

# Implementing Fault-tolerance through Dell OpenManage and the VMware Software Development Kit

---

Enterprise Product Group (EPG)

**Dell White Paper**

By Dave Jaffe and Todd Muirhead

**Dell** | Enterprise Systems

March 2005

Updated September 2005

# Contents

---

Executive Summary.....	3
Introduction.....	4
VMware Scripting .....	6
Dell OpenManage and VMware ESX Server .....	9
Integrating OpenManage with VMware ESX Server .....	11
Conclusions .....	13
Acknowledgements.....	14
Appendix A. vmall.cs .....	15
Appendix B. remote2.vbs.....	21
Figure 1 Using OpenManage and VMware Scripting for Fault Tolerance.....	11

## Executive Summary

A new scripting facility from VMware, the Virtual Infrastructure SDK Package, enables programs written in C#, Java and other popular languages to access information about VMware® ESX Server hosts managed by VMware VirtualCenter and to manipulate these servers and the virtual machines that run on these servers. In this paper, this facility is combined with Dell's OpenManage to demonstrate a powerful fault tolerance capability. A script was developed to call the VMware VMotion™ feature to move all the virtual machines off of a server to other ESX Server hosts, without having to shut down the virtual machines, and with little impact to the end users. When OpenManage IT Assistant, the central OpenManage console, detects a failing ESX Server host, it calls this script to move all the virtual machines off that server. That server can then be taken down, serviced, and brought back online, at which time the virtual machines can be moved back.

---

## Introduction

For server consolidation, fault tolerance and ease of administration, many enterprises have moved all or part of their data center applications onto virtual machines using VMware ESX Server running on multiple Dell servers. The VMware central administration program, VirtualCenter, is used to manage these farms of ESX Server hosts. VirtualCenter lists the physical servers and virtual machines (VMs) in the farm and displays current information regarding performance, tasks, and events happening on the servers in the farm. VirtualCenter also allows the administrator to create, clone, start and stop VMs as well as move them from one ESX Server host to another without having to stop them, using a process called "VMotion". These features are discussed in "The Scalable Enterprise: VMware ESX Server on the Dell PowerEdge 6650", at <http://www.dell.com/downloads/global/power/1q04-jav.pdf>.

By enabling developers to access these VirtualCenter features through a web services interface accessible from most popular programming languages, the VMware Virtual Infrastructure SDK Package extends the functionality of VirtualCenter and enables users to create custom VM applications. For example, a script can be created to clone new VMs from existing "golden" VMs in a repository in response to end-users requests, and to tailor those VMs (hostname, IP address) as desired.

Server administration tools such as Dell's OpenManage™ IT Assistant (ITA) and OpenManage Server Administrator (OMSA) are the second key piece needed to manage large server farms. ITA (the centralized management console) and OMSA (the agent running on each server) are used to detect and report error conditions, to inventory hardware assets, and to update server software and firmware.

A key feature of IT Assistant is the ability to call user scripts in response to certain events such as warnings coming from specific servers. This can be combined with VMware scripting to provide powerful tools that unleash the power of virtualization. This paper describes an example of this, in which an error condition on a server (a temperature, voltage or fan speed that has gone out of specification, for example) causes all the VMs to be moved from the failing server to the other ESX Server hosts in the farm in a load-balanced fashion. When the error is detected by OMSA running on the failing server, OMSA sends a warning to ITA, which in turns runs a script on the VirtualCenter server. This script (a C# program) uses the VMware virtual infrastructure to call the VMware VMotion facility to move the VMs off the failing server.

The VMware scripting facility is discussed in Section 3, followed by details on how to set up OpenManage to work with VMware scripts in Section 4. Finally, the integration of OpenManage with VMware scripting is described in Section 5.

## VMware Scripting

VMware now provides the VMware Virtual Infrastructure SDK Package (<http://www.vmware.com/support/developer/vc-sdk>), which enables developers to access VirtualCenter data and call VirtualCenter commands through a web services interface. Programs communicate with VirtualCenter using the standard SOAP/XML web services protocol to access data (hosts, VMs, performance counters, events, etc.) or issue commands (Clone, Migrate, Power On/Off, etc.). The interface to the VMware VirtualCenter Web Service is defined by the VMware API Web Services Description Language file, *vma.wsdl*. Any programming language that can issue SOAP/XML web service requests can be used for the client programs. The SDK supplies examples in C#, Java, Visual Basic and Perl.

Programming client applications in C# is straightforward, following the examples provided in the SDKs. The details of the web service interface are contained in a C# source file, *vmaService\_proxy.cs*, which needs to be compiled with the client source code. In the sample C# files included in the SDK, and in the program described below, the calls to this proxy are wrapped by client-callable programs inside the client application.

The program used in this demo, *vma* (VMotion All VMs), illustrates the use of the VMware virtual infrastructure to move the VMs on a given ESX Server host sequentially to other hosts in the server farm in a load-balanced fashion. The program uses the VirtualCenter Web Service to list the VMs on the specified host, then to check the CPU utilization on the other ESX Server hosts in a specified migration pool, and finally to issue the VMotion command repeatedly to move the VMs off the specified ESXS host.

As seen in the source file, *vma.cs* (Appendix A), the *vma* program follows the architecture used in the SDK sample files. A class, *VmaClient*, is defined which includes VMware code to create instances of the class, plus code to connect to the Web Service, and code that wrappers *vmaService\_proxy.cs* functions that manipulate the VirtualCenter objects (ResolvePath, GetContents, MigrateVM, etc.). The user-written function *get\_host\_cpu\_time* is a special function that uses the performance objects exposed by VirtualCenter to return the total CPU time (in milliseconds) used by an ESX Server host during the previous minute. To increase the accuracy of this sample a new Performance counter, "Past Hour" needs to be created in the VirtualCenter user interface, with 1 sample per minute for 60 minutes. Also, with the SDK Package for VirtualCenter 1.2 version

(released 12/12/04) it is necessary to make one change to a configuration file to enable the performance counter to work. To the file  
C:\Documents and Settings\All Users\Application Data\VMware\VMware VirtualCenter\VMA\vmaConfig.xml add the following line anywhere between the <subject> and </subject> tags:

```
<periodicPerfRefreshEnable>true</periodicPerfRefreshEnable>
```

Then restart the VMware VirtualCenter Webservice service from the Windows services control panel.

The *vma* action occurs in the user-written function *Main*. There are four steps: 1) connect to the VirtualCenter Web Service, 2) check that the specified source server is part of the ESX Server host migration pool, 3) determine the number of VMs on the source ESX Server host and list the other ESX Server hosts in the migration pool and 4) iterate through the list of VMs and move each VM to the least-loaded ESX Server host in the migration pool at the time.

To connect with the VirtualCenter Web Service the user's credentials (username and password) are needed along with the URL of the web service. The username and password are hard-coded into the program in plain text for demo purposes, and the URL references a non-SSL version of the web page, but in production the username and password would be encrypted and the web service would be accessed via SSL. A new instance of the *VmaClient* object, *vma*, is created, then the *Connect* method of *VmaClient* (which wrappers the *vmaService\_proxy.cs Login* function) is called with the URL, username and password to connect to the VirtualCenter Web Service.

Next the program checks that the source host (the host from which the VMs will move) is a member of the ESX Server host migration pool. For demonstration purposes the set of hosts that are available for VM migrations is hard coded into the program in the array *migration\_pool\_hostnames*. A production version of this script could determine this set at runtime by querying VirtualCenter for a list of hosts meeting certain criteria.

Next the program uses the VirtualCenter Web Service to count the VMs on the source server and list the other ESX Server hosts in the migration pool. The method *ResolvePath("/host")* is used to obtain a handle to the VirtualCenter host list, which is used by *GetContents* to capture this list in a *Container* object. The program then iterates through this list of ESX Server hosts. If a listed host matches the source host, the number of VMs currently on the source host is determined. If a listed host is not the source but is contained in the *migration\_pool\_hostnames* array, the number of CPUs on that host is stored in the *hosts\_to\_migrate\_to\_n\_cpus* array and a new array of hostnames, *hosts\_to\_migrate\_to*, is generated.

Finally the program generates a list of VMs on the source host by drilling down into the VirtualCenter host object, and then steps through this list, using the *MigrateVM* object to move each VM to one of the hosts in the *hosts\_to\_migrate\_to* array. The particular host is chosen by checking the CPU utilization of each host

and choosing the host with the least-loaded CPUs. This is done by using the *get\_host\_cpu\_time* method described above to get the total CPU utilization time of that host for the last minute (in milliseconds), which is then converted to a utilization percent by dividing by the number of milliseconds in a minute (60,000) and the number of CPUs in the host and multiplying by 100. Each call to *MigrateVM* returns a handle to a task that is used to monitor the progress of the VMotion process. The VMotion processes occur sequentially in this program but other methods are certainly possible. The program writes the progress of each migration to the screen (the progress can also be seen by viewing Tasks in VirtualCenter). Finally the program disconnects from the web service and returns.



---

## Dell OpenManage and VMware ESX Server

Dell OpenManage is a set of systems management tools for monitoring, updating, and managing Dell hardware. Specifically, the monitoring of servers is accomplished with the OpenManage IT Assistant (ITA) centralized console and the OpenManage Server Administrator (OMSA) server monitoring and management tool. ITA is typically installed on a single server that will act as the single monitoring console. OMSA is installed on each Dell PowerEdge server and monitors that server. OMSA utilizes SNMP (Simple Network Management Protocol) to send information to ITA about hardware status changes such as a disk failure or an overheated processor. Once the information reaches the ITA console a set of rules can be defined that dictate what actions should be taken. For example, if a disk fails on the database server, ITA can be set up to send an email to the database administrator, and possibly run a script or program as well.

Every Dell™ PowerEdge™ server (except for the PowerEdge SC line) ships with the OpenManage software including OMSA and ITA. There are versions of OMSA for Windows, Linux, and NetWare. For ESX Server, the Linux® version of OMSA is used. OMSA will monitor the hardware of the ESX Server host meaning that the VMs running on ESX Server do not need to have any hardware monitoring capability. To set up OMSA under ESX Server see [http://www.dell.com/downloads/global/solutions/vmware\\_251\\_installing\\_openmanage.pdf](http://www.dell.com/downloads/global/solutions/vmware_251_installing_openmanage.pdf)

The IT Assistant central console can be set up to respond to various types of events coming from specified or all servers. For the demo (which is triggered by intentionally mis-setting a fan speed warning value so that OMSA sends a warning that a fan speed is too slow) a new Event Filter is created which is triggered by Environmental warnings (which includes fan speed) coming from all ESX Server hosts. In the ITA console:

- Configuration->Event Filters->Add
  - Filter Name: ESXDEMO1
  - Severity Configuration: select Warning only
  - Select Event Categories/Types: select all Environmental
  - Select Source Nodes: select ESX1, ESX2 and ESX3
  - OK

The new Event Filter also contains a list of actions that will occur when the event is received. In addition to displaying an Alert message on the ITA console and

writing the event in the Windows Event Log, a new Action is created which calls a Visual Basic Script, remote2.vbs which remotely calls the vsmall program on the VirtualCenter server and passes it the hostname of the ESX Server host sending the warning:

- Event Filters->select ESXDEMO1->Actions
  - New Action->Create from Template->select Application Launch->Create
    - Action Type: 32-bit Application Launch
    - Action Name: VMALL
    - Description: Move all VMs off server
    - Executable Name: cscript
    - Arguments: C:\remote2.vbs \$n
    - Save, Close
  - Make sure that VMALL appears in right column (Assigned Actions) of ESXDEMO1
- Close

The script remote2.vbs (see Appendix B) is used to call the vsmall.exe program on the VirtualCenter server. Note that the Action's executable is actually cscript.exe, the Windows Script Host executable, and that the "\$n" in the Argument list will be replaced at the time the Action is called by the hostname of the server originating the event. The script remote2.vbs passes on this hostname to the vsmall program as its only argument.

## Integrating OpenManage with VMware ESX Server

The entire process of using OpenManage and VMware ESX Server to automatically move virtual machines off a failing server is illustrated in Figure 1.

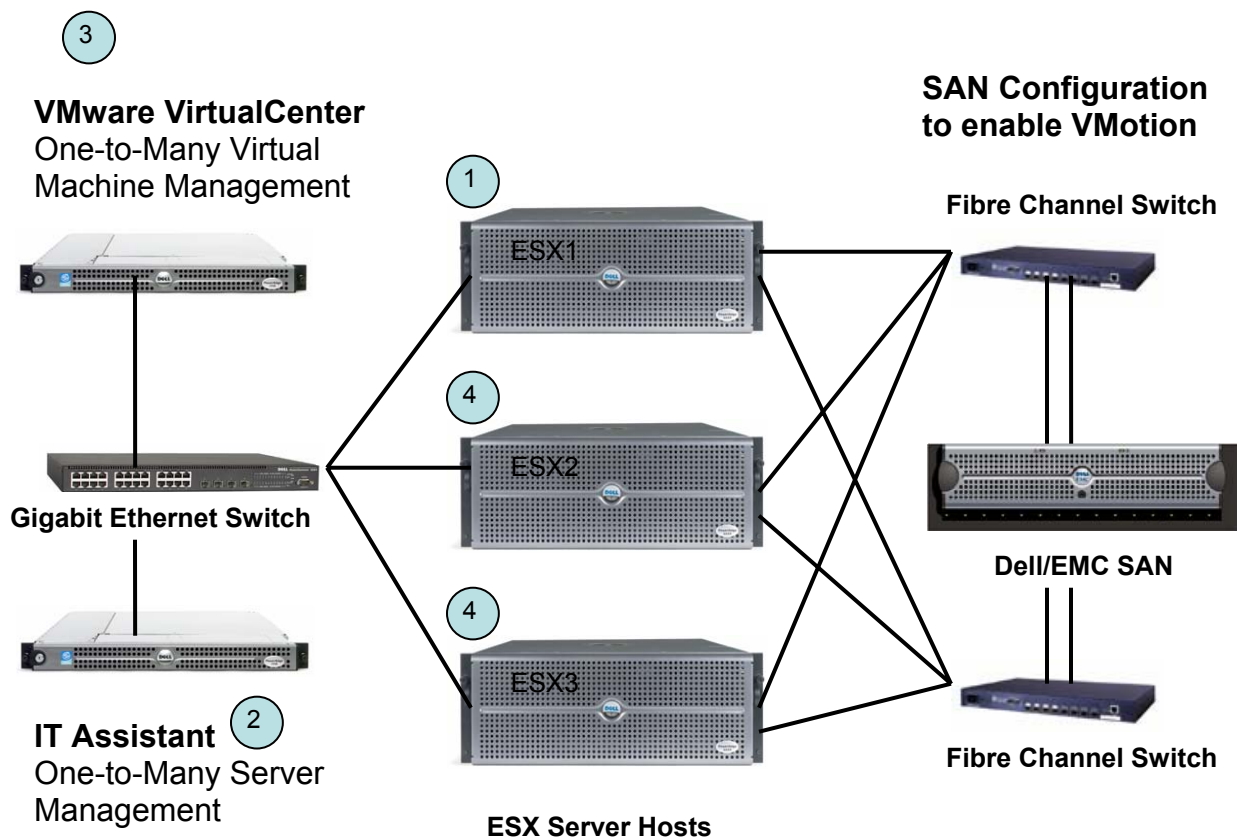


Figure 1 Using OpenManage and VMware Scripting for Fault Tolerance

Step 1: OMSA detects low fan speed on ESX 1 and sends warning via SNMP to IT Assistant

Step 2: Event from ESX1 is detected by IT Assistant

Step 3: IT Assistant fires Event Action that calls local script that remotely calls VMware script on VirtualCenter system to move VMs off of ESX1

Step 4: Script moves VMs to other ESX Server hosts in load-balanced fashion

In this scenario a server farm consisting of three ESX Server hosts is managed by VirtualCenter running on a PowerEdge 1850. A second PowerEdge 1850 runs IT Assistant to manage the ESX Server hosts as well as other servers in the data center (not shown). The two PowerEdge 1850s are connected to the ESX Server hosts and the rest of the data center through a Gigabit Ethernet switch. The three ESX Server hosts are connected to a Storage Area Network (SAN) using redundant host bus adapters in each server, redundant fibre channel switches, and redundant connections to the storage.

In the event of an error or warning on an ESX Server host, OMSA running on that server detects it and sends an event to ITA via SNMP. Using the ESXDEMO1 Event Filter and associated Action described in the previous section, ITA reacts to the warning event by calling a local script, `remote2.vbs`, which calls the VMware script `vmall.exe` on the VirtualCenter system, passing it the hostname of the system sending the warning. `vmall` then sequentially moves the virtual machines from the failing server to the other ESX Server hosts, moving each VM to the least-loaded ESX Server host at the time. When the process is done the failing server can be taken off-line for repair.

---

## Conclusions

Dell OpenManage tools such as IT Assistant and Server Administrator can be combined with the new scripting facility from VMware, the Virtual Infrastructure SDK, to create tools to help manage virtual server infrastructures. In this paper such a combination is described which uses OpenManage to detect a failing server and call a VMware script to move all VMs off that server. The VMs are moved while running with little impact to the end user. The server can then be brought down for repair without the end user even being aware that the server they were running on was starting to fail. Since the Virtual Infrastructure SDK makes all the features of VMware VirtualCenter available, scripts can be written to automate a variety of system management tasks in addition to the one demonstrated here.

---

THIS WHITE PAPER IS FOR INFORMATIONAL PURPOSES ONLY, AND MAY CONTAIN TYPOGRAPHICAL ERRORS AND TECHNICAL INACCURACIES. THE CONTENT IS PROVIDED AS IS, WITHOUT EXPRESS OR IMPLIED WARRANTIES OF ANY KIND.

Dell, PowerEdge, and PowerConnect are trademarks of Dell Inc. Intel and Xeon are registered trademarks of Intel Corp. VMware, the VMware "boxes" logo, ESX Server, GSX Server, VirtualCenter and VMotion are either registered trademarks and/or trademarks of VMware, Inc. in the United States and/or other jurisdictions. Other trademarks and trade names may be used in this document to refer to either the entities claiming the marks and names or their products. Dell disclaims proprietary interest in the marks and names of others.

Copyright 2005 Dell Inc. All rights reserved. Reproduction in any manner whatsoever without the express written permission of Dell Inc. is strictly forbidden. For more information, contact Dell.

Information in this document is subject to change without notice.

## Acknowledgements

The authors would like to thank Sudhir Shetty for the WMI scripts, Mark Clifton and Felipe Payet for valuable discussions regarding the integration of OpenManage and VMware ESX Server, and Shivraj Ahluwalia at VMware for support on the VMware scripting API.

## Appendix A. vmall.cs

```
using System;
using System.Threading;
using System.Web.Services.Protocols;
using VMware.vma;

namespace vmall
{
    /// <summary>
    /// vmall - VMware VMotion Program - move all VMs from specified server to
    /// other ESX servers based on server load
    /// Uses VirtualCenter SOAP API
    /// Code is from VMware SDK except where noted - requires vmaService_proxy.cs
    /// To compile:
    /// C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin\vcvars32
    /// (adjust if VS.NET was installed to a different directory)
    /// csc vmaService_proxy.cs vmall.cs
    /// (in directory where vmall.cs and vmaService_proxy.cs are located)
    /// </summary>

    public class GlobalConstants
    {
        public const int MAX_HOSTS = 10;
    }

    public class VmaClient
    {
        public enum VmaClientState
        {
            Connected,
            Disconnected,
        }

        protected vmaService vma_;
        protected VmaClientState state_;
        public event VmaClientEventHandler AfterDisconnect;

        public VmaClient()
        {
            state_ = VmaClientState.Disconnected;

            // Manage bad certificates our way:
            //System.Net.ServicePointManager.CertificatePolicy = new CertPolicy();
        }

        /// <summary>
        /// Creates an instance of the VMA proxy and establishes a connection
        /// </summary>
        /// <param name="username"></param>
        /// <param name="password"></param>
        public void Connect(string url, string username, string password)
        {
            if (vma_ != null)
            {
                Disconnect();
            }

            vma_ = new vmaService();
            vma_.Url = url;
            vma_.CookieContainer = new System.Net.CookieContainer();
            vma_.Login(username, password);
            state_ = VmaClientState.Connected;
        }

        public vmaService Vma
        {
            get
            {
                return vma_;
            }
        }
    }
}
```

```

    }

public VmaClientState State
{
    get
    {
        return state_;
    }
}

/// <summary>
/// Disconnects the VmaClient
/// </summary>
public void Disconnect()
{
    state_ = VmaClientState.Disconnected;
    if (vma_ != null)
    {
        vma_.Dispose();
        vma_ = null;
        if (AfterDisconnect != null)
        {
            AfterDisconnect(this, new VmaClientEventArgs());
        }
    }
}

public string ResolvePath(string path) // Added by DJ
{
    return vma_.ResolvePath(path);
}

public ViewContents GetContents(string handle)
{
    return vma_.GetContents(handle);
}

public ViewContents MigrateVM(string vm, string host, Level priority, string
dataLocator) // Added by DJ
{
    return vma_.MigrateVM(vm, host, priority, dataLocator);
}

public class VmaClientEventArgs : System.EventArgs
{
}

public delegate void VmaClientEventHandler(object sender, VmaClientEventArgs e);

// vmall - VMware VMotion - move all VMs from specified host to other hosts based
// on host loading
// Author: Dave Jaffe
// Last modified: 9/28/05 for SDK Package for VirtualCenter 1.2
// Copyright Dell 2005
static void Main(string[] args)
{
    int i, j, n_hosts_to_migrate_to=0, source_ind=0, lowest_tot_cpu_ind=0;
    double tot_cpu_util, lowest_tot_cpu_util;
    ViewContents vc;
    Host host;
    VirtualMachine vm;
    Task task;
    Container c;
    Item[] items;
    Boolean in_migration_pool, source_found;
    string source_hostname;
    string[] hostnames = new string[GlobalConstants.MAX_HOSTS];
    string[] migration_pool_hostnames = {"esxdemo1", "esxdemo2", "esxdemo3"};
    string[] hosts_to_migrate_to = new string[GlobalConstants.MAX_HOSTS];
    int[] hosts_to_migrate_to_n_cpus = new int[GlobalConstants.MAX_HOSTS];
    string target_hostname = null, taskhandle = null;

```

```

string vmname = null, target_hostkey = null;
TimeZone ctz = System.TimeZone.CurrentTimeZone;

if (args.Length != 1)
{
    Console.WriteLine("Usage: vmall host_to_move_from");
    return;
}
source_hostname = args[0];

// Verify source host is in migration pool
in_migration_pool = false;
for (i=0; i<migration_pool_hostnames.Length; i++)
{
    if (source_hostname == migration_pool_hostnames[i]) in_migration_pool = true;
}
if (in_migration_pool)
    Console.WriteLine("specified source host: {0}", source_hostname);
else
{
    Console.WriteLine("specified source host {0} is not in migration pool",
        source_hostname);
    return;
}

string url = "http://localhost:8088";
string username = "vcadmin";
string password = "password";

VmaClient vma = new VmaClient();

Console.WriteLine("Logging in...");
vma.Connect(url, username, password);

// Look up source host, count VMs on it, determine other hosts in migration pool
string hosthandle = vma.ResolvePath("/host");
//Console.WriteLine("hosthandle = " + hosthandle);
vc = vma.GetContents(hosthandle);
c = (Container) vc.body;
items = c.item;
source_found = false;
for (i=0; i<items.Length; i++)
{
    if (items[i].name == source_hostname)
    {
        source_found = true;
        vc = vma.GetContents(items[i].key);
        host = (Host) vc.body;
        if (host.vm == null)
        {
            Console.WriteLine("Host {0} has no VMs to move, exiting",source_hostname);
            return;
        }
    }
    else
    {
        source_ind = i;
        Console.WriteLine("Host {0} has {1} VMs to move", source_hostname,
            host.vm.Length);
    }
}
else
{
    for (j=0; j<migration_pool_hostnames.Length; j++)
    {
        if (items[i].name == migration_pool_hostnames[j])
        {
            vc = vma.GetContents(items[i].key);
            host = (Host) vc.body;
            hosts_to_migrate_to_n_cpus[n_hosts_to_migrate_to] =
                host.hardware.cpu.Length;
            hosts_to_migrate_to[n_hosts_to_migrate_to++] =

```

```

        migration_pool_hostnames[j];
    }
}
}

if (!source_found)
{
    Console.WriteLine("Source host {0} not found, exiting",
        source_hostname);
    return;
}

if (n_hosts_to_migrate_to == 0)
{
    Console.WriteLine("No hosts to migrate to, exiting");
    return;
}
else
{
    Console.WriteLine("Hosts to migrate to:");
    for (i=0; i<n_hosts_to_migrate_to; i++)
        Console.WriteLine("  host {0}: {1} w/ {2} CPUs",
            i, hosts_to_migrate_to[i], hosts_to_migrate_to_n_cpus[i]);
}

// Sequentially move VMs from source host to least-loaded other host
vc = vma.GetContents(items[source_ind].key);
host = (Host) vc.body;
Console.WriteLine("Moving {0} VMs from source host {1}:", host.vm.Length,
    host.info.hostname);
for (i=0; i<host.vm.Length; i++)
{
    vc = vma.GetContents(host.vm[i]);
    vm = (VirtualMachine) vc.body;
    vmname = vm.info.name;
    Console.WriteLine("  ({0}) {1}", i, vmname);
    // Find least loaded host to migrate to
    lowest_tot_cpu_util = 100.0;
    for (j=0; j<n_hosts_to_migrate_to; j++)
    {
        // get host_cpu_time() returns total CPU time on host in ms for a 60 sec interval
        // 60,000 * n_cpus is total CPU time in ms available in 60 sec for n_cpus
        // thus 100*get_host_cpu_time()/(60000*n_cpus) is total CPU utilization % on host
        tot_cpu_util =
            100.0*((double) vma.get_host_cpu_time(hosts_to_migrate_to[j]))/
            (60000*hosts_to_migrate_to_n_cpus[j]);
        Console.WriteLine("    host {0}: {1} w/ {2} CPUs  Total CPU Util%={3}",
            j, hosts_to_migrate_to[j], hosts_to_migrate_to_n_cpus[j], tot_cpu_util);
        if (tot_cpu_util < lowest_tot_cpu_util)
        {
            lowest_tot_cpu_util = tot_cpu_util;
            lowest_tot_cpu_ind = j;
        }
        //Console.WriteLine("lowest: {0}  total: {1}", lowest_tot_cpu_util,
        //    tot_cpu_util);
    }
    target_hostname = hosts_to_migrate_to[lowest_tot_cpu_ind];
    target_hostkey = vma.ResolvePath("/host/" + target_hostname);
    Console.WriteLine(
        "Migrating VM {0} from host {1} to host {2} with high priority",
        vmname, source_hostname, target_hostname);
    vc = null;

    vc = vma.MigrateVM(host.vm[i], target_hostkey, Level.high, null);
    if (vc == null)
    {
        Console.WriteLine("vc returned from MigrateVM is null");
    }
    else
    {

```

```

        if (vc.body == null)
        {
            Console.WriteLine("vc.body is null");
        }
        else
        {
            taskhandle = vc.handle;
            //Console.WriteLine("taskhandle = " + taskhandle);
            do
            {
                Thread.Sleep(2000);
                vc = vma.GetContents(taskhandle);
                task = (Task) vc.body;

                // Need to explicitly account for Daylight Savings Time due to .NET
                // problem (not VMware bug!)
                Console.WriteLine(
                    "task: {0} % completed: {1} Status: {2} Started: {3} Now: {4}",
                    task.operationName, task.percentCompleted, task.currentState,
                    task.queueTime.AddHours(ctz.IsDaylightSavingTime(task.queueTime) ?
                        -1:0).ToLongTimeString(),
                    System.DateTime.Now.ToLongTimeString());
            } while (task.currentState.ToString() != "completed");
        }
    }

    Console.WriteLine("Logging out...");
    vma.Disconnect();
}

public int get_host_cpu_time(string hostname)
{
    // Retrieve CPU time used (in ms) of specified host in last sample
    // Need to create new performance interval:
    // In Virtual Center: File->VMware VirtualCenter Settings->Performance
    // Name: Past Hour Minutes per sample: 1 Number of samples: 60
    // (one sample per minute is apparently fastest sample rate)
    // Path name on next line refers to numbers of seconds per sample
    int i, j, used_total = 0;
    ViewContents vc;
    Host host;
    Container c;
    Item[] items;
    PerfCollection stats;
    string used_string = null;

    string perfhandle = vma_.ResolvePath("/perf/0000000060");
    vc = vma_.GetContents(perfhandle);
    c = (Container) vc.body;
    items = c.item;
    for (i=0; i<items.Length; i++)
    {
        //Console.WriteLine("i= {0} items[i].type = {1}",i,items[i].type.ToString());
        if (items[i].type.ToString() == "PerfCollection")
        {
            vc = vma_.GetContents(items[i].key);
            stats = (PerfCollection) vc.body;
            for (j=0; j<stats.source.Length; j++)
            {
                vc = vma_.GetContents(stats.source[j].key);
                if (vc.body.GetType().ToString() == "VMware.vma.Host")
                {
                    host = (Host) vc.body;
                    if (host.info.hostname == hostname)
                    {
                        PerfSample[] sample = stats.source[j].sample;
                        PerfStat[] sample_stats = sample[0].stat;
                        for (int k=0; k<sample_stats.Length; k++)
                        {

```

```

        PerfStat stat = sample_stats[k];
        PerfStatType statType = stat.type;
        if (statType == PerfStatType.cpu)
        {
            CPUPerf data = (CPUPerf) stat.data;
            used_string = data.used.ToString();
            used_total += Convert.ToInt32(used_string);
            //Console.WriteLine("hostname = {0} j= {1} k= {2} used= {3} total= {4}",
            // hostname, j, k, used_string, used_total);
        } // End if (statType == PerfStatType.cpu)
    } // End loop on k
} // End if (host.info.hostname == hostname)
} // End if (vc.body.GetType().ToString() == "VMware.vma.Host")
} // End loop on j
} // End if (items[i].type.ToString() == "PerfCollection")
} // End loop on i
return (used_total);
} // End get_host_cpu_time
} // End class VmaClient
} // End namespace vma11

```

## Appendix B. remote2.vbs

```
rem remote2.vbs Based on remote.vbs from Sudhir Shetty, Dell

strTarget    = WScript.Arguments.Item(0)
strCommand   = "C:\vmware\vmall.exe " & strTarget
strComputer  = "VirtualCenterServer"

Set objWMIService = GetObject("winmgmts:" _
    & "{impersonationLevel=impersonate}!\\" & strComputer & _
    "\root\cimv2:Win32_Process")

errReturn = objWMIService.Create(strCommand,null,null,intProcessID)
if errReturn = 0 Then
    Wscript.Echo strCommand & " was started with a process ID of " _
    & intProcessID & "."
Else
    Wscript.Echo strCommand & " could not be started due to error " _
    & errReturn & "."
End If
```