VMware VMotion Performance on the Dell PowerEdge 1855 Blade Server

Dell White Paper

By Dave Jaffe, Todd Muirhead and Balasubramanian Chandrasekaran

Dell | Enterprise Systems

December 2005

Contents

Section 1

Executive Summary

To demonstrate the power of running VMware's ESX Server on Dell PowerEdge 1855 blade servers, including use of the VMotion live virtual machine migration facility, a test was run to quantify the impact of sharing the production and VMotion networks on virtual machine performance. It was found that even very frequent VMotion usage, far beyond what would be expected in typical VMotion scenarios (such as load balancing, fault tolerance or routine maintenance), has negligible impact on virtual machine application performance.

Section 2

Introduction

The Dell PowerEdge 1855 blade server, provided in a chassis supporting up to 10 blades in a 7 rack unit (12.25") chassis, is a very dense, high-performance, and cost effective platform on which to build "scaled-out" server farms. VMware's ESX Server software, which enables multiple virtual machines (VMs) to be installed on a physical server, is ideally suited for such a server farm. With its VirtualCenter management software to manage the farm of ESX Server hosts, and VMotion to move live VMs from one ESX Server host to another, VMware on PowerEdge 1855 blades provides an excellent platform for hosting server applications.

The VMotion migration of running VMs from one ESX Server host to another is especially useful in enterprise data centers. The scenarios where VMotion is utilized include dynamic load balancing of VMs across an ESX Server farm; graceful failover of VMs off a failing ESX Server host; and movement of VMs off an ESX Server host to bring that host down for routine maintenance. In all cases it is desirable that any performance impact due to VMotion be negligible as seen by the end user running applications on the VMs.

Each PowerEdge 1855 blade server ships with two embedded Gigabit Ethernet (GbE) controllers, which are wired though the blade chassis to two GbE switches or Pass-Through modules in the back of the chassis. Additionally, each blade server supports an optional dual port daughter card which can be used for more Ethernet ports, Fibre Channel, or Infiniband input/output (I/O) interfaces. To use VMotion on VMware, it is necessary to populate the daughter card slot with a Fibre Channel (FC) daughter card to connect the blade to an external Dell/EMC Storage Area Network (SAN). Thus each PowerEdge 1855 blade server is limited to 2 GbE network connections in a VMware with VMotion setup.

The purpose of this study is to quantify the impact on typical VMware operations of having two network connections available for each ESX Server host running on a set of PowerEdge 1855 blades. The white paper, "Deployment of VMware ESX 2.5.1 Server Software on Dell PowerEdge Blade Servers" (http://www.dell.com/downloads/global/solutions/vmware_deployment_blades.pdf), describes in detail the many configuration options available for ESX Server on the PowerEdge 1855 with two network connections. In this study we chose the simplest method, where the ESX Service Console uses one NIC and the second NIC is used for both the network traffic of the VMs running on the blades as well as that used for the VMotion of VMs from one blade to another. This results in network traffic from the VMs and the network traffic for VMotion to compete for bandwidth on the same NIC.

In this test a constant database workload ran against 20 VMs on two PowerEdge 1855 blades while the rate at which the VMs were moved back and forth between the two blades was steadily increased. The test shows that the moves had negligible effect on performance of the VMs, even as VM migrations reached the level of every one of the 20 VMs moving once during a 10 minute period.

Section 3

The Dell Blade Server Solution

The Dell PowerEdge 1855 blade server (Figure 1) provides a rack dense server solution based on industry standard Intel Xeon processors. The PowerEdge 1855 blade server resides in the Dell Modular Server Enclosure that takes up 7 rack units (12.25") in a standard server rack and holds up to 10 blade servers. The chassis provides power, cooling, network connectivity and, optionally, Fibre Channel or InfiniBand® connectivity. This allows for a greatly reduced amount of required cabling and a much simpler deployment of new blades into an existing chassis.

Figure 1: Dell Modular Server Enclosure

Each individual PowerEdge 1855 blade server supports up to two Intel Xeon processors and up to 12 GB of DDR-2 memory. Processors currently available for the PowerEdge 1855 range from 2.8 GHz to 3.8 GHz single core processors (Low Voltage single core processors are also available at 3Ghz which can help to dramatically lower overall system power consumption and thermal output) and there is a 2.8 GHz dual core processor option allowing for an individual blade to have up to 4 processing cores. The PowerEdge 1855 also has two standard Gigabit Ethernet ports and allows for an optional daughter card that can provide two more gigabit Ethernet ports, two Fibre Channel ports, OR two InfiniBand ports. To provide the connectivity in the chassis for the daughtercard-provided connections, corresponding Ethernet, Fibre Channel, or InfiniBand pass-thru modules can be installed or Ethernet or Fibre Channel switch modules can be installed.

For this test the chassis was equipped with two Dell PowerEdge 1855 blade servers, each with VMware ESX Server 2.5.1. The Dell PowerEdge 1855s were configured with two 3.2 GHz Intel Xeon processors with 2 MB L2 cache, 8GB of DDR-2 memory, and two internal SCSI U320 73GB 10K rpm disks. The detailed system configuration of each blade is shown in

Table 1.

Parameter	Description
Operating System	ESX 2.5.1
CPU	2x 3.2 GHz/ Xeon with 2 MB L2 Cache
Memory	8 GB (4 x 2 GB DDR-2 DIMMs)
Internal Disks	2x 73 GB 10K RPM Ultra320 SCSI
NICs	2x 10/100/1000[1] Mb/s (Internal)
Disk Controller	PERC 4im
Fiber Channel Daughter Card	QLA 2342

Table 1: Dell PowerEdge 1855 Blade Features Used in Test

To provide shared storage needed for the live VMotion events the two PowerEdge 1855 server blades were connected to a Storage Area Network (SAN) based on a Dell/EMC CX700 storage controller with redundant service processors (SPA and SPB), connected through redundant Fibre Channel switches. Each blade was configured with the Dell QLogic 2342 daughter card. The chassis was loaded with the pass-thru Fibre Channel modules allowing for the Fibre Channel connections to be made into the external Fibre Channel switches.

Two 5 disk RAID 5 (4+1) LUNs were created on the CX700 to be used for the VMs. The VMs used were evenly spread across the two LUNs with ten on each LUN. Both LUNs and the two blades were assigned to the same storage group allowing for the storage to be shared by both the blades. The storage components used for both servers in the test are shown in

Table 2.

Component	Description
Controller	1 Dell/EMC CX700
Disk Enclosures	1 Dell/EMC DAE2
Disks	10 x 73 GB/ 10K RPM
LUNs	2 5-Disk RAID 5, 1 HotSpare Disk
Software	Navisphere® Manager, Access Logix™

Table 2: Dell/EMC Storage Configuration Used in Test

The entire network setup, including the blade chassis, the Fibre Channel SAN, Virtual Center running on a central server, and a management console, is shown in Figure 2.

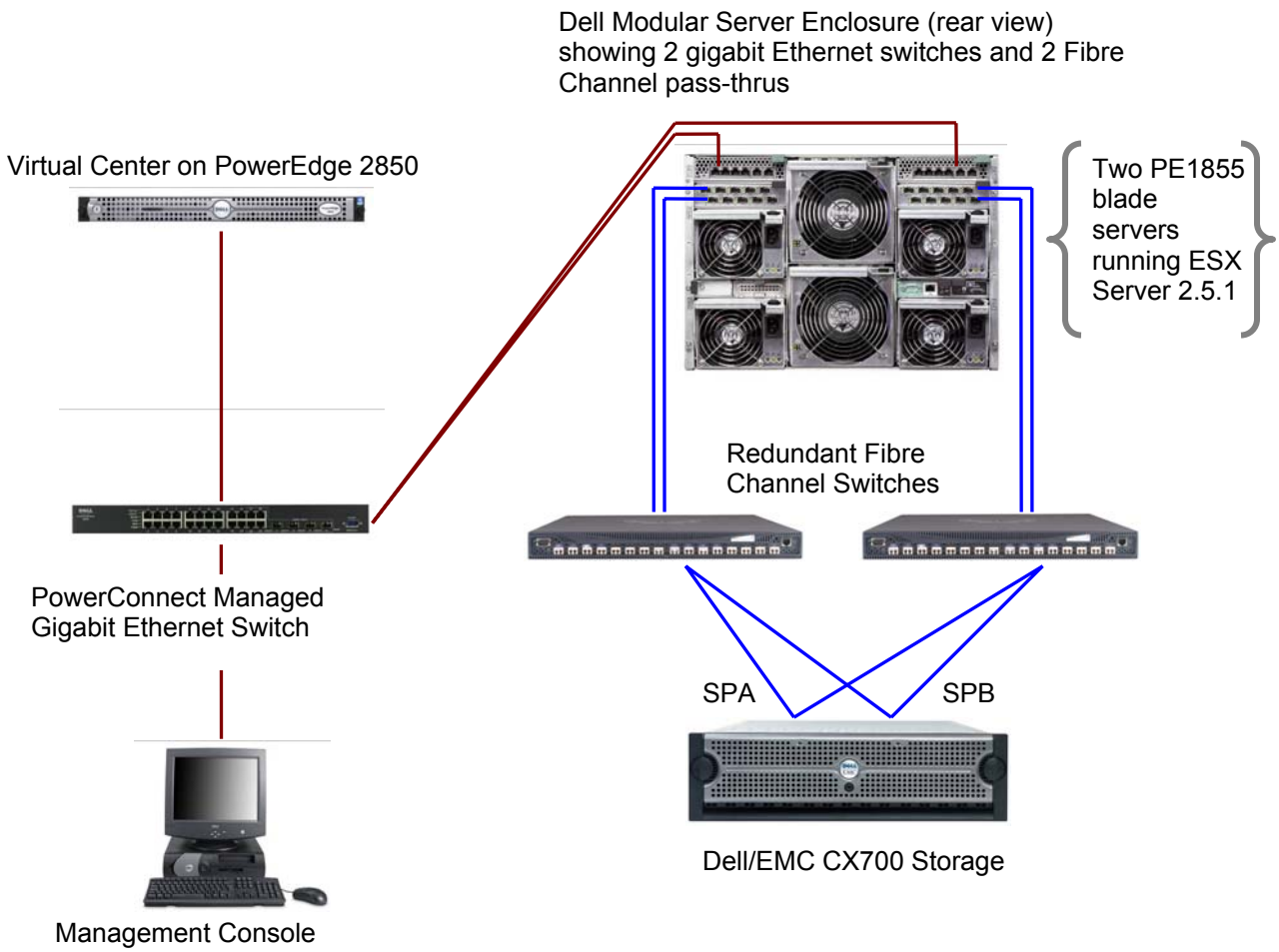

Figure 2: Network Setup Used in Test

Section 4

Optimizing ESX Server on the Dell PowerEdge 1855

To enable VMotion, the virtual machine's virtual disk must reside in storage that is accessible from all the ESX Server hosts which will potentially host that VM. To connect to a Dell/EMC Fibre Channel storage area network for this purpose, each PowerEdge 1855 blade server must be configured with a dual port Fibre Channel daughter card, in addition to the two embedded gigabit Ethernet NICs (NIC0 and NIC1). In ESX Server the NICs have to be dedicated to the Service Console, dedicated to the Virtual Machines or shared between the Service Console and Virtual Machines. In a default installation, which is the configuration used in this test, NIC0 is dedicated to the Service Console and NIC1 is dedicated to the VMkernel and is used by VMs and for VMotion.

VMotion is performed by copying the memory state of the virtual machine from the source physical server to the destination server, through the network fabric. VMotion is a network I/O intensive operation that involves bursty network traffic (in the order of several megabytes to a few gigabytes) for a short duration of time on NIC1. The actual throughput would depend on the size of the allocated memory for the virtual machine. In the following section, we see how the VMotion traffic affects the network traffic of the other VMs.

Setup parameters for ESX 2.5.1 on the two blades are shown in Table 3.

Parameter	Description
Ethernet controller 0	Dedicated to Service Console
Ethernet controller 1	Dedicated to Virtual Machines
SCSI Storage Controller	Dedicated to Virtual Machines
Fiber Storage Controller	Dedicated to Virtual Machines
Virtual Switch	Adapter0 Network

Table 3: ESX Parameters

Section 5

The Test

The test used a SQL Server 2000 implementation of Dell's online DVD store database test application, DS2. On each virtual machine a 1GB database, representing an online DVD store with 100,000 DVD titles, was built. The twenty databases were driven by separate instances of a C# program simulating users logging in to the online store, browsing for DVDs by title, author or category, and finally submitting an order. The driver program measures the number of orders per minute that the database can handle as well as the total response time as seen by the simulated end users.

The complete DVD Store application code is freely available for public use under the GPL license at http://www.linux.dell.com/dvdstore.

Each of the twenty VMs was cloned from the same golden master VM using the VMware Virtual Infrastructure SDK cloning script shown in Appendix A. The parameters of each VM are shown in Table 4.

Parameter	Description
Memory	512 MB
Hard Disk	10 GB
NIC1	Vmxnet
Virtual CPUs	1
Operating System	Microsoft 2003 Server Enterprise Edition

Table 4: VM Parameters

There were two 100 minute parts to the test. In the first run the database workload was driven against all 20 VMs, with 10 VMs on each ESX Server, and no VMotion events occurring. Then the same workload was applied for the same length of time, but with VMs being moved from one blade to the other in increasing frequency. In the first 10 minutes 2 VMotion events occur, with one VM going from blade1 to blade2, and a different VM being moved from blade 2 to blade 1. In the second 10 minutes, 2 VMotion events occur in each direction, and so on until the final 10 minutes in which 10 moves occur in each direction. In other words at the end of the test, each VM is moving on average once every 10 minutes.

The two runs – with and without moves occurring – were compared to measure the effect of the moves on VM performance. The results are shown in the next section.

Section 6

Results

As described in the previous section, the performance of 20 VMs running on ESX Server on two PowerEdge 18555 server blades was measured with and without VMotion events occurring. In the first test (blue line in Figure 3) the SQL Server 2000 database on each of the 20 VMs was driven with about 50 orders per minute for a total average of 999.3 orders per minute (opm). This workload was selected to drive each ESX Server host to about 55-60% CPU utilization, a loading typical of enterprise virtualization hosts. Then the same workload was applied

Time Interval	Number of VMotion events	Average Time of VMotion (sec)	Average Orders Per Minute in Interval
First 10 Minutes	2	29.8	967
Second 10 Minutes	4	30.1	968
Third 10 Minutes	6	28.8	978
Fourth10 Minutes	8	29.3	980
Fifth 10 Minutes	10	29.3	982
Sixth 10 Minutes	12	28.7	986
Seventh10 Minutes	14	29.7	986
Eighth 10 Minutes	16	29.7	985
Ninth 10 Minutes	18	29.6	985
Tenth 10 Minutes	20	29.3	985

Table 5: Results

while VMs were moved randomly between the two ESX Server hosts, with the number of moves increasing as shown in

Table 5. To keep a rough balance of work on each server, each move from blade1 to blade2 was followed by a move from blade2 to blade1.

The results of the second test are shown as the pink line in Figure 3. After a very small initial deficit (967 opm) the order rate rose to very close to the order rate without VMotion events (985 opm) and then held that level as the number of VMotion events was increased to 20 VMotion events in a 10 minute period. Additionally, the time taken by the VMotion events, shown by the yellow diamonds, did not increase as the frequency of VMotion events increased. Figure 4 shows that CPU utilization on the ESX Server hosts increased slightly as the VMotion load was ramped up, to about 65%. The VMotion events are shown as green triangles along the time axis of that graph.

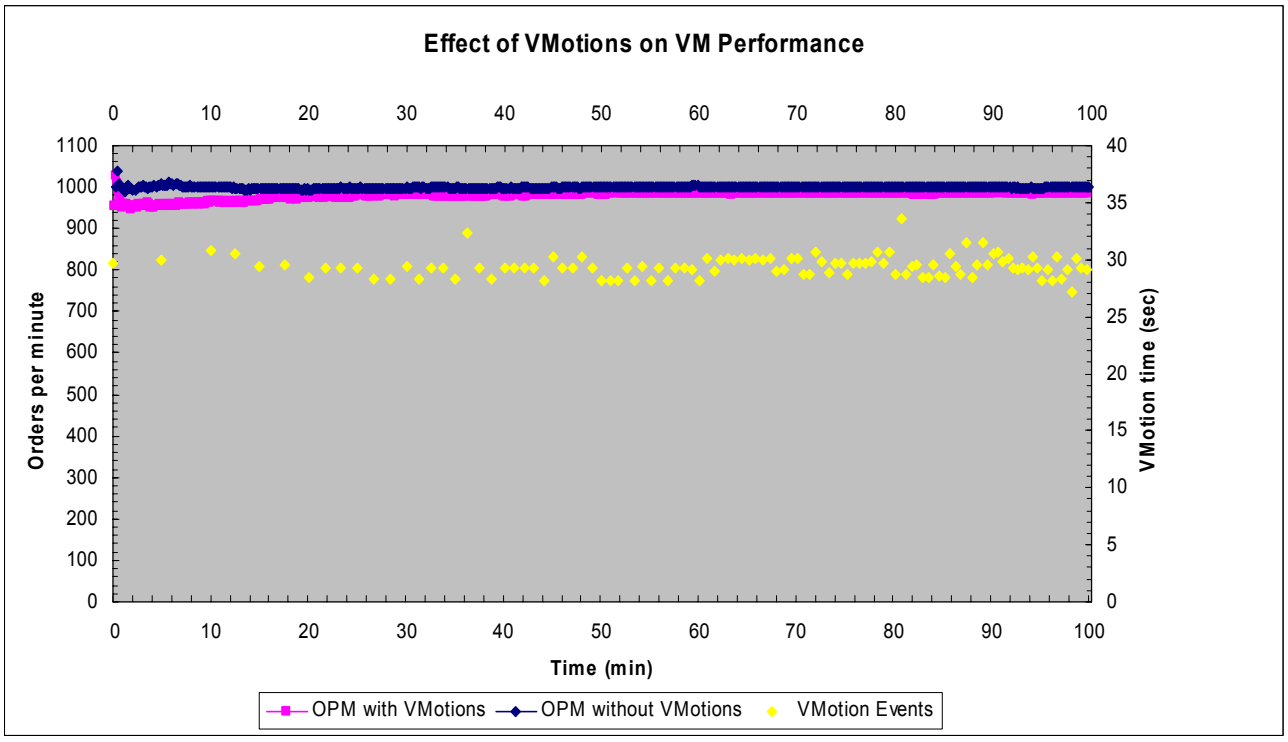

Figure 3: Effect of VMotions on VM Performance

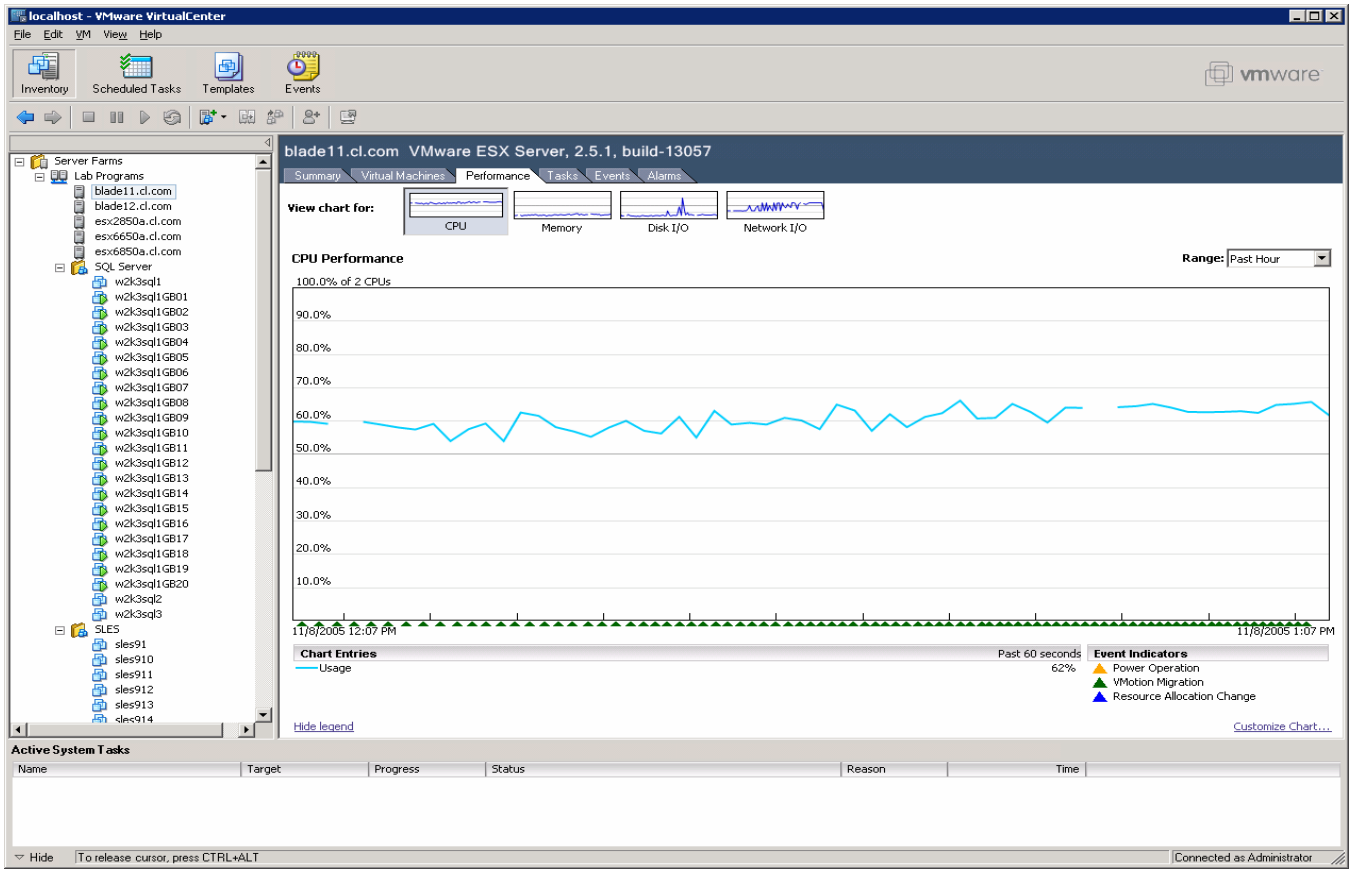

Figure 4: ESX Performance on One Blade During Test

Section 7

Conclusions

The VMotion feature of VMware's ESX Server, which enables live migration of running VMs from one ESX Server to another, can be used to implement high availability or load balancing, or to facilitate routine maintenance. When the server farm consists of Dell PowerEdge 1855 blade servers the VMotion capability may still be used effectively even though the same Ethernet controller on each blade has to support the network traffic to the VMs residing on that blade as well as the VMotion traffic. As shown by the test documented in this paper, even a high rate of VMotions (each of 20 VMs moved within a 10 minute period) did not substantially reduce application performance on the VMs.

[1] This term does not connote an actual operating speed of 1 Gb/sec. For high speed transmission, connection to a Gigabit Ethernet server and network infrastructure is required.

Appendix A. vmclone1.cs

```
using System;
using System.IO;
using System.Threading;
using System.Web.Services.Protocols;
using System.Xml;
using VMware.vma;
using VMware.sysimage;

namespace vmclone1
  {
  /// <summary>
  /// vmclone1 - create and customize clones of specified virtual machine
  /// Uses VMware Virtual Infrastructure SDK
  /// Code is from VMware SDK except where noted - requires vmaService_proxy.cs
  /// To compile: csc vmaService_proxy.cs vmclone1.cs
  /// Syntax: vmclone1 <source vm hostname> <clone config file>
  /// Format of clone config file: each line contains 1 clone's hostname, IP address,
  ///   datastore separated by one or more spaces
  /// example:
  ///   w2k31  192.168.1.11  [SAN1]
  ///   w2k32  192.168.1.12  [SAN2]
  /// </summary>

  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 ViewInfo GetInfo(string handle) // Added by DJ
  {
  return vma_.GetInfo(handle);
  }

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

public ViewContents CloneVM(string srcHandle, string parentHandle, string destHostHandle,
  string name, string datastore, object customization, bool autopoweron)
  {
  return vma_.CloneVM(srcHandle, parentHandle, destHostHandle, name, datastore,
    customization, autopoweron);
  }

public class VmaClientEventArgs : System.EventArgs
  {
  }

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



// vmclone1 - create and customize clones of specified virtual machine
// Author: Dave Jaffe
// Last modified: 9/1/05
// Copyright Dell 2005
// Syntax: vmclone1 <source vm hostname> <clone config file>
// Format of clone config file: each line contains 1 clone's hostname, IP address,
```

```
//   datastore separated by one or more spaces
// example:
//   w2k31  192.168.1.11  [SAN1]
//   w2k32  192.168.1.12  [SAN2]

static void Main(string[] args)
  {
  int i, j, line_no, i_clone, n_clones;
  ViewContents VC;
  ViewInfo VI;
  Host host;
  VirtualMachine vm;
  Task task;
  Container c;
  Item[] items;
  string[] hostnames = new string[10];
  string srcHandle=null, current_hostname = null, parentHandle = null, destHostHandle = null;
  DateTime task_qt = System.DateTime.Now;
  TimeZone ctz = System.TimeZone.CurrentTimeZone;

  string clone_line;
  int MAX_CLONES = 100;
  string[] destVMName = new string[MAX_CLONES];
  string[] destIPAddress = new string[MAX_CLONES];
  string[] datastore = new string[MAX_CLONES];

  // Check input
  if (args.Length < 2)
    {
    Console.WriteLine("Syntax: vmclone1 <source vm hostname> <clone config file>");
    return;
    }

  // Parse input file
  string srcVMName = args[0];
  string config_file_name = args[1];
  if (File.Exists(config_file_name))
    {
    StreamReader sr = new StreamReader(config_file_name);
    line_no = 0;
    while ((clone_line = sr.ReadLine()) != null)
      {
      // Break each line into separate words; handle case where >1 space between words
      string[] p = new string[20];
      p = clone_line.Split(null);

      i=0;
      while (p[i++].Trim() == "") ;
      destVMName[line_no] = p[i-1];

      while (p[i++].Trim() == "") ;
      destIPAddress[line_no] = p[i-1];

      while (p[i++].Trim() == "") ;
      datastore[line_no] = p[i-1];

      ++line_no;
      }
    n_clones = line_no;
    }
  else
    {
    Console.WriteLine("Cannot open config file " + config_file_name);
    return;
    }

  // Establish connection to Virtual Center
  string url = "http://localhost:8088";
  string username = "vcadmin";
  string password = "password";
```

```
VmaClient vma = new VmaClient();

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

// Create and customize clones
for (i_clone=0; i_clone < n_clones; i_clone++)
  {
  Console.WriteLine("\nCloning {0} from {1} with IP= {2} and datastore= {3}",
    destVMName[i_clone], srcVMName, destIPAddress[i_clone], datastore[i_clone]);

  // Find which host srcVMName is on; get srcHandle, destHostHandle, parentHandle
  string hosthandle = vma.ResolvePath("/host");
  VC = vma.GetContents(hosthandle);
  c = (Container) VC.body;
  items = c.item;
  for (i=0; i<items.Length; i++)
    {
    hostnames[i] = items[i].name;
    VC = vma.GetContents(items[i].key);
    host = (Host) VC.body;
    if (host.vm != null)
      {
      //Console.WriteLine("Discovered host: {0}  {1} VMs", hostnames[i], host.vm.Length);
      for (j=0; j<host.vm.Length; j++)
        {
        VC = vma.GetContents(host.vm[j]);
        vm = (VirtualMachine) VC.body;
        if (vm.info.name == srcVMName)
          {
          //Console.WriteLine("Source VM {0} is on host {1}",  srcVMName, hostnames[i]);
          srcHandle = host.vm[j];
          // Make destination host same as source host
          destHostHandle = vma.ResolvePath("/host/" + hostnames[i]);
          VI = vma.GetInfo(srcHandle);
          parentHandle = VI.parent;
          current_hostname = hostnames[i];
          } // End if (vm.info.name == srcVMName)
        } // End for j<host.vm.Length
      } // End if (host.vm != null)
    } // End for (i=0; i<items.Length; i++)

  // Set up customization object
  Autoprep autoprep = new Autoprep();

  Sysprep sysprep = new Sysprep();

  GuiUnattended guiUnattended = new GuiUnattended();
  guiUnattended.TimeZone = "020";  // US Central Time
  guiUnattended.AutoLogon = false;
  guiUnattended.AutoLogonSpecified = true;
  sysprep.GuiUnattended = guiUnattended;

  LicenseFilePrintData licenseFilePrintData = new LicenseFilePrintData();
  licenseFilePrintData.AutoMode = "PerServer";
  licenseFilePrintData.AutoModeSpecified = true;
  licenseFilePrintData.AutoUsers = 5;
  licenseFilePrintData.AutoUsersSpecified = true;
  sysprep.LicenseFilePrintData = licenseFilePrintData;
  sysprep.LicenseFilePrintDataSpecified = true;

  UserData userData = new UserData();
  userData.FullName = "Test User";
  userData.OrgName = "Dell";
  userData.ComputerName = destVMName[i_clone];
  userData.ProductID = "ABCDE-FGHIJ-KLMNO-PQRYS-UVWXY";
  userData.ProductIDSpecified = true;
  sysprep.UserData = userData;

  Identification id = new Identification();
  id.JoinWorkgroup = "WORKGROUP";
```

```
  id.JoinWorkgroupSpecified = true;
  sysprep.Identification = id;

  Adapters adapters = new Adapters();
  Adapter[] tmp = new Adapter[1];
  tmp[0] = new Adapter();
  tmp[0].MACAddress = "MAC00"; // This is just a tag for the script, not the real MAC
  tmp[0].MACAddressSpecified = true;
  tmp[0].UseDHCP = false;
  tmp[0].UseDHCPSpecified = true;
  tmp[0].IPAddress = destIPAddress[i_clone];
  tmp[0].IPAddressSpecified = true;
  tmp[0].SubnetMask = "255.255.255.0";
  tmp[0].SubnetMaskSpecified  = true;

  DNSServers[] dnstmp = new DNSServers[1];
  dnstmp[0] = new DNSServers();
  string[] dnstmp_strarray = new string[1];
  dnstmp_strarray[0] = "192.168.1.10";
  dnstmp[0].DNSServer = dnstmp_strarray;
  tmp[0].DNSServers = dnstmp;

  Gateways[] gateways = new Gateways[1];
  Gateway[] gateway = new Gateway[1];
  gateways[0] = new Gateways();
  gateway[0] = new Gateway();
  gateway[0].datavalue = "192.168.1.1";
  gateway[0].CostMetric = 1;
  gateways[0].Gateway = gateway;
  tmp[0].Gateways = gateways;

  adapters.adapter = tmp;
  autoprep.adapters = adapters;
  autoprep.adaptersSpecified = true;

  autoprep.sysprep = sysprep;
  autoprep.sysprepSpecified = true;

  //Console.WriteLine("srcHandle= {0}, parentHandle= {1}, destHostHandle= {2}, " +
  //  "destVMName= {3},  datastore= {4}", srcHandle, parentHandle, destHostHandle,
  //  destVMName[i_clone], datastore[i_clone]);

// Note: failure of Clone operation doesn't always generate Exception!
try
  {
  VC = vma.CloneVM(srcHandle, parentHandle, destHostHandle, destVMName[i_clone],
    datastore[i_clone], autoprep, true);
  }
catch (Exception e)
  {
  Console.WriteLine("Error in Clone operation: " + e.Message);
  return;
  }

// Allow time for Virtual Center to start operation
Thread.Sleep(10000);

if (VC == null)
  {
  Console.WriteLine("ViewContents returned from CloneVM is null");
  }
else
  {
  if (VC.body == null)
    {
    Console.WriteLine("VC.body is null");
    }
  else
    {
    task = (Task) VC.body;
    // Note: task generated from VC = vma.CloneVM does not always have accurate
```

```
        // currentState (so it can't be used to track progress) but it does have accurate
        // queueTime, so use queueTime to look up task from array of tasks
        //Console.WriteLine("task: {0}  % completed: {1} {2} {3}",
        //  task.operationName, task.percentCompleted, task.currentState, task.queueTime);
        task_qt = task.queueTime;
        }
       }

    string taskhandle = vma.ResolvePath("/task");

    do
      {
      VC = vma.GetContents(taskhandle);
      c = (Container) VC.body;
      items = c.item;
      i=0;
      do
        {
        VC = vma.GetContents(items[i++].key);
        task = (Task) VC.body;
        } while (task.queueTime != task_qt);

      // Need to explicitly account for Daylight Savings Time due to .NET problem (not VMware
      // bug!)
      Console.WriteLine
        ("task: {0}  target VM: {1}  % completed: {2}  Status: {3}  Started: {4}  Now: {5}",
        task.operationName, destVMName[i_clone], task.percentCompleted, task.currentState,
        task.queueTime.AddHours(ctz.IsDaylightSavingTime(task.queueTime) ? -
          1:0).ToLongTimeString(),
        System.DateTime.Now.ToLongTimeString());
      Thread.Sleep(10000);
      } while((task.currentState.ToString() != "completed") && (task.currentState.ToString() !=
          "failed"));

      if (task.currentState.ToString() == "failed")
        Console.WriteLine("Clone {0} failed", destVMName[i_clone]);
      else
        Console.WriteLine("Clone {0} finished", destVMName[i_clone]);
      } // End Main for loop
    vma.Disconnect();
    } // End Main
  } // End public class VmaClient
} // End namespace vmclone1
```