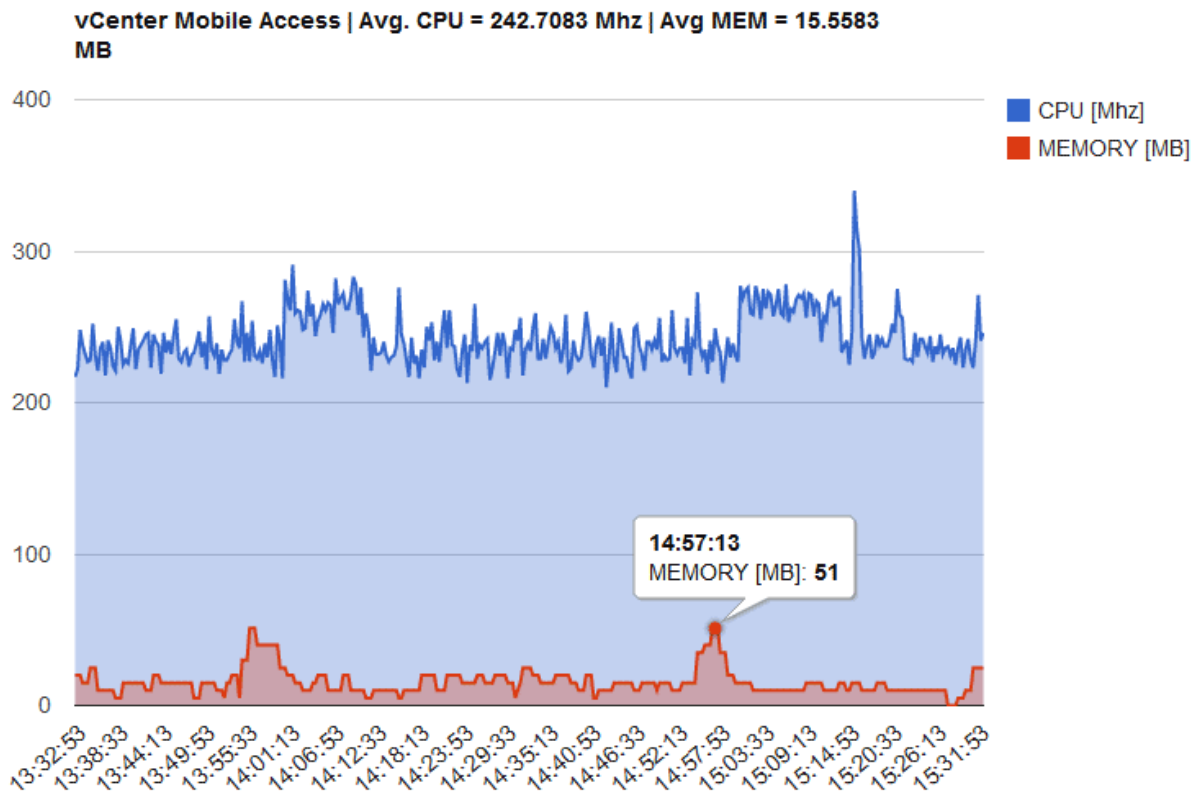
Teljesítmény alapú számlázás megvalósítása VMware platformon vSphere Web Services SDK felhasználásával

2011. december 3.



Virtualizációs technológiák és alkalmazásaik
VIMIAV89

Biri Mátyás

Tartalomjegyzék

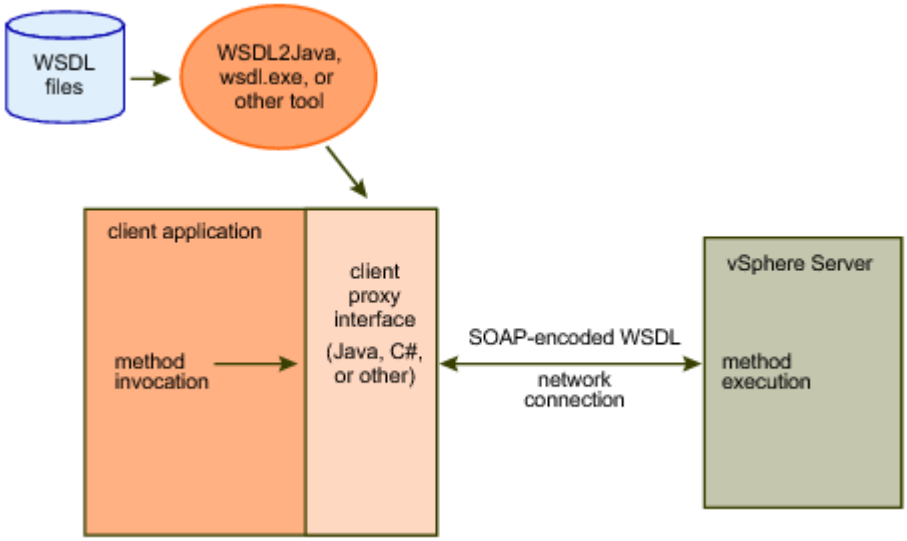
A projekt célkitűzése

A projekt célja megismerni a VMware Hypervisorokhoz tartozó management API-kat, majd a legmegfelelőbb eszközzel létrehozni egy szoftvert, ami monitorozza a virtuális gépek teljesítmény felhasználását adott standalone esxi hoston. A pillanatnyi teljesítmény mutatókat adatbázisban eltárolva, lehetőség nyílik kimutatásokat készíteni adott virtuális gép teljesítmény felhasználásáról tetszőleges idő intervallumra. Ilyen lehetséges kimutatás egy teljesítmény alapú számlázás, ami jelen projekt célja.

A vSphere WEB Services API

A VMware számos API-t nyújt Hypervisorjainak programozott kezelésére. Kezdve a hardver felügyeletet megkönnyítő CIM Interfacen át egészen a Windows Power Shell interfaceig. (Ezekről részletes tájékoztatás a http://www.vmware.com/support/developer weboldalon található.)

A sok közül egy, ami automatizált menedzsment feladatok ellátását segíti a **vSphere WEB Services API**. Ennek nagy erőssége, hogy nincs programnyelvhez kötve, hiszen a szerverrel való kommunikáció **SOAP** protokollon keresztül történik. Ez XML fájlok cseréjét jelenti HTTP protokoll felett. Az XML fájlok formátumát a https://<HOST_IP>/sdk/vim.wsdl fájl írja le. Ezt a kommunikációt pedig tetszőleges programnyelvben meg lehet valósítani.



Az API felhasználásának lehetőségei:

- Virtuális gépek készítése (akár template-ből), konfigurálása, ki-be kapcsolása
- Virtuális eszközök készítése, konfigurálása (CD-DVD meghajtók, virtuális hálókártyák, virtuális switchek, stb...)
- ESX és ESXI hostok ki-be kapcsolása, csatlakoztatása vCenterhez
- Virtuális gépek teljes "snapshot" managementje
- Statisztikák gyűjtése a Virtuális gépekről és a hostokról (**Ezt fogjuk most használni!**)
- Virtuális gépek mozgatása Hostok között (LiveMotion)

- VMware DRS és VMware HA konfigurálása (kizárólag vCenteren keresztül)

A VI (Virtual Infrastructure) API

A **VI API** gyakorlatilag egy **JAVA könyvtár,** ami a vSphere Web Services API-val való fejlesztést könnyíti meg. Levezényli a teljes SOAP kommunikációt a háttérben a Hypervisor és a kliens alkalmazás között. Ismeri a beburkolt API minden lehetőségét, a programozó felé a távoli objektumokat helyi objektumokra képezi le, s fenntartja a szinkront (ha helyi objektumon változtatunk, a Hypervisoron is életbe lép a változás).

Nézzünk egy példa kódot, ami kiírja host típusát és az elérhető API verziót:

```
1.  import java.net.URL;
2.  import com.vmware.vim25.*;
3.
4.  public class HelloVI
5.    {
6.      public static void main(String[] args) throws Exception
7.        {
8.          if(args.length != 1)
9.            {
10.             System.out.println("Usage: java HelloVI <url>");
11.           }
12.
13.         //ignore the SSL certificate
14.         System.setProperty(
15.         "org.apache.axis.components.net.SecureSocketFactory",
16.         "org.apache.axis.components.net.SunFakeTrustSocketFactory"
17.         );
18.
19.         //create the interface on which services are defined
20.         VimServiceLocator vsl = new VimServiceLocator();
21.         VimPortType vimService = vsl.getVimPort(new URL(args[0]));
22.
23.         //create a ManagedObjectReference to the ServiceInstance
24.
25.         ManagedObjectReference siMOR = new ManagedObjectReference();
26.         siMOR.set_value("ServiceInstance");
27.         siMOR.setType("ServiceInstance");
28.
29.         //retrieve ServiceContent data object from ServiceInstance
30.         ServiceContent sc = vimService.retrieveServiceContent(siMOR);
31.
32.         //get the AboutInfo object from ServiceContent
33.         AboutInfo ai = sc.getAbout();
34.
35.         //print out just one of the many properties here
36.         System.out.println("Hello " + ai.getName());
```

```
37.         System.out.println("API type: " + ai.getApiType());
38.       }}
```

Ha látni szeretnénk pontosan milyen SOAP kommunikáció zajlik a háttérben azt egy forgalomanalizáló programmal megtehetjük. (pl.: WireShark)

A bemutatott példakód nagyon egyszerű volt, nem minden lehetőség érhető el ilyen könnyen. Az API működése mégsem túl bonyolult. A szerverre való csatlakozás után különböző osztályok segítségével kiszűrhetjük a számunkra szükséges ManagedObjecteket (Ilyen pl. a virtuális gép, egy virtuális hálókártya, switch, gyakorlatilag minden aminek beállításai vannak), amiknek visszakapjuk a referenciáit. Ezeket a referenciákat felhasználva feladatokat hajthatunk végre az adott objektumon (pl. VM_Shutdown()), vagy egy PropertyCollector osztály segítségével lekérdezhetjük és módosíthatjuk az egyes beállításait.

A keretrendszer dokumentációja, elérhető a VMware hivatalos weboldalán:
http://www.vmware.com/support/developer/vc-sdk/visdk25pubs/ReferenceGuide/

A keretrendszer megismerésére a ”**VMware VI and vSphere SDK: Managing the VMware Infrastructure and vSphere”** című könyvet bátran merem ajánlani. (http://www.amazon.com/VMware-VI-vSphere-SDK-Infrastructure/dp/0137153635)
650 oldalas terjedelme ellenére gyorsan meg lehet belőle tanulni az API használatát, részletesen bemutatja a hátterét, sok példával segíti a megértést. Jól tagolt, amennyiben csak 1-1 része érdekel az API-nak akkor is érdemes fellapozni.

A VI JAVA Keretrendszer

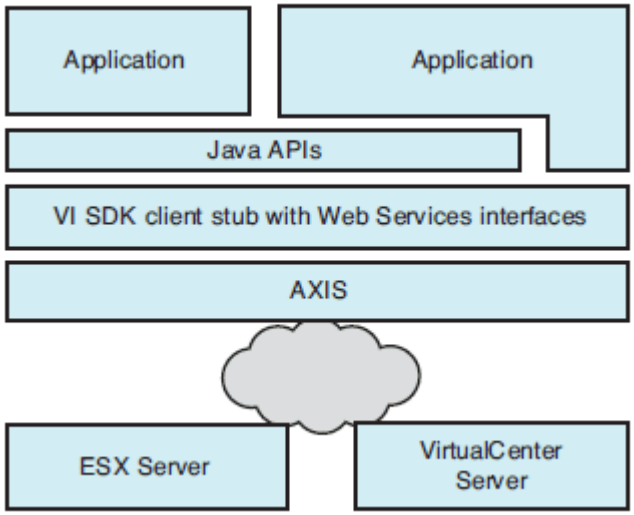
Az előző részben bemutatott API használatát megkönnyítő, arra ráépülő API. Sokkal egyszerűbb és gyorsabb benne kódolni. Gyakorlatilag burkoló osztályokat használ, tudja, hogy milyen távoli objektumnak milyen beállításai és manipulációs lehetőségei vannak, ezek felajánlja nekünk egy jól átgondolt rendszerben. Nem kell PropertyCollector, s hasonló osztályokkal bajlódnunk többé.

Jóval többet mond egy példakód látványos egyszerűsége. A kód lekérdezi a hoston található VM-eket, majd kiírja azok elnevezését, s aktuális futási állapotát. (Kikapcsolva, bekapcsolva, hibernálva)

```
1.  import java.net.URL;
2.
3.  import com.vmware.vim25.mo.*;
4.
5.  public class PrintInventory
6.  {
7.      public static void main(String[] args) throws Exception
8.      {
9.
10.         ServiceInstance
    si = new ServiceInstance(new URL("https://server/sdk"), "root", "password",tr
    ue);
11.         Folder rootFolder = si.getRootFolder();
12.
13.         ManagedEntity[] vms = new
    InventoryNavigator(rootFolder).searchManagedEntities("VirtualMachine");
14.         for(int i=0; i<vms.length; i++)
15.         {
16.            System.out.println("vm["+i+"]=" + vms[i].getName());
17.            System.out.println("\t" + vms[i].getSummary().getRuntime().getPowe
    rState());
18.         }
19.
20.         si.getServerConnection().logout();
21.     }
22. }
```

Ez a keretrendszer egy jó háttértámogatással rendelkező OpenSource kezdeményezés. Minden információ, dokumentáció megtalálható a weboldalán: http://vijava.sourceforge.net/
A fejlesztés vezetője **Steve Jin,** aki az említett 650 oldalas könyvet írta a VI api-ról. 16 éve VMware alkalmazott és virtualizáció menedzsmenttel foglalkozik. Hasznos olvasmány a blogja, ahol sok példa és érdekesség olvasható a keretrendszerről, s általában a virtualizáció menedzsment kihívásairól. (http://www.doublecloud.org/).

A továbbiakban a projekt fejlesztése ennek a keretrendszernek a felhasználásával fog történni. Ez nem jelent megkötést, mert kompatibilis az alap VI API-val, így kódon belül is bármikor át lehet térni alacsonyabb szintre, ha szükség lenne rá.



A teljesítmény mérő alkalmazás megvalósítása

Az alkalmazás logikája a következő: Lekérdezzük a virtuális gépeket, amik az adott hoston vannak. Ha éppen futnak, akkor lekérdezzük az aktuális processzor, és memória használatukat, majd letároljuk adatbázisban, egy időbélyeggel együtt. Mindezt megismételjük 20 másodpercenként, így lesz egy adatbázisunk, amiből tetszőleges kimutatásokat készíthetünk később. Azért 20 másodperc, mert ilyen időközönként frissíti alapértelmezetten a host a VM-ek teljesítmény fogyasztását.

Hogyan is néz ki mindez kódban?

Virtuális gépek lekérése

```
1.  ServiceInstance si = null;
2.
3.    try{
4.      si = new ServiceInstance(new URL("https://" + Main.config.getPrope
    rty("esxi.host") + "/sdk"), Main.config.getProperty("esxi.username"),
    Main.config.getProperty("esxi.password"), true);
5.     }
6.    catch(Exception e){
7.      System.out.println("Unable connect to host: " + e.getMessage());
8.      System.exit(1);
9.     }
10.
11. Folder rootFolder = si.getRootFolder();
12.
13. //Query VM list
14.    ManagedEntity[] vms = null;
15.
16.    try{
17.      vms = new InventoryNavigator(rootFolder).searchManagedEntities("V
    irtualMachine");
18.    }
19.    catch(Exception e)
20.     {
21.      System.out.println("Unable to receive list of VMs:
    " + e.getMessage());
22.     }
```

Azokkal a virtuális gépekkel foglalkozunk, amik éppen futnak

```
1.  for(int i=0; i < vms.length; i++)
2.    {
3.         VirtualMachine actualVM = (VirtualMachine)vms[i];
4.
5.       if(actualVM.getSummary().getRuntime().getPowerState().equals(Virt
    ualMachinePowerState.poweredOn))
6.         {
7.           //DO SOMETHING
8.         }
9.      }
```

Ha minden feltétel teljesül, kérdezzük le a virtuális gép teljesítmény mutatóit

A könyv 8. fejezetét átfutva, megtaláljuk, hogy minden VirtualMachine objektumnak van egy „VirtaulMachine Summary” tulajdonsága, ami az alábbi információkat tartalmazza a virtuális gépről:

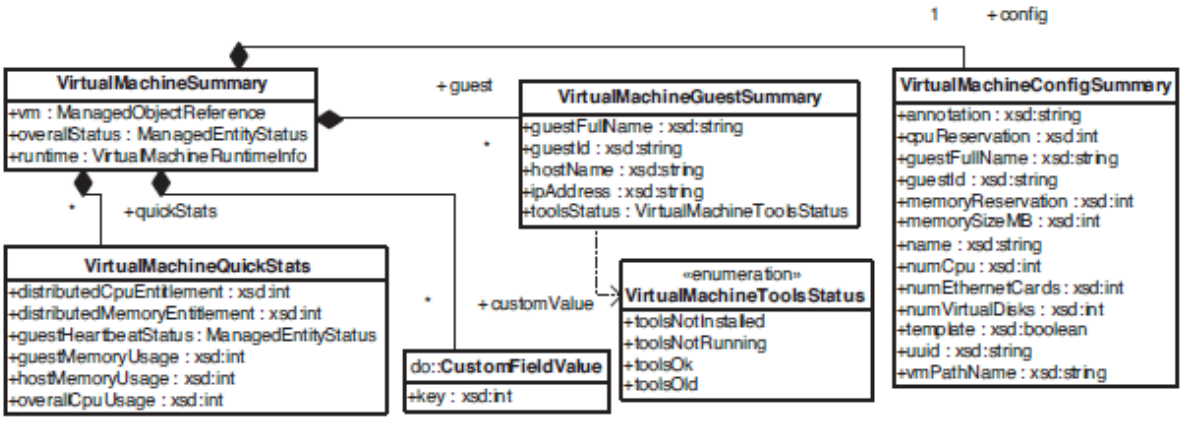


Figure 8-5 `VirtualMachineSummary` data object

A VI API dokumentációjából* pedig az alábbi derül ki a VirtualMachineQuickStats objektum tulajdonságairól:

guestMemoryUsage:

Guest memory utilization statistics, in MB. This is also known as active guest memory. The number can be between 0 and the configured memory size of the virtual machine. Valid while the virtual machine is running.

overallCpuUsage

Basic CPU performance statistics, in MHz. Valid while the virtual machine is running.

*http://www.vmware.com/support/developer/vc-sdk/visdk25pubs/ReferenceGuide/vim.vm.Summary.QuickStats.html

Éppen erre a két adatra van szükségünk. Ennek alapján a kód:

```
1.  VirtualMachineQuickStats
    quickstats = actualVm.getSummary().getQuickStats();
2.
3.      cpuCurrent = quickstats.getOverallCpuUsage();
4.      memoryCurrent = quickstats.getGuestMemoryUsage();
```

Eredmények adatbázisba mentése

Egyedileg azonosítanunk kell, hogy melyik virtuális géphez tartozik a mért adat, mielőtt adatbázisba mentenénk. Erre a virtuális gép neve nem alkalmas, mert bármikor megváltozhat. Ami nekünk kell az a **uuid** tulajdonsága a virtuális gépnek. Ez szintén megtalálható a fenti UML diagramban. Lekérdezhető az alábbi paranccsal: **vmuuid = actualVM.getConfig().getUuid();**

A méréseket időben is azonosítanunk kell. Egy időbélyeget hozzárendelve a minták készen állnak adatbázisba töltésre. (egy ingyenes MySQL több mint elég lesz). Az adatbázis tábla felépítése az előzmények alapján:

```
1.  CREATE TABLE IF NOT EXISTS `performanceshots` (
2.    `vmuuid` VARCHAR(255) NOT NULL,
3.    `timestamp` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CU
    RRENT_TIMESTAMP,
4.    `cpu` INT(11) NOT NULL COMMENT '[Mhz]',
5.    `mem` INT(11) NOT NULL COMMENT '[MB]'
6.  ) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Érdemes egy további táblát is létrehozni, ami többlet infót is eltárol a virtuális gépről. Ezek hasznosak lesznek kimutatások készítésekor.

```
1.  CREATE TABLE IF NOT EXISTS `vmconfig` (
2.    `vmuuid` VARCHAR(255) NOT NULL,
3.    `vmname` VARCHAR(255) NOT NULL,
4.    `cpureserve` INT(5) NOT NULL,
5.    `cpulimit` INT(5) NOT NULL,
6.    `memreserve` INT(6) NOT NULL,
7.    `memlimit` INT(6) NOT NULL,
8.    UNIQUE KEY `vmuuid` (`vmuuid`)
9.  ) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Ezzel a két metrikával (processzor és memória használat) egy virtuális gép egy órányi futásához tartozó teljesítmény adatai 20 másodperces bontásban 10KB tárterületen elférnek. Így akár egy éves teljesítmény előzmény is eltárolható!

Felmerülő problémák

Időzítés: Sok virtuális gép esetén több másodpercig is eltarthat, amíg lekérdezzük a teljesítmény mutatókat, így nem lesz mindegyik mintának ugyanaz az időbélyege, 1-2 másodperc elcsúszás lesz. Ez később problémákat okozhat, amikor össze szeretnénk hasonlítani adott pillanatban az összes virtuális gép teljesítményét, ezért célszerűbb most megoldani, hogy egy mintavételi körben mindegyik minta ugyanazt az időbélyeget kapja.

Rugalmasság: Új virtuális gép létrehozását célszerű a programnak automatikusan észlelnie, hogy azonnal mérni kezdhesse a teljesítmény felhasználást.

Megvalósítás: A konkrét megvalósítás programozás technikai kérdés, nem tartozik jelen dokumentum közvetlen céljai közé ennek részletes bemutatása. Mindenesetre közlöm a végleges forráskódot, ami tartalmazza a megoldásokat is. Maga a forráskód nem elegáns, sokat kellett trükközni az időzítésekkel, mert nem lehetett többszálúsítani a nem szálbiztos adatbázis kezelő osztály miatt...

Ahhoz, hogy ez a kód forduljon számos egyedi osztályra van még szükség, ezeket nem csatolom, mert ezek nélkül is érthető a program logikája.

A forrás kód:

```
1.  /*
2.   * To change this template, choose Tools | Templates
3.   * and open the template in the editor.
4.   */
5.  package esximonitor;
6.
7.  import com.vmware.vim25.*;
8.  import java.net.URL;
9.  import com.vmware.vim25.mo.*;
10. import java.util.ArrayList;
11.
12. /**
13.  *
14.  * @author Biri Mátyás
15.  */
16. public class Main {
17.
18.
19.   public static mysqlConnection Mysqlconn = new mysqlConnection();
```

```
20.  public static centralConfig  config; //Initialized later
21.  public static logging     logging  = new logging();
22.
23.  /**
24.   * @param args the command line arguments
25.   */
26.  public static void main(String[] args) {
27.    // TODO code application logic here
28.
29.    /*******************************
30.    ** Building configs from file **
31.    *******************************/
32.       if(args.length == 0)
33.        {
34.         System.out.println("Location of config file is not presented in the
   first argument");
35.         System.out.println("Program terminates");
36.         System.exit(1);
37.        }
38.
39.      try{
40.         config = new centralConfig(args[0]);
41.        }
42.      catch(Exception E){
43.         System.out.println("Unable to load config file: " + E.getMessage());
44.         System.exit(1);
45.        }
46.
47.     /************************
48.     ** Connect to MySQL
49.     ************************/
50.      Mysqlconn.connect(true); //Exit on fail => true
51.
52.    /************************
53.     ** CONNECT TO ESXI
54.     ************************/
55.
56.
57.    ServiceInstance si = null;
58.
59.    try{
60.        si = new ServiceInstance(new URL("https://" + Main.config.getProperty("e
   sxi.host") + "/sdk"), Main.config.getProperty("esxi.username"),
   Main.config.getProperty("esxi.password"), true);
61.      }
62.    catch(Exception e){
63.        System.out.println("Unable connect to host: " + e.getMessage());
64.        System.exit(1);
65.      }
```

```
66.
67.     //Read timing parameters from config file
68.     int performanceCheckInterval =Integer.parseInt(Main.config.getProperty("esxi
    .performanceCheckInterval"));
69.     int performanceChecksPerReScanInventory=Integer.parseInt(Main.config.getProp
    erty("esxi.performanceChecksPerReScanInventory"));
70.
71.     /************************
72.      ** START INFINITE LOOP
73.      ************************/
74.
75.     Folder rootFolder = si.getRootFolder();
76.     long lastforcedTimeStamp = 0; //Need to know for correct timing
77.
78.     while(true)
79.     {
80.       System.out.println("\n\n############################");
81.       System.out.println("## SEARCH FOR RUNNING VMS AND UPDATE THEIR CONFIG");
82.       System.out.println("############################\n\n");
83.
84.       //Query VM list
85.       ManagedEntity[] vms = null;
86.
87.       try{
88.          vms = new InventoryNavigator(rootFolder).searchManagedEntities("Virtual
    Machine");
89.       }
90.       catch(Exception e)
91.         {
92.           System.out.println("Unable to receive list of VMs:
    " + e.getMessage());
93.         }
94.
95.        /************************
96.        ** FOR EVERY VM
97.        ************************/
98.
99.       for(int i=0; i< vms.length; i++)
100.          {
101.            VirtualMachine actualVM = (VirtualMachine)vms[i];
102.
103.            //For running VM-s only
104.            //Get uuid name, cpu memo limits and store into DB
105.           if(actualVM.getSummary().getRuntime().getPowerState().equals(Virtual
    MachinePowerState.poweredOn))
106.             {
107.               VMConfig VMConfigShot = new VMConfig(actualVM);
108.                 VMConfigShot.print();
109.
```

```
110.                     try{
111.                         VMConfigShot.savetoDB();
112.                       }
113.                     catch(Exception e)
114.                       {
115.                         Main.logging.error("Unable to write config shot to DB:
    " + e.getMessage());
116.                         System.out.println("Unable to write config shot to DB:
    " + e.getMessage());
117.                       }
118.                }
119.            }
120.
121.
122.          System.out.println("\n\n##########################");
123.          System.out.println("## MEASURE PERFORMACE FOR A WHILE");
124.          System.out.println("##########################\n\n");
125.
126.          ArrayList<VMperformanceShot> shotlist = new ArrayList<VMperformanceShot
    >();
127.
128.           //If first entry, but not the first big round (outer while) need to
    sleep some for correct timing
129.           for(int q=0;q < performanceChecksPerReScanInventory;q++)
130.             {
131.              if(q == 0 && lastforcedTimeStamp != 0)
132.              {
133.                 try{
134.                   long TimeToSleep = (lastforcedTimeStamp + performanceCheckInte
    rval) -System.currentTimeMillis()/1000;
135.
136.                   System.out.println("Sleep for " + TimeToSleep + "
    seconds...");
137.                   if(TimeToSleep <0)
138.                     TimeToSleep = 0;
139.                   Thread.sleep(1000 * TimeToSleep);
140.                 }
141.                 catch(Exception e)
142.                   {
143.                    System.out.println("Unable to sleep thread! Maybe
    performanceCheckInterval is to short for reScan the
    inventory...");
144.                   }
145.              }
146.
147.              System.out.println("\n\n## START NEW PERFORMANCE MEASURMENT
    ROUND\n\n");
148.
```

```
149.             long forcedTimeStamp = System.currentTimeMillis()/1000; //Force
    shots to see this timestamp (if query takes more than 1 sec, need to be the same
    timestamp)
150.             //Load shots, but leave DB now (Increase query performance)
151.             for(int i=0; i< vms.length; i++)
152.               {
153.                   VirtualMachine actualVM = (VirtualMachine)vms[i];
154.
155.                 if(actualVM.getSummary().getRuntime().getPowerState().equals(V
    irtualMachinePowerState.poweredOn))
156.                   {
157.                     shotlist.add(new VMperformanceShot(actualVM,forcedTimeSta
    mp));
158.
159.                   }
160.               }
161.
162.             //Store shots into DB
163.             for(int i=0;i<shotlist.size();i++)
164.             {
165.               shotlist.get(i).print();
166.
167.               try{
168.                   shotlist.get(i).savetoDB();
169.                 }
170.               catch(Exception e)
171.                 {
172.                   Main.logging.error("Unable to write performance shot to DB:
    " + e.getMessage());
173.                 }
174.             }
175.
176.             //DROP shots
177.             shotlist.clear();
178.
179.           //Sleep a while tzhen collect shots again!
180.           if(q != performanceChecksPerReScanInventory - 1)
181.            {
182.               try{
183.                 long TimeToSleep = (forcedTimeStamp + performanceCheckInterval
    ) -System.currentTimeMillis()/1000;
184.
185.                 System.out.println("Sleep for " + TimeToSleep + "
    seconds...");
186.                 if(TimeToSleep <0)
187.                   TimeToSleep = 0;
188.
189.
190.                 Thread.sleep(1000 * TimeToSleep);
```

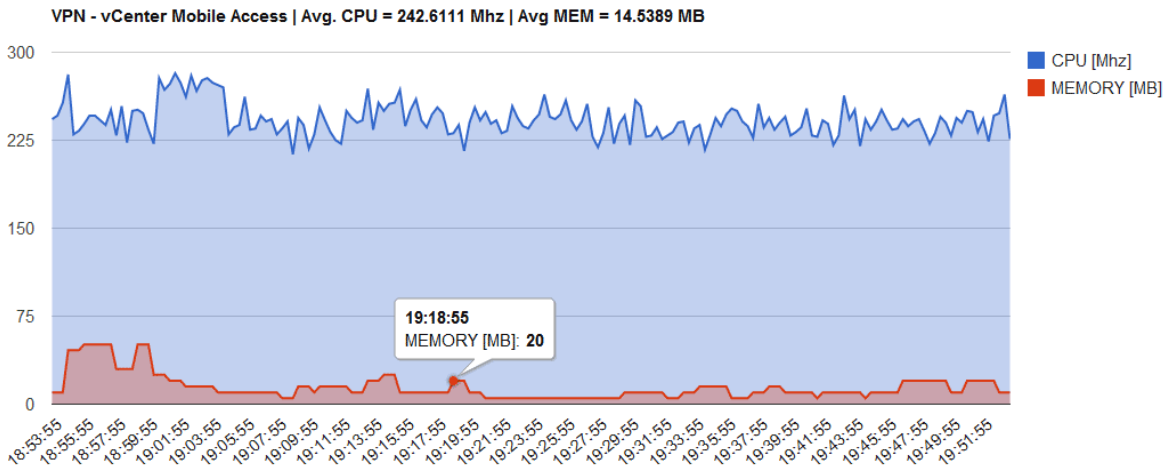
```
191.                 }
192.                 catch(Exception e)
193.                   {
194.                    System.out.println("Unable to sleep thread!");
195.                    System.exit(0);
196.                   }
197.             }
198.             else
199.             {
200.               lastforcedTimeStamp = forcedTimeStamp;
201.             }
202.           }
203.       }
204.
205.
206.       //Disconnect from server (will never reach!)
207.       //si.getServerConnection().logout();
208.     }
209.   }
```

Mérési eredmények grafikus ábrázolása

A projekt jelenlegi állása szerint van egy háttérben futó alkalmazásunk, ami folyamatosan gyűjti a statisztikákat egy adatbázisba. Ennek az adatbázisnak egy felhasználási lehetősége a sok közül egy alkalmazás, ami webes felületen követhetővé teszi grafikus formában a virtuális gépek teljesítmény változásait. **Ez ismét egyszerű programozás, ezért csak a végeredményt közlöm.**

A program PHP nyelven íródott és a Google Charts API-t használja vizualizációk készítésére. A Google Charts API-nak egyszerűen át kell adni egy mátrixot, ami tartalmazza a megjelenítendő adatokat, s a **kliens számítógép teljesítményét felhasználva** rajzol belőle grafikont.

Az elkészült program a képen látható minőségű grafikonokat tud generálni, tetszőleges paraméterezéssel (időtartomány, méret, stb…):



30 napos intervallumra felparaméterezve a grafikont megjeleníti a havi átlag teljesítményt (lásd. grafikon fejléce), ami alapján lehetséges teljesítmény alapon számlázni. Ez volt a projekt célja.

A grafikon generáló forráskódja:

```
1. <?php
2. $graph_with = 1200;
3. $graph_height = 250;
4. $perfSamples = 4320;
5. $margin = 40;
6. $hours = 1;
7.
8. if(!empty($_GET['width']))
```

```
9.   $graph_with = $_GET['width'];
10.
11. if(!empty($_GET['height']))
12.  $graph_height = $_GET['height'];
13.
14. if(!empty($_GET['samples']))
15.  $perfSamples = $_GET['samples'];
16.
17. if(!empty($_GET['margin']))
18.  $margin = $_GET['margin'];
19.
20. if(!empty($_GET['hours']))
21.  $hours = $_GET['hours'];
22.
23. ?>
24.
25.  <html>
26.   <head>
27.   <?php if(!empty($_GET['refresh'])) {echo '<meta http-equiv="refresh"
    content="20">';} ?>
28.   <script type="text/javascript" src="https://www.google.com/jsapi"></script>
29.   <script type="text/javascript">
30.    google.load("visualization", "1", {packages:["corechart"]});
31.    google.setOnLoadCallback(drawChart);
32.    function drawChart() {
33.
34. <?php
35.
36.
37. $mysqli = new mysqli('127.0.0.1',user,pass,'esximonitor');
38. ##############################
39. ## Query uuids
40. ##############################
41.
42. $sql = "SELECT vmuuid FROM vmconfig ORDER BY vmname DESC;";
43. if($result = $mysqli->query($sql))
44.  {
45.   while ($row = $result->fetch_object())
46.   {
47.     $vmuuids[] = $row->vmuuid;
48.    }
49.   foreach($vmuuids as $uuid)
50.     {
51.
52.      $uniqueid = 'a' . md5($uuid);
53.
54.
55.      $sql = "SELECT *
56.     FROM `performanceshots`
```

```
57.     WHERE `vmuuid` LIKE '" . $uuid . "'
58.       AND timestamp > '" . date("Y-m-d H:i:s",date("U") -
    ($hours * 3600)) . "'
59.     ORDER BY `performanceshots`.`timestamp` ASC LIMIT " . $perfSamples;
60.
61.      if($result = $mysqli->query($sql))
62.        {
63.          //$result->num_rows
64.
65.          echo "
66.         var " . $uniqueid . " = new google.visualization.DataTable();
67.
68.         " . $uniqueid . ".addColumn('string', 'Time');
69.         " . $uniqueid . ".addColumn('number', 'CPU [Mhz]');
70.         " . $uniqueid . ".addColumn('number', 'MEMORY [MB]');
71.
72.         " . $uniqueid . ".addRows([";
73.
74.          while ($row = $result->fetch_object())
75.            {
76.
77.
78.
79.              echo "['" . date("H:i:s",strtotime($row-
   >timestamp)) ."'," . $row->cpu . ", " . $row->mem ."],";
80.
81.
82.
83.
84.            }
85.            echo ']);';
86.        }
87.      else
88.        {
89.          echo $mysqli->error;
90.        }
91.
92.
93.      $sql = "SELECT *
94.     FROM `vmconfig`
95.     WHERE `vmuuid` LIKE '" . $uuid . "'";
96.      $result = $mysqli->query($sql);
97.      $row = $result->fetch_object();
98.      $vmname = $row->vmname;
99.
100.           $sql = "SELECT avg(cpu) as avgcpu,avg(mem) as avgmem
101.          FROM `performanceshots`
102.          WHERE `vmuuid` LIKE '" . $uuid . "'
```

```
103.            AND timestamp > '" . date("Y-m-d H:i:s",date("U") -
    ($hours * 3600)) . "';";
104.           $result = $mysqli->query($sql);
105.           $row = $result->fetch_object();
106.           $avgcpu = $row->avgcpu;
107.           $avgmem = $row->avgmem;
108.
109.
110.           echo "var chart" . $uniqueid . " = new
    google.visualization.AreaChart(document.getElementById('chart_div" . $uniquei
    d . "'));
111.               chart" . $uniqueid . ".draw(" . $uniqueid . ", {width:
    " . $graph_with . ", height: " .$graph_height . ", title: '" . $vmname . " |
    Avg. CPU = " . $avgcpu ." Mhz | Avg MEM = " . $avgmem . " MB',
112.               vAxis: {},
113.               hAxis: {slantedTextAngle: 40}
114.                       });";
115.          }
116.
117.       }
118.      else
119.       {
120.        echo $mysqli->error;
121.       }
122.       /*----------------------*/
123.      $mysqli->close();
124.
125.      ?>
126.
127.      }
128.       </script>
129.       </head>
130.       <body>
131.       <?php
132.       foreach($vmuuids as $uuid)
133.       {
134.
135.        $uniqueid = 'a' . md5($uuid);
136.       echo '<div style="margin:' . $margin . 'px; display:inline-block;"
    id="chart_div' . $uniqueid . '"></div>';
137.       }
138.       ?>
139.
140.       </body>
141.      </html>
```

Összefoglalás és a projekt további lehetőségei

A projekt során teljesen ingyenes eszközök felhasználásával **elkészült egy program**, ami bármely esxi hosthoz (vagy akár vCenterhez) kapcsolódva 20 másodperces bontásban lekérdezi a pillanatnyilag futó virtuális gépek teljesítményét (processzor és memória felhasználás), majd egy MySQL adatbázisba menti a mintákat. Az adatbázisból tetszőleges intervallumra jól paraméterezhető grafikonokat készít, átlagteljesítményt számít. A program nem igényel konfigurációt, kizárólag az esxi host és az adatbázis elérhetőségét. Észleli, ha új virtuális gép indul el a hoston, elkezdi mérni a teljesítményét automatikusan.

A program jelenleg is használatban van, standalone 4.1-es esxi hosthoz kapcsolódva 168 óra folyamatos futás után is stabil. A kliens program platform független. Windows 7 és Debian 6.03 operációs rendszereken lett tesztelve.

A programnak ismert hiányossága grafikon rajzoláskor, hogy amennyiben bizonyos intervallumra nincs minta, akkor nem nulla értéket jelez a grafikonon, hanem átugorja az adott intervallumot. Így nem látszik a grafikonon, hogy leállt egy virtuális gép, csak ha jól szemügyre vesszük az idő tengely feliratait. Ez nem szép megoldás, de könnyen javítható a PHP programban.

A program továbbfejlesztésében két komolyabb lehetőséget látok. Az egyik szerint a teljesítményt több metrikával is lehetne mérni. (HDD teljesítmény, Network I/O, stb.) Nem várt, vagy nem jellemző teljesítmény változás esetén akár SMS-ben, vagy automata telefonhívással lehetne értesíteni az adminisztrátort. (Ilyen szoftver már létezik, de drága: http://www.uptimesoftware.com/vmware-monitoring.php) A másik irány szerint különböző trigger feltételek szerint a rendszer automatikusan is képes lenne beavatkozni a host működésébe. Észlelni tudná az „elszállt” operációs rendszereket a virtuális gépekben, s újraindítaná, stb. A trigger feltételeket, s a teljesülésük esetén végrehajtandó feladatokat a program fájlból olvasná fel, így könnyen konfigurálható lenne a működése.

Biri Mátyás
2011. december 3.