

Implementing Fault-tolerance Through VMware Scripting and Dell OpenManage

By Dave Jaffe, Ph.D. and Todd Muirhead

vmall.cs

```
-- Copyright Dell Inc. 2005
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
            {

```

Implementing Fault-tolerance Through VMware Scripting and Dell OpenManage, continued

```

        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: 10/26/04
// Copyright Dell 2004
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];
    // Hard-coded migration pool for demo purposes
    string[] migration_pool_hostnames = {"esx-demo1", "esx-demo2", "esx-demo3"};
    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;

```

Implementing Fault-tolerance Through VMware Scripting and Dell OpenManage, continued

```

string vmname = null, target_hostkey = null;

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

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

VmaClient vma = new VmaClient();

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

// Verify source host is in migration pool
// In production look up ESX hosts using vma
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;
}

// 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_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;
        }
    }
    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);
        }
    }
}
}

```

Implementing Fault-tolerance Through VMware Scripting and Dell OpenManage, continued

```

do
{
    Thread.Sleep(2000);
    vc = vma.GetContents(taskhandle);
    task = (Task) vc.body;
    Console.WriteLine("task: {0} % completed: {1} {2} {3}",
        task.operationName, task.percentCompleted, task.currentState,
        task.queueTime);
    } while (task.currentState.ToString() != "completed");
}
}
}

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

public int get_host_cpu_time(string hostname)
{
    // get_host_cpu_time: Retrieve CPU time used (in ms)
    // of specified host in last sample
    // Author: Dave Jaffe
    // Last modified: 10/26/04
    // Copyright Dell 2004
    // 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;
    ViewContents vc;
    Host host;
    Container c;
    Item[] items;
    PerfCollection stats;
    string data_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++)
    {
        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)
                                {
                                    PerfStat stat = (PerfStat) stats.source[j].sample[0].stat[0];
                                    CPUPerf data = (CPUPerf) stat.data;
                                    data_string = data.used.ToString();
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return Convert.ToInt32(data_string);
}
}
}

```

remote2.vbs

```
-- Copyright Dell Inc. 2005
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
```