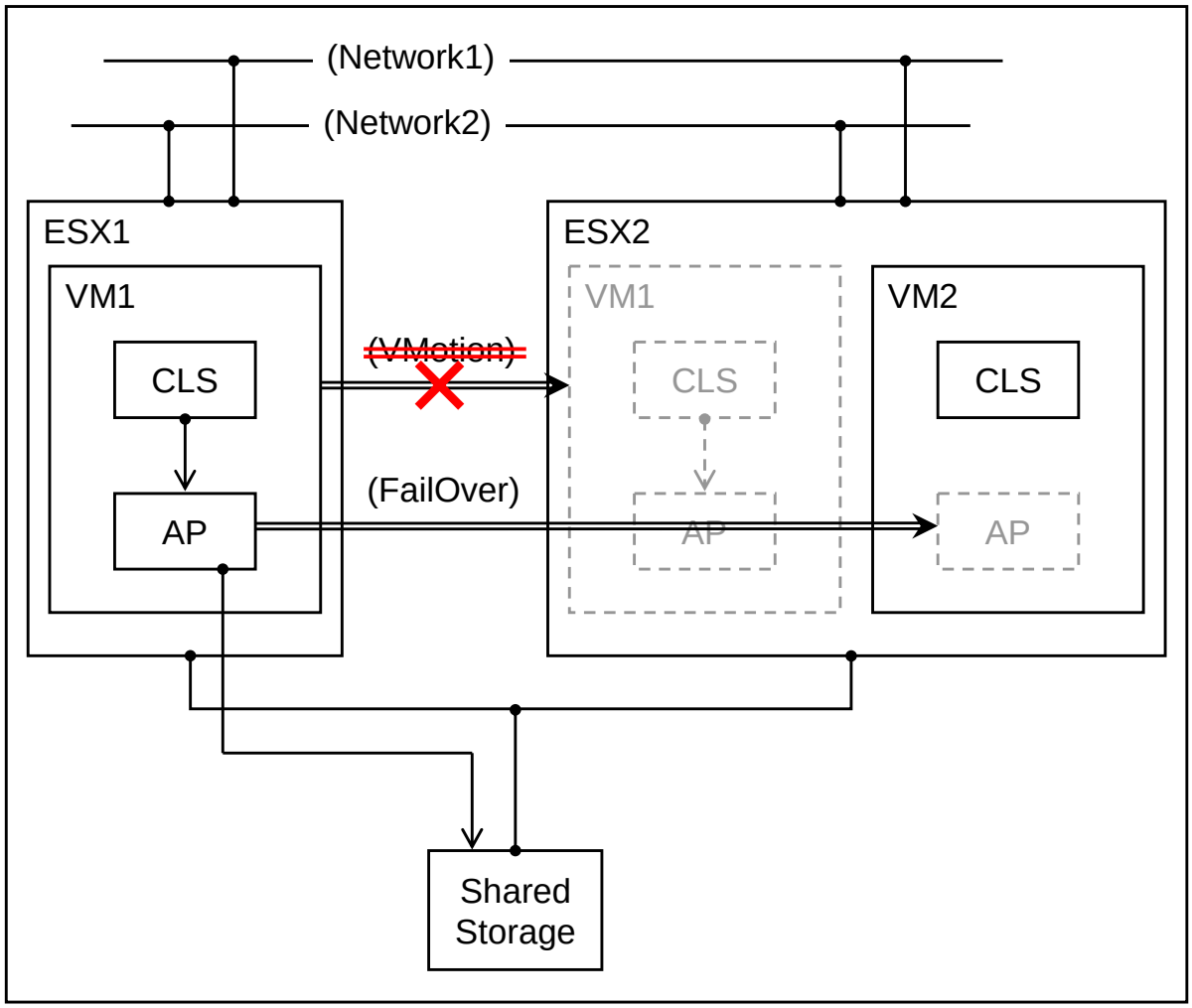


上記の構成の場合、ESX1 上の VM を ESX2 へ移動できます。

ESX1 上または ESX2 上で、共有ディスクを持つ複数の VM が同時に起動されないため、SCSI Bus Sharing の設定を"none"にできます。

VMotion 併用不可能な構成例

VM-VM 共有ディスク型クラスタ



上記のような構成の場合、ESX1 上の VM1 を ESX2 に移動することはできません。

同時に起動される VM1 と VM2 の両方から Shared Storage が見えていないといけないため、SCSI Bus Sharing の設定を"none"にできません。

付録

vmpower.start.pl

```
#!/usr/bin/perl -w

#
# Script for powere on the VM
#

#--------------------------------------------------------------------------------
# Configuration
#--------------------------------------------------------------------------------
# The path to VM configuration file. This must be absolute UUID-based path.
my $cfg_path  = "/vmfs/volumes/456430d5-96915600-1f3a-0011095c26be/vm3-1/vm3-1.vmx";
#--------------------------------------------------------------------------------
my $vmname = $cfg_path;  # VMname to be outputted on log.
$vmname =~ s/^(.*¥/)(.*)(¥.vmx)/$2/;

use VMware::VmPerl;
use VMware::VmPerl::VM;
use VMware::VmPerl::Server;
use VMware::VmPerl::ConnectParams;
use VMware::VmPerl::Question;
use strict;

my $state_string_map = {};
my @state_strings = (
        "VM_EXECUTION_STATE_ON",
        "VM_EXECUTION_STATE_OFF",
        "VM_EXECUTION_STATE_SUSPENDED",
        "VM_EXECUTION_STATE_STUCK",
        "VM_EXECUTION_STATE_UNKNOWN"
);

foreach my $state_string (@state_strings) {
   $state_string_map->{&vm_constant($state_string)} = $state_string;
}
#--------------------------------------------------------------------------------
if(&IsPoweredOn()){
        exit 0;
}
else{
        if(&PowerOn()){
                exit 0;
        }
        else{
                exit 1;
        }
}
exit 0;
#--------------------------------------------------------------------------------
# Functions
#--------------------------------------------------------------------------------
sub IsPoweredOn{
        my $vm = VMware::VmPerl::VM::new();
        if(!&ConnectVm($vm)){
                my ($error_number, $error_string) = $vm->get_last_error();
                &Log("[E] [$vmname] at localhost could not connect: " .
                        "$error_number : $error_string¥n");
                undef $vm;
                &Log("[D] [$vmname] at localhost disconnected.¥n");
                return 0;
        }
        else{
```

```
                &Log("[D] [$vmname] at localhost connected.¥n");
        }

        my $cur_state = $vm->get_execution_state();
        if (!defined($cur_state)) {
                my ($error_number, $error_string) = $vm->get_last_error();
                &Log("[E] [$vmname] at localhost could not get power status: " .
                        "$error_number : $error_string.¥n");
                undef $vm;
                &Log("[D] [$vmname] at localhost disconnected.¥n");
                return 0;
        }

        &Log("[D] [$vmname] at localhost [$state_string_map->{$cur_state}]¥n");
        if($cur_state ne &vm_constant("VM_EXECUTION_STATE_ON")){
                &Log("[W] [$vmname] at localhost is not powered on.¥n");
                &Log("[D] [$vmname] at localhost disconnected.¥n");
                undef $vm;
                return 0;
        }

        undef $vm;
        &Log("[D] [$vmname] at localhost disconnected.¥n");
        return 1;
}
#--------------------------------------------------------------------------------
sub RegisterVm{
        my $connect_params = VMware::VmPerl::ConnectParams::new(undef, undef, undef, undef);
        my $server = VMware::VmPerl::Server::new();
        if(!$server->connect($connect_params)){
                my ($error_number, $error_string) = $server->get_last_error();
                &Log("[E] Could not connect to localhost: Error $error_number:
$error_string¥n");
                undef $server;
                return 0;
        }

        my @list = $server->registered_vm_names();
        if(!defined $list[0]){
                if(!$server->register_vm($cfg_path)){
                        my ($error_number, $error_string) = $server->get_last_error();
                        &Log("[E] Could not register VM: Error $error_number:
$error_string¥n");
                        undef $server;
                        return 0;
        }
                else{
                        &Log("[I] [$vmname] at localhost registered.¥n");
                        undef $server;
                        return 1;
                }
        }
        else{
                foreach(@list){
                        if(/$cfg_path/){
                                &Log("[I] [$vmname] at localhost already registered.¥n");
                                undef $server;
                                return 1;
                        }
                }

                if(!$server->register_vm($cfg_path)){
                        my ($error_number, $error_string) = $server->get_last_error();
                        &Log("[E] Could not register VM: Error $error_number:
$error_string¥n");
                        undef $server;
                        return 0;
                }
                else{
                        &Log("[I] [$vmname] at localhost registered VM.¥n");
                        undef $server;
                        return 1
```

```
                }
        }
}
#-------------------------------------------------------------------------------
sub PowerOn{
        if(!&RegisterVm()){
                return 0;
        }

        my $vm = VMware::VmPerl::VM::new();
        if(!&ConnectVm($vm)){
                my ($error_number, $error_string) = $vm->get_last_error();
                &Log("[E] [$vmname] at localhost could not connect: " .
                        "$error_number : $error_string¥n");
                undef $vm;
                &Log("[D] [$vmname] at localhost disconnected.¥n");
                return 0;
        }
        else{
                &Log("[D] [$vmname] at localhost connected.¥n");
        }

        if(!$vm->start(vm_constant("VM_POWEROP_MODE_TRYSOFT"))){
                my ($error_number, $error_string) = $vm->get_last_error();
                &Log("[E] Could not start VM: Error $error_number: $error_string¥n");

                my $cur_state = $vm->get_execution_state();
                if($cur_state eq &vm_constant("VM_EXECUTION_STATE_STUCK")){
                        my $q = $vm->get_pending_question();
                        if(!defined $q){
                                undef $vm;
                                &Log("[D] [$vmname] at localhost disconnected.¥n");
                                return 0;
                        }
                        &Log("[I] Question and Choices¥n");
                        &Log("-------------------------¥n");
                        print $q->get_text() . "¥n";
                        my @choices = $q->get_choices();
                        for (my $i = 0; $i <= $#choices; $i++) {
                                print "¥t$i) $choices[$i]¥n";
                        }
                        &Log("-------------------------¥n");
                        $vm->answer_question($q, "1");     # keep vm config.
                        &Log("[I] Answered (1)¥n");
                }
                else{
                        &Log("[D] [$vmname] at localhost
[$state_string_map->{$cur_state}]¥n");
                        undef $vm;
                        &Log("[D] [$vmname] at localhost disconnected.¥n");
                        return 0;
                }
        }
        &Log("[I] [$vmname] at localhost started.¥n");
        undef $vm;
        &Log("[D] [$vmname] at localhost disconnected.¥n");
        return 1;
}
#-------------------------------------------------------------------------------
sub vm_constant {
        my $constant_str = shift;
        return VMware::VmPerl::constant($constant_str, 0);
}
#-------------------------------------------------------------------------------
sub Log{
        my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime(time);
        $year += 1900;
        $mon += 1;
        my $date = sprintf "%d/%02d/%02d %02d:%02d:%02d", $year, $mon, $mday, $hour, $min,
$sec;
        print "$date $_[0]";
        return 0;
```

```
}
#--------------------------------------------------------------------------------
sub ConnectVm{
        my $connect_params = VMware::VmPerl::ConnectParams::new(undef, undef, undef, undef);
        if (! $_[0]->connect($connect_params, $cfg_path)) {
                return 0;
        }
        else{
                return 1;
        }
}
```

vmpower.stop.pl

```
#!/usr/bin/perl -w

#--------------------------------------------------------------------------------
# Configuration
#--------------------------------------------------------------------------------
# The path to VM configuration file. This must be absolute UUID-based path.
my $cfg_path  = "/vmfs/volumes/456430d5-96915600-1f3a-0011095c26be/vm3-1/vm3-1.vmx";
# The interval to check the vm status. (second)
my $interval  = 1;
#--------------------------------------------------------------------------------
my $vmname = $cfg_path;  # VMname to be outputted on log.
$vmname =~ s/^(.*¥/)(.*)(¥.vmx)/$2/;

use VMware::VmPerl;
use VMware::VmPerl::VM;
use VMware::VmPerl::Server;
use VMware::VmPerl::ConnectParams;
use VMware::VmPerl::Question;
use strict;

my $state_string_map = {};
my @state_strings = (
        "VM_EXECUTION_STATE_ON",
        "VM_EXECUTION_STATE_OFF",
        "VM_EXECUTION_STATE_SUSPENDED",
        "VM_EXECUTION_STATE_STUCK",
        "VM_EXECUTION_STATE_UNKNOWN"
);

foreach my $state_string (@state_strings) {
   $state_string_map->{&vm_constant($state_string)} = $state_string;
}
#--------------------------------------------------------------------------------
if(&IsPoweredOn()){
        if(&PowerOff()){
                while(&IsHbIncrease()){
                }
                exit 0;
        }
        else{
                exit 1;
        }
}
else{
        exit 0;
}
exit 0;

#--------------------------------------------------------------------------------
# Functions
#--------------------------------------------------------------------------------
sub IsPoweredOn{
        my $vm = VMware::VmPerl::VM::new();
        if(!&ConnectVm($vm)){
                my ($error_number, $error_string) = $vm->get_last_error();
```

```
                &Log("[E] [$vmname] at localhost could not connect: " .
                        "$error_number : $error_string¥n");
                undef $vm;
                &Log("[D] [$vmname] at localhost disconnected.¥n");
                return 0;
        }
        else{
                &Log("[D] [$vmname] at localhost connected.¥n");
        }

        my $cur_state = $vm->get_execution_state();
        if (!defined($cur_state)) {
                my ($error_number, $error_string) = $vm->get_last_error();
                &Log("[E] [$vmname] at localhost could not get power status: " .
                        "$error_number : $error_string.¥n");
                undef $vm;
                &Log("[D] [$vmname] at localhost disconnected.¥n");
                return 0;
        }

        &Log("[D] [$vmname] at localhost [$state_string_map->{$cur_state}]¥n");
        if($cur_state ne &vm_constant("VM_EXECUTION_STATE_ON")){
                &Log("[W] [$vmname] at localhost is not powered on.¥n");
                &Log("[D] [$vmname] at localhost disconnected.¥n");
                undef $vm;
                return 0;
        }

        undef $vm;
        &Log("[D] [$vmname] at localhost disconnected.¥n");
        return 1;
}
#-------------------------------------------------------------------------------
sub PowerOff{
        my $vm = VMware::VmPerl::VM::new();
        if(!&ConnectVm($vm)){
                my ($error_number, $error_string) = $vm->get_last_error();
                &Log("[E] [$vmname] at localhost could not connect: " .
                        "$error_number : $error_string¥n");
                undef $vm;
                &Log("[D] [$vmname] at localhost disconnected.¥n");
                return 0;
        }
        else{
                &Log("[D] [$vmname] at localhost connected.¥n");
        }

        my $status = $vm->stop(vm_constant("VM_POWEROP_MODE_TRYSOFT"));
        if(!defined $status){
                my ($error_number, $error_string) = $vm->get_last_error();
                &Log("[E] Could not stop (TRYSOFT) VM: $error_number: $error_string¥n");

                $status = $vm->stop(vm_constant("VM_POWEROP_MODE_HARD"));
                if(!defined $status){
                        my ($error_number, $error_string) = $vm->get_last_error();
                        &Log("[E] Could not stop (HARD) VM: $error_number:
$error_string¥n");
                        return 0;
                }
                else{
                        &Log("[I] [$vmname] at localhsot stopped. (HARD)¥n");
                }
        }
        else{
                &Log("[I] [$vmname] at localhost stopped. (TRYSOFT)¥n");
        }
        undef $vm;
        &Log("[I] [$vmname] at localhost was disconnected.¥n");
        return 1;
}
#-------------------------------------------------------------------------------
sub IsPoweredOn2{
```

```
        my $cur_state = $_[0]->get_execution_state();
        if (!defined($cur_state)) {
                my ($error_number, $error_string) = $_[0]->get_last_error();
                &Log("[E] [$vmname] at localhost could not get power status: " .
                        "$error_number : $error_string.¥n");
                return 0;
        }

#       &Log("[D] [$vmname] at localhost [$state_string_map->{$cur_state}]¥n");
        if($cur_state ne &vm_constant("VM_EXECUTION_STATE_ON")){
                &Log("[W] [$vmname] at localhost is not powered on.¥n");
                return 0;
        }
        else{
                return 1;
        }
        return 0;
}
#-------------------------------------------------------------------------------
sub IsHbIncrease{
        my $vm = VMware::VmPerl::VM::new();
        if(!&ConnectVm($vm)){
                undef $vm;
                &Log("[D] [$vmname] at localhost disconnected.¥n");
                return 0;
        }
        else{
#               &Log("[D] [$vmname] at localhost connected.¥n");
        }
        my $last_hb = -1;
        for(my $i = 0; $i < 2; $i++){
                if(&IsPoweredOn2($vm)){
                        my $hb = $vm->get_heartbeat();
                        if(!defined $hb) {
                                my ($error_number, $error_string) =
$vm->get_last_error();
                                &Log("[E] Could not got HB count: Error $error_number:
$error_string¥n");
                                undef $vm;
                                &Log("[D] [$vmname] at localhost disconnected.¥n");
                                return 0;
                        }
                        else{
#                               &Log("[D] Got HB count $hb.¥n");
                        }

                        if ($hb == $last_hb) {
                                &Log("[E] [$vmname] is stalled.¥n");
                                undef $vm;
                                &Log("[D] [$vmname] at localhost disconnected.¥n");
                                return 0;
                        }
                        else{
                                $last_hb = $hb;
                        }
                }
                else{
                        undef $vm;
#                       &Log("[D] [$vmname] at localhost disconnected.¥n");
                        return 0;
                }
                sleep $interval;
        }

        undef $vm;
#       &Log("[D] [$vmname] at localhost disconnected.¥n");
        return 1;
}
#-------------------------------------------------------------------------------
sub vm_constant {
        my $constant_str = shift;
        return VMware::VmPerl::constant($constant_str, 0);
```

```
}
#-------------------------------------------------------------------------------
sub Log{
        my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime(time);
        $year += 1900;
        $mon += 1;
        my $date = sprintf "%d/%02d/%02d %02d:%02d:%02d", $year, $mon, $mday, $hour, $min,
$sec;
        print "$date $_[0]";
        return 0;
}
#-------------------------------------------------------------------------------
sub ConnectVm{
        my $connect_params = VMware::VmPerl::ConnectParams::new(undef, undef, undef, undef);
        if (! $_[0]->connect($connect_params, $cfg_path)) {
                return 0;
        }
        else{
                return 1;
        }
}
```

clpvmmon.pl

```
#!/usr/bin/perl -w

#-------------------------------------------------------------------------------
# Configuration
#-------------------------------------------------------------------------------

# The path to VM configuration file. This must be absolute UUID-based path.
my $cfg_path  = "/vmfs/volumes/456430d5-96915600-1f3a-0011095c26be/vm3-1/vm3-1.vmx";

# The interval to check the vm status. (second)
my $interval  = 1;
#-------------------------------------------------------------------------------
my $vmname = $cfg_path;  # VMname to be outputted on log.
$vmname =~ s/^(.*¥/)(.*)(¥.vmx)/$2/;

use VMware::VmPerl;
use VMware::VmPerl::VM;
use VMware::VmPerl::Server;
use VMware::VmPerl::ConnectParams;
use VMware::VmPerl::Question;
use strict;

my $state_string_map = {};
my @state_strings = (
        "VM_EXECUTION_STATE_ON",
        "VM_EXECUTION_STATE_OFF",
        "VM_EXECUTION_STATE_SUSPENDED",
        "VM_EXECUTION_STATE_STUCK",
        "VM_EXECUTION_STATE_UNKNOWN"
);

foreach my $state_string (@state_strings) {
   $state_string_map->{&vm_constant($state_string)} = $state_string;
}
#-------------------------------------------------------------------------------
system("¥"ulimit¥" -s unlimited");
while(&IsHbIncrease()){
}
exit 0;
#-------------------------------------------------------------------------------
# Functions
#-------------------------------------------------------------------------------
sub IsPoweredOn{
        my $cur_state = $_[0]->get_execution_state();
        if (!defined($cur_state)) {
```

```
                my ($error_number, $error_string) = $_[0]->get_last_error();
                &Log("[E] [$vmname] at localhost could not get power status: " .
                        "$error_number : $error_string.¥n");
                return 0;
        }

        &Log("[D] [$vmname] at localhost [$state_string_map->{$cur_state}]¥n");
        if($cur_state ne &vm_constant("VM_EXECUTION_STATE_ON")){
                &Log("[W] [$vmname] at localhost is not powered on.¥n");
                return 0;
        }
        else{
                return 1;
        }
        return 0;
}
#--------------------------------------------------------------------------------
sub IsHbIncrease{
        my $vm = VMware::VmPerl::VM::new();
        if(!&ConnectVm($vm)){
                undef $vm;
                &Log("[D] [$vmname] at localhost disconnected.¥n");
                return 0;
        }
        else{
                &Log("[D] [$vmname] at localhost connected.¥n");
        }
        my $last_hb = -1;
        for(my $i = 0; $i < 2; $i++){
                if(&IsPoweredOn($vm)){
                        my $hb = $vm->get_heartbeat();
                        if(!defined $hb) {
                                my ($error_number, $error_string) =
$vm->get_last_error();
                                &Log("[E] Could not got HB count: Error $error_number:
$error_string¥n");
                                undef $vm;
                                &Log("[D] [$vmname] at localhost disconnected.¥n");
                                return 0;
                        }
                        else{
                                &Log("[D] Got HB count $hb.¥n");
                        }

                        if ($hb == $last_hb) {
                                &Log("[E] [$vmname] is stalled.¥n");
                                undef $vm;
                                &Log("[D] [$vmname] at localhost disconnected.¥n");
                                return 0;
                        }
                        else{
                                $last_hb = $hb;
                        }
                }
                else{
                        undef $vm;
                        &Log("[D] [$vmname] at localhost disconnected.¥n");
                        return 0;
                }
                sleep $interval;
        }

        undef $vm;
        &Log("[D] [$vmname] at localhost disconnected.¥n");
        return 1;
}
#--------------------------------------------------------------------------------
sub vm_constant {
        my $constant_str = shift;
        return VMware::VmPerl::constant($constant_str, 0);
}
#--------------------------------------------------------------------------------
```

```
sub Log{
        my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime(time);
        $year += 1900;
        $mon += 1;
        my $date = sprintf "%d/%02d/%02d %02d:%02d:%02d", $year, $mon, $mday, $hour, $min,
$sec;
        print "$date $_[0]";
        return 0;
}
#-------------------------------------------------------------------------------
sub ConnectVm{
        my $connect_params = VMware::VmPerl::ConnectParams::new(undef, undef, undef, undef);
        if (! $_[0]->connect($connect_params, $cfg_path)) {
                return 0;
        }
        else{
                return 1;
        }
}
```

vmpreaction.sh

```
#! /bin/sh
#*********************************************
#*              preaction.sh                 *
#*********************************************

ulimit -s unlimited

# ESX が起動する VM のリソース名を記述
CLPRSC="vmpower"
# カンマ区切りで各 ESX の IP を記述
CLPIP="10.0.0.1,10.0.0.2"

/opt/nec/clusterpro/bin/clptrnreq -t GRP_FAILOVER -r $CLPRSC -h $CLPIP

exit 0
```