Automation and Optimization in IT-DB OracleVM virtualization systems

Tomas Tauber

CERN openlab

20 August 2010

oTN-2010-01 openlab Summer Student Report

Automation and Optimization in IT-DB OracleVM virtualization systems

Tomas Tauber
Giacomo Tenaglia & Luigi Gallerani

August 2010
Version 1.2
Distribution: **Public**

Abstract

The goal of this openlab summer student project was to have a transparent, dynamic and automated management of virtual machine units using OracleVM, the current IT-DB virtualization technology, architecture and configuration. The other aim of this work was to optimize the hardware usage in order to run as much virtual machines as possible on a single physical host.

Introduction

The first chapter explains the architecture concept and then focuses on various categories of automating the virtual machine management. In the second chapter, the memory overcommitment technique, *“ballooning”*, is described, followed by performed tests specifications and their results.

The intention is to integrate Oracle VM, the server virtualization solution from Oracle Corporation, based on the open-source Xen hypervisor technology, into the current CERN IT-DB infrastructure. The first major step which had been already done was to have the virtual machines managed by OVM compatible with Quattor.

The other challenge is to make use of the features of these virtualization technologies which are key solutions to many current IT problems. One of the major tasks is to have a better utilization of new hardware by running several virtual machines on it. Nowadays, memory is becoming a bottleneck in virtualized systems, hence performance tests of physical hosts, using some memory overcommitment techniques, give a good background for determining the optimal number of virtual machines that could run on a single physical host.

Xen hypervisor technology[1]

The Xen hypervisor is a layer of software running directly on computer hardware replacing the operating system thereby allowing the computer hardware to run multiple guest operating systems concurrently. Support for x86, x86-64, Itanium, Power PC, and ARM processors allow the Xen hypervisor to run on a wide variety of computing devices and currently supports Linux, NetBSD, FreeBSD, Solaris, Windows, and other common operating systems as guests running on the hypervisor. The Xen.org community develops and maintains the Xen hypervisor as a free solution licensed under the GNU General Public License. A computer running the Xen hypervisor contains three components:

- **Xen Hypervisor**
- Domain 0, the Privileged Domain (**Dom0**) – Privileged guest running on the hypervisor with direct hardware access and guest management responsibilities
- Multiple DomainU, Unprivileged Domain Guests (**DomU**) – Unprivileged guests running on the hypervisor; they have no direct access to hardware (e.g. memory, disk, etc.)

The Xen hypervisor runs directly on the hardware and becomes the interface for all hardware requests such as CPU, I/O, and disk for the guest operating systems. By separating the guests from the hardware, the Xen hypervisor is able to run multiple operating systems securely and independently. The Domain 0 Guest referred to as Dom0 is launched by the Xen hypervisor during initial system start-up and can run any operating system except Windows. The Dom0 has unique privileges to access the Xen hypervisor that is not allocated to any other Domain Guests. These privileges allow it to manage all aspects of Domain Guests such as starting, stopping, I/O requests, etc. A system administrator can log into Dom0 and manage the entire computer system. The Domain Guests referred to as DomUs or unprivileged domains are launched and controlled by the Dom0 and independently operate on the system. These guests are either run with a special modified operating system referred to as paravirtualizion or unmodified operating systems leveraging special virtualization hardware (Intel VT and AMD-V) referred to as hardware virtual machine (HVM). Note – Microsoft Windows requires a HVM Guest environment.

[1]Stephen Spector, 'New to Xen Guide', *xen.org*, 15 July 2010, <http://www.xen.org/files/Marketing/NewtoXenGuide.pdf> [accessed 4 August 2010]

Paravirtualization

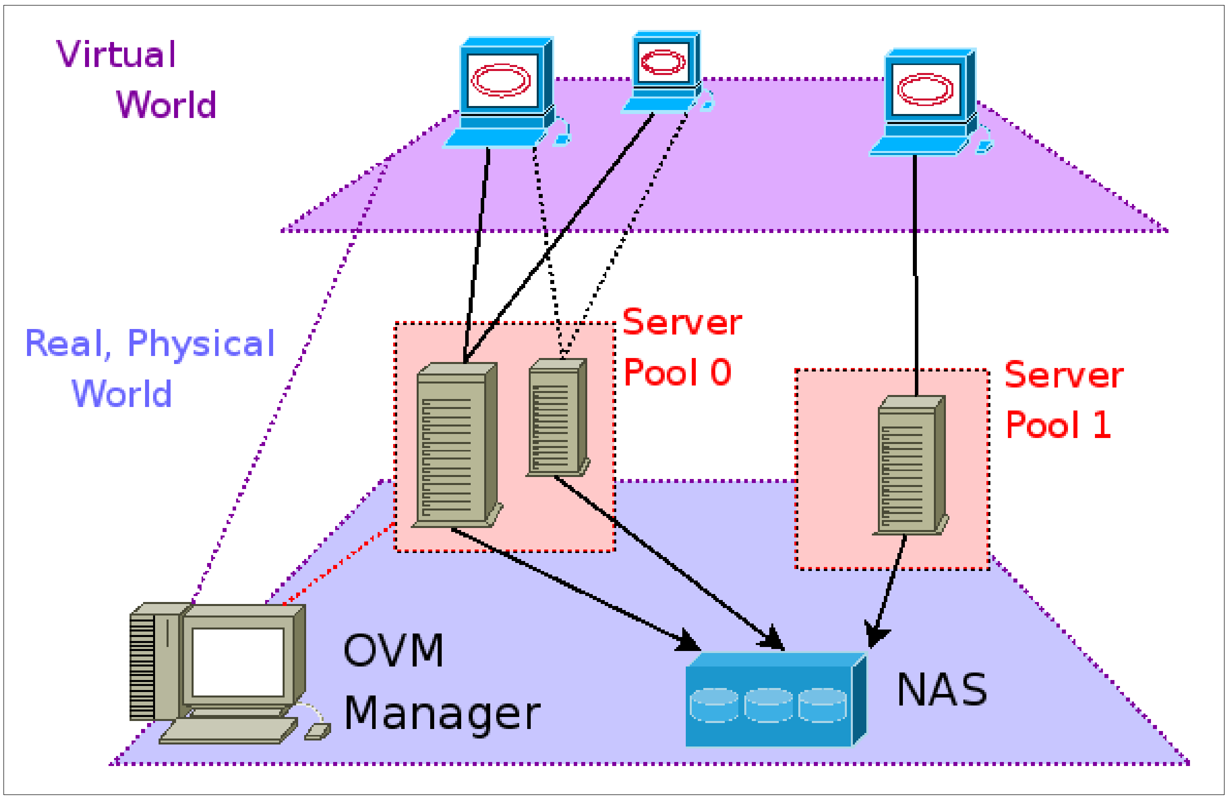
A term used to describe a virtualization technique that allows the operating system to be aware that it is running on a hypervisor instead of base hardware. The operating system must be modified to accommodate the unique situation of running on a hypervisor instead of basic hardware.

Hardware Virtual Machine (HVM)

A term used to describe an operating system that is running in a virtualized environment unchanged and unaware that it is not running directly on the hardware. Special hardware is required to allow this, thus the term HVM.

1 Automation

1.1 Architecture design[*]



OracleVM Architecture design
Figure 1

As shown in Figure 1, we have two worlds: the real one with physical servers and the virtual one with running virtual machines. In the real world, there are the physical servers which run the OracleVM Server. These servers are connected to NAS in order to share a folder, called OVS (VM Repository). In this repository, the images of the virtual machines and the configuration files are stored. The virtual machines run on the OracleVM Server. If the OracleVM Servers are logically grouped in a server pool, the virtual machines can run on each server in the same server pool. An external computer, with Oracle HTTP Server (based on Apache HTTP server) installed, can run the OC4J based web application called OracleVM Manager. OracleVM Manager can start and stop both physical and virtual machines, migrate virtual machines from one physical server to another, define server pools, create and delete virtual machines. In this picture, we have 3 OracleVM Servers, 2 are grouped in the ServerPool0. Also, we have 3 virtual machine running. The first 2 are running on the first server of the ServerPool0, but they could

[*] The picture and background information kindly provided by Luigi Gallerani

be run on the second server too, because the servers are in the same pool. The ServerPool1 is composed of a single OracleVM Server and on this Server, one virtual machine is running. All these servers are connected to the same VM Repository which contains images of the virtual machines and a configuration file for each virtual machine.

Physical Servers are the machines that run OracleVM Servers, a dedicated operating system, Linux/Xen based, on which OracleVM Agent is automatically installed. OracleVM Servers run virtual machines, the installed OracleVM Agent communicates with the OVM Manager.

NAS Servers are the file servers, reachable via network by the Physical Servers on which we store the virtual machine hardware configuration and the virtual machine images. Physical Servers mount the OVS folder shared by all the servers. This is the VM Repository.

Virtual Machines run (CPU) on the Physical Servers and are stored in the NAS Servers (images). The virtual hardware configuration of a virtual machine, e.g. memory size, processor number, network interfaces specification, is written in a dedicated configuration file stored in the NAS. Typically, the same folder in the /OVS is used for the configuration file and the data image. The virtual software configuration is provided by Quattor and is fully transparent. No special configuration for a virtual machine profile is needed. A virtual machine can be installed via Quattor like a physical one.

Manager Machine is an external machine running Oracle HTTP Server and OracleVM Manager, a web application, that communicates with OracleVM Manager on the servers. If Java is installed, a console via Java Applet is also available for remote control. Also, a Command-Line Interface can be installed.

Server Pools are logical groups of Physical Servers running OVM Servers. Server Pools share the same VM Repository. Virtual Machines in the same Server Pools can run on different Physical Servers, live migrate, load balance, fault tolerant live migrate etc. Server Pools can be easily managed by the OracleVM Manager via web interface or partly by the Command-Line Interface. Each server communicates with other Server in the pool using OVS Agents.

1.2 Management

1.2.1 OracleVM Manager

Oracle VM Manager provides virtual machine life cycle management, including creating virtual machines from installation media or from templates. It provides features such as power on, power off, deleting, importing, deploying, and live migration of virtual machines. Oracle VM Manager also effectively manages resources, including ISO files, virtual machine templates, and shared virtual disks.[2] In addition to the web interface, a command-line interface can be installed which can be useful for automating tasks regarding the virtual machine life cycle management, however its functionality is limited (as for the version 2.2-9, MAC addresses cannot be specified etc.).

1.2.2 Searching a physical host of a virtual machine

In the situation without an access to the OVM Manager (e.g. OVM failure), it is complicated to find where the specified virtual machine is running. All the machines in the cluster have to be contacted via ssh and the Xen command line is used to check if the virtual machine is running on any of them. This task had been already automated with the "SearchVM" script, however its performance was not efficient when searching more virtual machines or for the use in other tools. For this reason, a new version was implemented and the performance was improved by using "fork" to have a parallel process for each

[2]'Oracle VM Manager Release Notes', *Oracle Corporation*, June 2009, <http://download.oracle.com/docs/cd/E11081_01/doc/doc.21/e10903/toc.htm> [accessed 4 August 2010]

machine in the cluster. The output is usually obtained within 1 second which is 5-10 times less compared to the previous version.

```
[lxadm04 ~]$ time ./oldsearchvm.sh dbvrtg001
dbsrvg3502
real 0m5.020s user 0m1.118s sys 0m0.187s

[lxadm04 ~]$ time ./searchvm.sh dbvrtg001
dbsrvg3502
real 0m0.823s user 0m1.106s sys 0m0.230s
```

1.2.3 Opening a VNC console

Based on the SearchVM script, two other utilities for the critical situations with no access to the OVM Manager were created. The first of them is used to access the specified virtual machine via VNC. The utility checks if X11 forwarding is enabled, then uses SearchVM to find the physical host, connects to it and opens vncviewer, built in the Xen command line.

1.2.4 Stopping a virtual machine

The second tool looks up the physical host (with SearchVM), connects to it, and shuts down the specified virtual machine using the XEN command line. It may take several seconds before the virtual machine turns off.

1.2.5 Creating a virtual machine

The creation of a virtual machine itself is done by a user in OracleVM Manager. Then, several steps for integrating the virtual machine into the infrastructure (LanDB, Quattor) must be done manually. For this reason, two possible approaches of automation were proposed:

1. The virtual machine will be created in OracleVM (either using templates, or specifying new one with PXE boot), then it will be registered in LanDB and Quattor with the generated MAC address.
2. The virtual machine will be registered in LanDB where the generated MAC address is obtained, then in Quattor and finally created in OracleVM where the MAC address must be specified. This is in principle how it had been done manually.

The current version of OVM Command-Line Interface does not allow users to specify MAC addresses, therefore the second approach would require more steps – searching the configuration file and making changes in it. As a result of this, the first approach was chosen to be the procedure that the automation script CreateVM would follow. For the virtual machine creation, we decided to create new machines rather than to create them from templates, because it offers more flexibility. Finally, CreateVM works as follows:

- The OracleVM Manager host is contacted via ssh and using CLI, the specified virtual machine is created and two network devices are added into it.
- From the output of the OVM CLI *vm nic ls* command, the generated MAC address of the newly created machine is obtained.
- The machine with management scripts is contacted via ssh and the modified Perl script for registering a new machine in LanDB using the SOAP interface is executed with two parameters: the specified name of the virtual machine and its MAC address.
- On lxadm, LEAFAddHost is executed with all necessary parameters in order to register the virtual machine in Quattor.

- The virtual machine is launched.
- After 20 minutes, PrepareInstall is run.

Whilst having this process automated, some tasks still need to be done manually (for instance, a service request for a block of new IPs for your virtual machines if needed).

1.3 Recovery

1.3.1 Standard backup approach

It is advised to backup resources which reside on the VM Servers (VM Images in the /OVS/running_pool, VM templates in /OVS/seed_pool, or ISO files in /OVS/iso_pool). Beside this, OracleVM Manager comes with a backup script placed in /opt/ovs-manager-[version]/bin. Running this script requires 4 users inputs (backup/restore selection, OVS database password, dump file path and log file path). To automate this procedure via cron, either some modifications of this script, or the *expect* unix tool are needed.

1.3.2 Standard restore approach

The provided script for backups is also used for restoring OracleVM Manager. It asks for the same user input, plus a SYS database password (to rebuild the OVM Manager Database schema).

1.3.3 Automatic backup tool

Based on the provided script, the modified version which can be run as a cron job was prepared. It uses the Oracle database exp utility to get the dump and log files. After this is done, it uploads these files (filename is generated from the current date) to NAS. In the current production state, the size of one OracleVM Manager database dump is around 1.3 MB, and hence this job is executed daily.

In addition to this, the external database connection string was found to be in: /opt/oc4j/j2ee/home/config/data-sources.xml This, for example, can be used for migrating or having multiple instances of OracleVM Manager pointing to the same external database.

1.3.4 Critical situation - OVM Emergency and Disaster Recovery

We assume a situation when either OVM backup files are not available, or the provided backup utility fails and OracleVM Manager cannot be restored with the standard approach. A new installation of OracleVM Manager and Command-Line Interface is needed and we aim to collect as much data about the previous setting as possible, and to rebuild the old structure based on the collected data and user input. OVM Emergency and Disaster Recovery is done in the following steps:

1. The list of physical hosts in the cluster is retried via CDBhost query; each host is running OVS agent (assuming it is not corrupted).
2. Using a Python RPC script, each host is asked for the master hostname and virtual machines it is running via XML-RPC.
3. Based on this, the pool structure prototypes are created.
4. User is asked to specify which of the found pools he wants to recreate and to enter their names.
5. The hosts which were included in the selected pools are contacted via ssh; their OVS agent databases are moved to backup files and OVS agents are restarted (virtual machines have to be powered off).
6. The host of the newly installed OVM Manager is contacted via ssh and Command-Line Interface is used to:

- create the specified server pools with found master servers
- add hosts into corresponding server pools
- discover VM images in the pool directories
- import found VM images
- approve imported VM images and assign VMs to their physical hosts

However as mentioned, the current version of the CLI has some limitation. Hence, the full automation of this task is not possible and some parts need to be done manually in the web interface.

1.3.5 OVM Structure Recovery - assistance tool

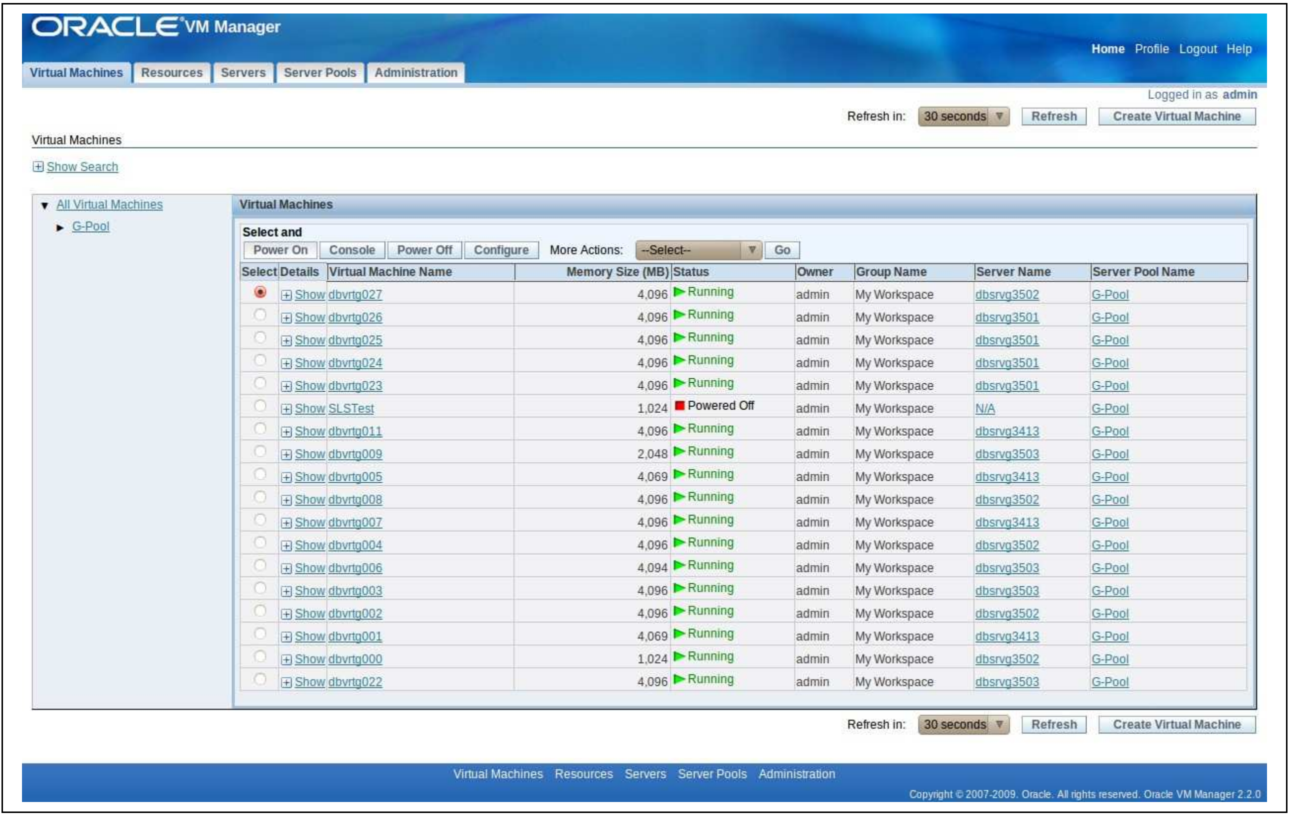
To simplify the critical situation recovery process, a tool which would help an operator with recreating the lost OVM structure was developed. OVS Agents of all physical hosts in the cluster are queried via the XML-RPC protocol, the return data is filtered and processed. Based on this, a tree structure is built. In this structure, all OracleVM servers have their master servers as parents and virtual machines they run are their children. The tree structure is displayed in the command line and stored in the Unix directory hierarchy, hence it can be easily accessed and used by some other potential recovery utilities.

Example command line output:

```
.
 |-dbsrvd249
 |---dbsrvd249
 |---dbsrvd283
 |-dbsrvg3413
 |---dbsrvg3413
 |-----13_dbvrtg001
 |-----23_dbvrtg007
 |-----27_dbvrtg005
 |-----31_dbvrtg011
 |-----42_dbvrtg025
 |---dbsrvg3502
 |-----12_dbvrtg000
 |-----15_dbvrtg002
 |-----22_dbvrtg004
 |-----26_dbvrtg008
 |-----34_SLSTest
 |-----38_dbvrtg023
 |-----44_dbvrtg026
 |---dbsrvg3503
 |-----10_dbvrtg022
 |-----18_dbvrtg003
 |-----19_dbvrtg006
 |-----29_dbvrtg009
 |-----40_dbvrtg024
 |-----46_dbvrtg027
```

1.4 Status monitoring

The individual machine status monitoring is possible in OracleVM:


ORACLE VM Manager
Home Profile Logout Help
Virtual Machines Resources Servers Server Pools Administration
Logged in as admin
Refresh in: 30 seconds Refresh Create Virtual Machine
Show Search
All Virtual Machines
G-Pool
Power On Console Power Off Configure More Actions: --Select-- Go
Virtual Machines Resources Servers Server Pools Administration

Virtual Machines

Select and

Select	Details	Virtual Machine Name	Memory Size (MB)	Status	Owner	Group Name	Server Name	Server Pool Name
●	Show	dbvrtg027	4,096	Running	admin	My Workspace	dbsrvg3502	G-Pool
○	Show	dbvrtg026	4,096	Running	admin	My Workspace	dbsrvg3501	G-Pool
○	Show	dbvrtg025	4,096	Running	admin	My Workspace	dbsrvg3501	G-Pool
○	Show	dbvrtg024	4,096	Running	admin	My Workspace	dbsrvg3501	G-Pool
○	Show	dbvrtg023	4,096	Running	admin	My Workspace	dbsrvg3501	G-Pool
○	Show	SLSTest	1,024	Powered Off	admin	My Workspace	N/A	G-Pool
○	Show	dbvrtg011	4,096	Running	admin	My Workspace	dbsrvg3413	G-Pool
○	Show	dbvrtg009	2,048	Running	admin	My Workspace	dbsrvg3503	G-Pool
○	Show	dbvrtg005	4,069	Running	admin	My Workspace	dbsrvg3413	G-Pool
○	Show	dbvrtg008	4,096	Running	admin	My Workspace	dbsrvg3502	G-Pool
○	Show	dbvrtg007	4,096	Running	admin	My Workspace	dbsrvg3413	G-Pool
○	Show	dbvrtg004	4,096	Running	admin	My Workspace	dbsrvg3502	G-Pool
○	Show	dbvrtg006	4,094	Running	admin	My Workspace	dbsrvg3503	G-Pool
○	Show	dbvrtg003	4,096	Running	admin	My Workspace	dbsrvg3503	G-Pool
○	Show	dbvrtg002	4,096	Running	admin	My Workspace	dbsrvg3502	G-Pool
○	Show	dbvrtg001	4,069	Running	admin	My Workspace	dbsrvg3413	G-Pool
○	Show	dbvrtg000	1,024	Running	admin	My Workspace	dbsrvg3502	G-Pool
○	Show	dbvrtg022	4,096	Running	admin	My Workspace	dbsrvg3503	G-Pool

Copyright © 2007-2009. Oracle. All rights reserved. Oracle VM Manager 2.2.0

Screenshot – OracleVM Manager
Figure 2

For the overall statistics, we decided to use Service Level Status (SLS). Two XML files are exported. One contains a ratio of the number of running virtual machines to the total number of virtual machines, the other contains the same ratio, but for the OracleVM servers. From these two values, the harmonic mean which is most appropriate, because it tends to mitigate the impact of large outliers and aggravate the impact of small ones[3], is counted. This gives a quick overview of the OracleVM service availability.

1.5 Summary

In this part, the tool-kit for managing virtual machines (SearchVM, ConsoleVM and StopVM) which does not depend on the OracleVM functionality was developed and in order to reduce manual steps required for deploying a virtual machine in the current infrastructure (LanDB, Quattor) altogether with OracleVM, CreateVM automates most of it. Then, two scenarios were examined for recovering: one for standard situations and one for critical situations. In the first scenario, autobackup tool was derived from the OracleVM Manager backup script. In the second one, the assistance tool for rebuilding the server pool structure was implemented. For the overall status monitoring, two scripts for exporting XML according to SLS specifications were written.

[3]Wikipedia contributors, 'Harmonic mean', *Wikipedia*, *The Free Encyclopedia*, 6 June 2010, 16:21 UTC, <http://en.wikipedia.org/w/index.php?title=Harmonic_mean&oldid=366388572> [accessed 5 August 2010]

2 Optimization

2.1 Motivation

Virtualization is a commonly used for consolidation of hardware resources which are shared by all guests and used depending on their needs. Physical memory, however, is an exception, because unlike CPU or I/O cards, operating systems generally tend to utilize their physical memory fully. Apart from kernel/user code and data, some memory is allocated for page caching. Since hard drives are slow, this can dramatically improve performance. However, as the whole memory is utilized, this is becoming a bottleneck in virtualized systems. For instance, a 4GB physical host is able to manage three 1GB virtual machines, but as its hypervisor uses some memory, more than three guests cannot be normally added.

2.2 Memory overcommitment

This problem can be solved by using one of the memory overcommitment techniques. Currently, there are several proposed mechanisms for it[4]:

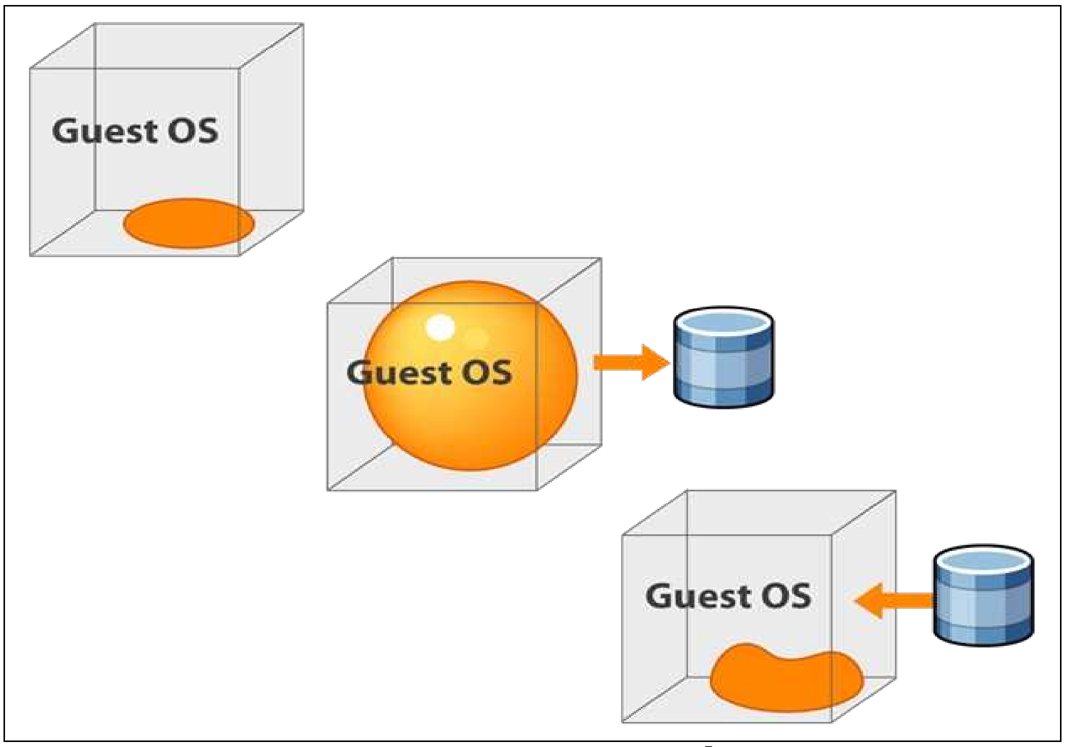
- *Ballooning* utilizes a special driver that “inflates” by requesting memory from the kernel and returning it to the Virtual Machine Monitor (*VMM*).
- *Content-based page sharing* transparently discovers identical pages and combines them, using copy-on-write to separate them again when necessary.
- In *demand paging*, the VMM maintains a swap area and can page out memory without guest involvement. Because of the so-called “semantic gap”, a VMM-based page replacement algorithm cannot be efficient and can lead to double paging.
- *Hot-plug memory add*[5] works as if a physical DIMM were added. The firmware, ACPI or pHYP for example, tells the OS a new address range of memory is available. After Linux finds the new memory it sets up a new mem_map[] and other structures. Finally the kernel then adds the new memory into the allocator, making it available for use.
- In *ticketed ballooning*, a ticket is obtained when a page is surrendered to the balloon driver. Original page can be retrieved if Xen has not given the page to another domain.
- *Swapping out entire* (idle/low-priority) *guests* - guests are then swapped in when triggered by a mechanism similar to wake-on-LAN.
- *Transcendent Memory*[6] is both: (a) a collection of idle physical memory in a system and (b) an API for providing indirect access to that memory. A tmem host (such as a hypervisor in a virtualized system) maintains and manages one or more tmem pools of physical memory. One or more tmem clients (such as a guest OS in a virtualized system) can access this memory only indirectly via a well-defined tmem API which imposes a carefully-crafted set of rules and restrictions.

  Because *ballooning* is commonly implemented nowadays and supported by both Xen and OracleVM Server, we chose it for our optimization test purposes.

[4]Dan Magenheimer, 'Memory Overcommit… without the commitment', *Xen Summit Boston 2008*, 24 June 2008, <http://wiki.xensource.com/xenwiki/Open_Topics_For_Discussion?action=AttachFile&do=get&target=Memory+Overcommit.pdf> [accessed 5 August 2010]
[5]J. Schopp et al, 'Resizing Memory with Balloons and Hotplug', *Ottawa Linux Symposium 2006*, 22 July 2006, <http://www.kernel.org/doc/ols/2006/ols2006v2-pages-313-320.pdf> [accessed 5 August 2010]
[6]Dan Magenheimer, 'Project: Transcendent Memory', *Oracle Corp.*, 22 July 2010, <http://oss.oracle.com/projects/tmem/> [accessed 5 August 2010]

2.3 Ballooning[4]

A balloon driver, when properly managed, complements nicely with the Linux kernel’s page replacement mechanisms. When the balloon is inflated by one page, the kernel surrenders the page that it believes is least likely to be used again. If the balloon can be inflated until just before the guest “hurts” for memory, the guest’s idle memory will be minimized, and Xen can use that otherwise idle memory for another guest. When a guest needs more memory, it must be able to quickly deflate the balloon; but if additional memory is unavailable, a properly configured Linux guest swaps pages to disk, just as if it reached a fixed limit of physical memory. Then when memory later becomes available, Linux can make use of that memory to swap pages in from disk if it needs to.



Memory ballooning on guests[7]
Figure 3

2.3.1 *dom0 auto-ballooning*

The balloon driver has been in Xen for 2 years and it is mostly used for dom0 auto-ballooning. At first, you allocate the dedicated memory. This is done by specifying "dom0_mem=512M" option for Xen hypervisor (usually xen.gz) in grub.conf/menu.lst[8]. This makes sure the initial size of memory allocated for dom0 is 512 MB (replace with the amount of memory you want), and the rest of the RAM is available for other guests in Xen hypervisor. The dom0 memory does not change unless in /etc/xen/xend-config.sxp, the "dom0-min-mem" option is set up to a smaller value (in this case, for instance, 192) and also the "enable-dom0-ballooning" option is configured to “yes”. Then, when guests need more memory, dom0 tries to reduce its allocated memory and make the released memory available to them.

[4]Dan Magenheimer, 'Memory Overcommit… without the commitment', *Xen Summit Boston 2008*, 24 June 2008, <http://wiki.xensource.com/xenwiki/Open_Topics_For_Discussion?action=AttachFile&do=get&target=Memory+Overcommit.pdf> [accessed 5 August 2010]
[7]Eric Horschman, 'Cheap Hypervisors: A Fine Idea -- If You Can Afford Them', *VMware: Virtual Reality*, 11 March 2008. <http://blogs.vmware.com/virtualreality/2008/03/cheap-hyperviso.html> [accessed 9 August 2010]
[8]Pasi Karkkainen, 'Best Practices for Xen', *Xen Wiki*, 13 May 2010. <http://wiki.xensource.com/xenwiki/XenBestPractices> [accessed 9 August 2010]

2.3.2 domU self-ballooning

The balloon driver needs to be installed either in-kernel (paravirtualized) or as a kernel module (hardware virtualized) on guests. Then, their memory can be set to from as low as possible (it can cause an out-of-memory condition) to the max value which is specified when they are created. For automatic changes in memory allocation (the active guests take available memory from the idle ones), the xenballoond deamon has to be present in them. The changes in the guest memory size are immediate, dom0 does not manage them and is not anyhow involved in them. For control and statistics of domU balloon sizes from the dom0 perspective, the directed ballooning can be used (several xenstore tools have to be installed on guests if this mechanism is desired).

2.4 Memory ballooning tests

2.4.1 "Plain" images

At first, we tested the dom0 auto-ballooning by adding dummy guests without a balloon driver, so the maximum number of machines was determined by the memory they were allocated during their creation. Once the number of machines reaches its limit, dom0 tries to reduce its memory and release it to guests.

2.4.2 Balloon driver

The functionality of a balloon driver was tested on machines created from the Oracle Enterprise Linux 5 - PV OVM template. Using the *xm mem-set* command, the smaller amount of memory can be allocated to a guest after its creation and this increases the total free memory.

2.4.3 Xenballond deamon - "clean" image

The mentioned OVM template image was modified by installing the xenballoond deamon on it. This enabled automatic memory allocation management on these idle, "clean", virtual machines. When needed, they lowered their memory down to the *Committed_AS* value (in /proc/meminfo) as described in [4]. We tested if this self-ballooning would work and the "clean" virtual machines would release as much as possible memory to the "stress" ones whose image containment is described below.

2.4.4 "Stress" image

Based on this "clean" image, we prepared the "stress" one which was highly used in our stress tests. It had a stress script (inspired by "eatmem" described in [4]) in rc.local, so once a "stress" virtual machine booted up, it started this script. The script allocates 1 GB of memory (using the malloc function) in 4 threads, writes up to 512MB chunks of random data into it, frees it and then sleeps for a random number of seconds (up to 63). This sequence runs forever, hence from the memory perspective, these virtual machines were heavily used.

2.4.5 Specifications

Both images had the size around 15 GB, so they were copied to a folder mounted via NFS (initially, 50 "stress" images were placed there and 2 "clean" ones). In their configuration files, they were assigned 2 virtual CPUs and 4 GB RAM. They were all allocated swap space of 6 GB (2 GB for the normal use + 4 GB in case of hot swapping) in their images. The testing machine was dbsrvg3501 (Intel Xeon CPU L5520 @ 2.27GHz – 8 cores, 48 GB RAM) running on OracleVM 2.2.0 (kernel 2.6.18-128.2.1.4.9.el5xen) with the Xen 3.4.0 hypervisor. It was running one domain based on a "plain" image (Red Hat 5), dbvrtg000. In the beginning. the dom0 was allocated 512 MB which could be reduced to 196 MB by auto-ballooning.

2.4.6 Measurements

In the first test, the virtual machines were launched sequentially in groups (1-7 machines) and for each group, the total free memory in Xen was measured 10 times with a 30-second delay inbetween.

This contained all three VM variants. Based on the results from this test, the seconds test was set up. Up to the previously measured optimal number of domains, the “stress” machines were automatically launched one by one and the total free memory was measured for each one in similar fashion to the first test.

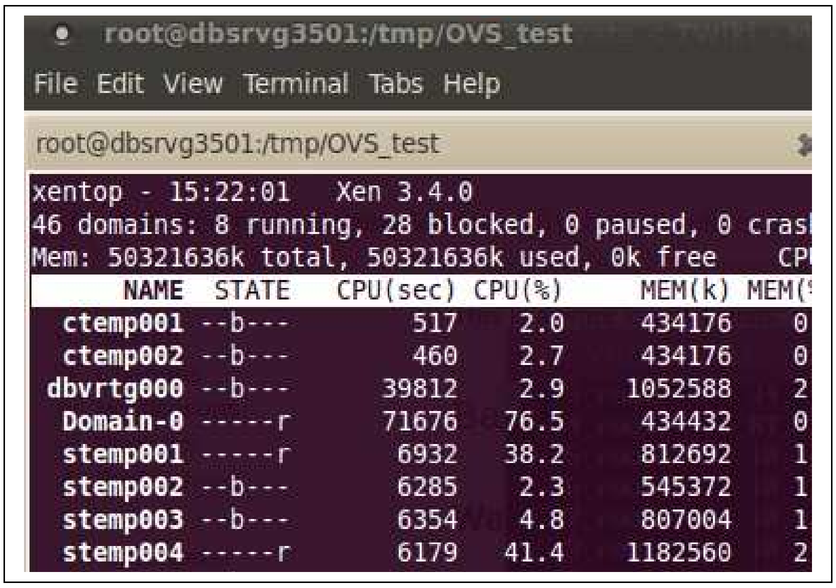
2.5 Results

2.5.1 Test 1

The “plain” virtual machine, dbvrtg000, was running without any problems and its memory was untouched by the others. The balloon driver in the “clean” VMs inflated and reduced their memory to around 400 MB. With the “stress” VMs, the behaviour was as follows:

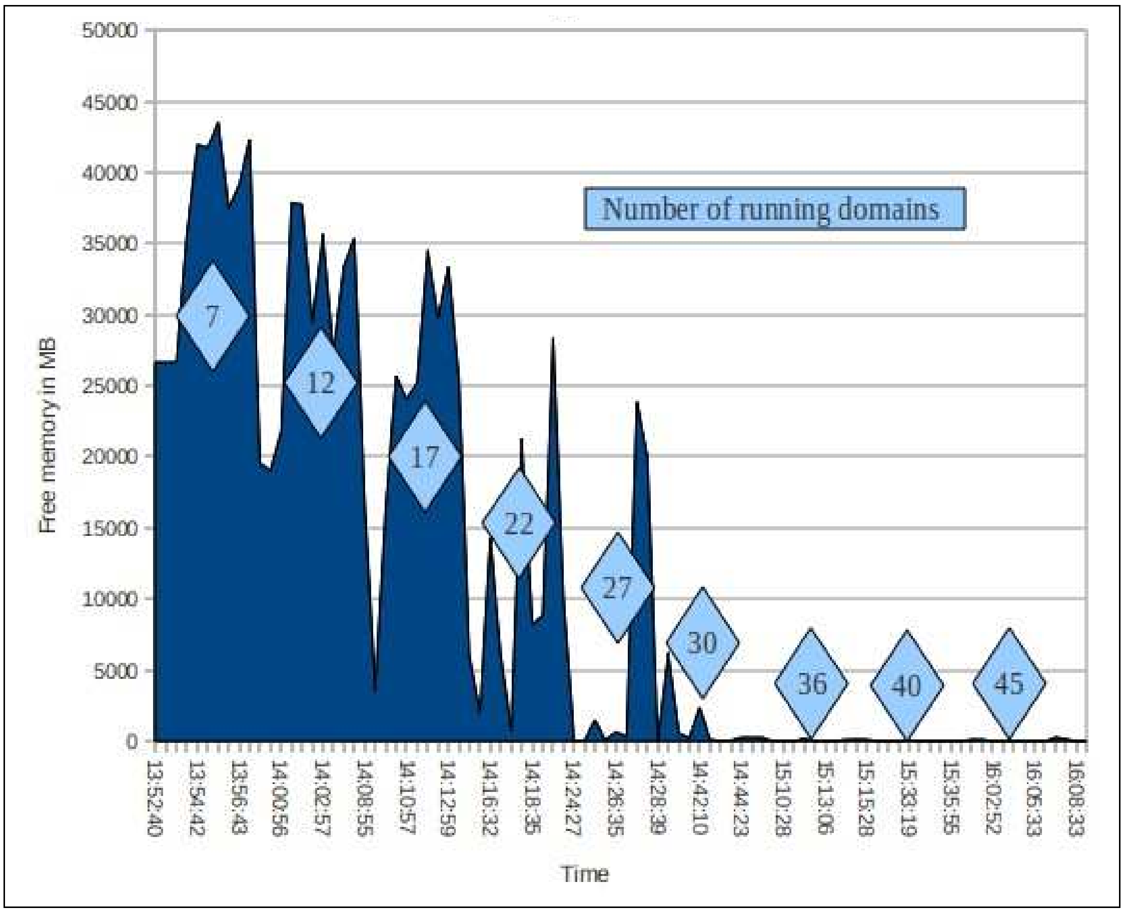
Number of running domains	Event description
Up to 27	Creation of new virtual machines without any problems, machines were running smoothly, dynamically changing their memory size
27	dom0 started auto-ballooning in order to enable new guests creation
30	The memory of dom0 was minimal
33-45	Problems with creating new virtual machines arose – during the creation time, some machines had to be paused in order to stop the full memory usage; minor swapping on guests
45	“Hot swapping” on guests

This behaviour pattern can be seen in Figure 5 which is a timeplot of free memory. It is based on the raw data, hence there are instantaneous drops in free memory during the domain creation time. As described in the table, there is a short peak when 27 domains were running because of the memory release from dom0.


root@dbsrvg3501:/tmp/OVS_test
File Edit View Terminal Tabs Help
root@dbsrvg3501:/tmp/OVS_test

```
xentop - 15:22:01   Xen 3.4.0
46 domains: 8 running, 28 blocked, 0 paused, 0 cras
Mem: 50321636k total, 50321636k used, 0k free     CP
      NAME  STATE   CPU(sec) CPU(%)     MEM(k) MEM(
  ctemp001 --b---        517    2.0     434176    0
  ctemp002 --b---        460    2.7     434176    0
 dbvrtg000 --b---      39812    2.9    1052588    2
  Domain-0 -----r      71676   76.5     434432    0
  stemp001 -----r       6932   38.2     812692    1
  stemp002 --b---       6285    2.3     545372    1
  stemp003 --b---       6354    4.8     807004    1
  stemp004 -----r       6179   41.4    1182560    2
```

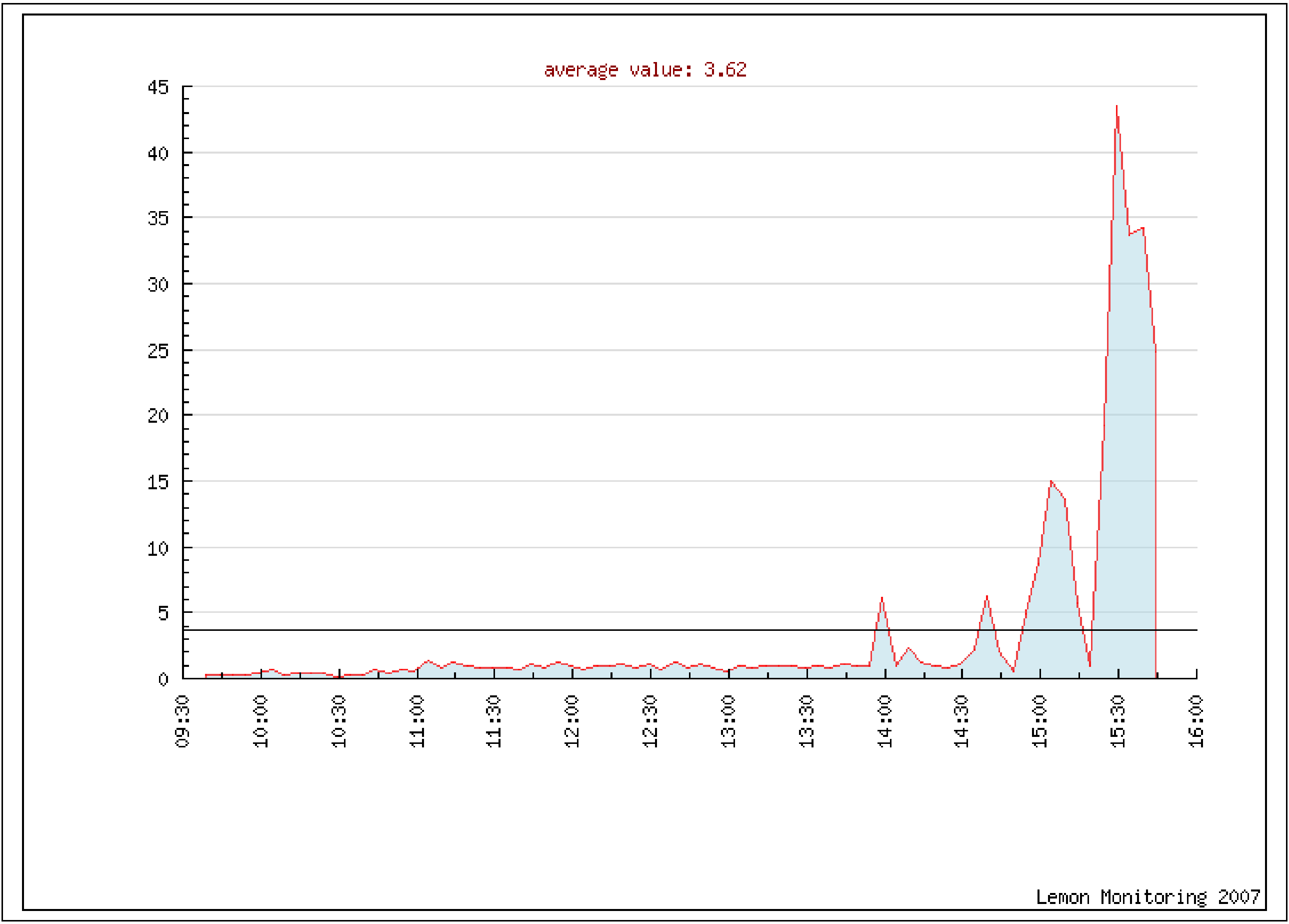
Screenshot – xen with 46 domains
Figure 4



Graph from the 1st test – free memory (in MB) against time
Figure 5

2.5.2 CPU usage

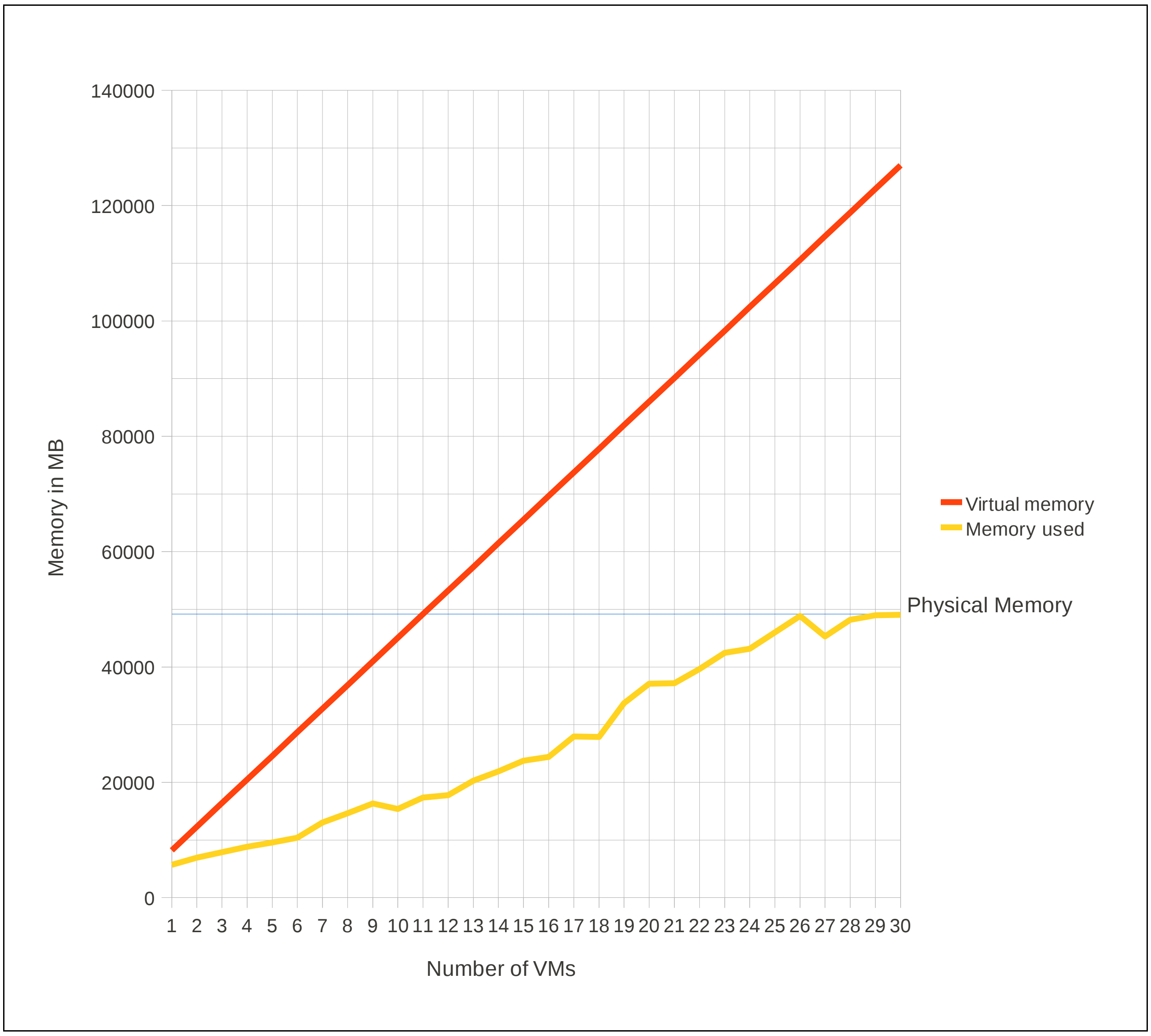
This was mainly a memory stress test, therefore the CPU had been idle for most of the time. As shown in Figure 6 (which is one of statistics plot for the physical host on Lemon Web), the utilization stayed under 5 % for almost the whole test, except of the time when guests started hot swapping where the CPU usage jumped to around 40 %.



CPU usage (in percentage) against time from the Lemon Web monitoring
Figure 6

2.5.3 Test 2

From the results of the first test, we estimated the limiting number of "stress" virtual machines to be 30. The second test ran up to this number with more precise measurement: the total free memory was measured 10 times after creation of every single VM and there was an additional 5-minute delay between the measurement and the VM creation in order to let the VMs properly boot up and start the stress script. From the raw data, the average of the total free memory for each state of running machines was counted and subtracted from the total amount of physical memory to get the size of used memory. This is represented as the yellow curve in Figure 7. The blue constant line shows the total physical memory and the yellow curve is gradually approaching to this value which is what we expected according to our previous estimates from the first test. The above described phenomenon of dom0 auto-ballooning at 27 virtual machines can be also seen on this curve. The orange linear line stands for the total amount of memory virtually allocated to guests. This would correspond to the used memory when no memory overcommitment mechanism is used and the intersection with the blue line would be the maximal number of virtual machine that could be created on this physical host. This point is at 11 machines, as expected: 48 GB could fit twelve 4GB virtual machines, but dom0 takes some memory, so only 11 guests would be added. On the contrary, the yellow curve continues far beyond this point.

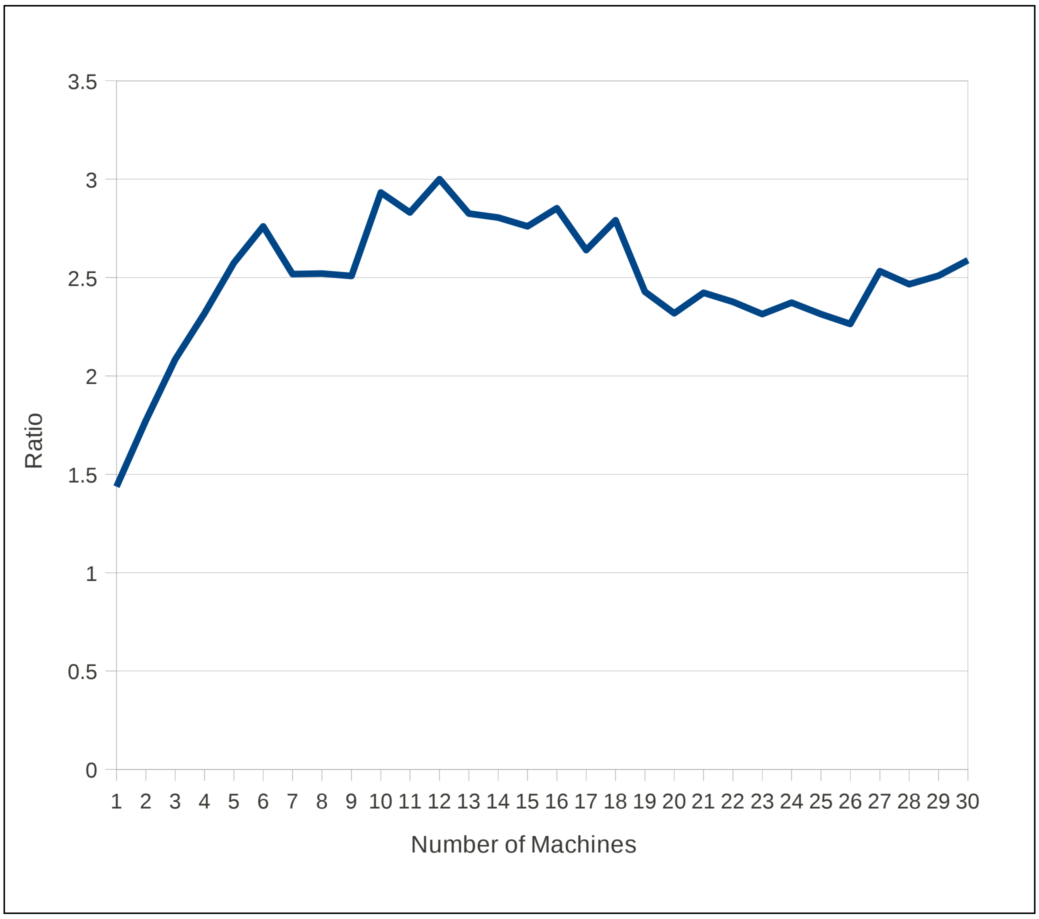


Comparison of used memory with the total memory allocated on guests
Figure 7

2.5.4 Number of machines

In these test conditions, we could run up to 30 “stress” virtual machines on a single physical host. This is more than two times what is normally possible without using any memory overcommitment techniques and shown in Figure 8 as a ratio between the used memory and the total memory allocated on guests (the yellow curve and the orange line in Figure 7). The average value of this ratio is 2.5.

As for the “clean” virtual machines, it is possible to create 120 of them, because each idle machine uses only around 400 MB. In this case, it would be more than 10 times the number of the “plain” virtual machine we could create.



Ratio between the used memory and "virtual memory"
Figure 8

2.6 Summary

Several memory overcommitment techniques were overviewed and *ballooning* was chosen for our test purposes. Three virtual machine images were prepared ("plain", "clean" and "stress"). The "stress" one loaded a script which simulated heavy memory usage at start-up. Two tests were done. In the first one, the overall behaviour and ballooning features were examined and the number of running virtual machines was pushed to the limit when the machines started hot-swapping. Based on the results, the estimates were made in order to prepare the second test which verified what we had previously measured and gave the detailed background of the region with the stable average amount of the total free memory. The CPU was mostly underutilized throughout these memory tests. We found out that we could create up to 30 virtual machines with heavy memory usage and up to 120 idle virtual machines in these test specifications. This is more than two times, respectively ten times what would possible without any memory overcommitment mechanisms.

3 Conclusion

In the first part, several virtual machine management tools were developed. Currently, they are distributed via syscontrol and used by various users. Even though the number of the steps required for deploying a new virtual machine into the current infrastructure was reduced, some manual work remains. The potential extension of the developed script which would assist the operator and speed up this process even more might be examined. Regarding recovery, the multiply instances of OracleVM Manager would increase redundancy of this virtualization system and the semi-automatic recovery tool could be based on the output from the implemented assistance tool. The SLS services have IDs DBVirt_pm for the physical machines' availability monitoring and DBVirt_vm for the virtual machines.

In the second part, two tests for determining the optimal number of running machines per one physical host were conducted. In these test specifications, we can create up to 30 heavily used (from the memory perspective) virtual machines, hence the optimal number of running machines would be between 20 and 25 in order to have more memory dedicated to the privileged domain which is running the Xen hypervisor and to have some memory reserves for the situations when more guests would require larger amounts of memory at the same time. As for the mostly idle virtual machine, we could create up to 120 of them, and so the optimal number would be between 80 and 100 for the same reason. However, 80 machines on one physical host would be complicated to maintain, therefore these numbers are rather giving the idea of the physical machine potential than the recommended approach. This is partly the reason why the current virtual machines in production do not use any memory overcommitment mechanisms.

For the future needs, it might be useful to consider some other possible tests which could be done for declaring the possible approaches of better hardware utilization. Namely, the memory stress script could be replaced by a CPU stress application, or a general usage simulation script (i.e. infinite Linux kernel compilation). In addition to this, the directed ballooning technique could be used for the overall guest monitoring, including their balloon sizes. Apart from this, a custom image with more compact configuration (so, it takes less than 400 MB which was the minimal virtual machine memory in our tests), possibly using the hardware virtualization, could be prepared. Additionally, other memory overcommitment techniques (for instance, transcendent memory) may be examined.

4 Acknowledgements

In the first place, I would like to express thanks to all people in the IT-DB-IMS section, especially to my supervisors, Luigi Gallerani and Giacomo Tenaglia, Artur Wiecek, Dmitry Ustyushkin. Paul Smith and Sophie Lemaître for any help with accomplishing my tasks. Also, I would like to thank other people in the IT-DB group for creating an environment in which the people cooperated and it was easy to find help outside my section. Then, my thanks is deserved by all people involved in CERN openlab, especially the ones preparing this student programme which gave me a lot of useful experiences. Last but not least, thanks belong to my friends and other summer students who I discuss my problems with or who gave me a hand with something, namely Peter Englund and Wenjing Wu, and of course my family for their support.

5 Bibliography

[1] Stephen Spector, 'New to Xen Guide', *xen.org*, 15 July 2010, <http://www.xen.org/files/Marketing/NewtoXenGuide.pdf> [accessed 4 August 2010]

[2] 'Oracle VM Manager Release Notes', *Oracle Corporation*, June 2009, <http://download.oracle.com/docs/cd/E11081_01/doc/doc.21/e10903/toc.htm> [accessed 4 August 2010]

[3] Wikipedia contributors, 'Harmonic mean', *Wikipedia, The Free Encyclopedia,* 6 June 2010, 16:21 UTC, <http://en.wikipedia.org/w/index.php?title=Harmonic_mean&oldid=366388572> [accessed 5 August 2010]

[4] Dan Magenheimer, 'Memory Overcommit… without the commitment', *Oracle Corp. (Xen Summit Boston 2008)*, 24 June 2008, <http://wiki.xensource.com/xenwiki/Open_Topics_For_Discussion?action=AttachFile&do=get&target=Memory+Overcommit.pdf> [accessed 5 August 2010]

[5] J. Schopp et al, 'Resizing Memory with Balloons and Hotplug', *Ottawa Linux Symposium 2006*, 22 July 2006, <http://www.kernel.org/doc/ols/2006/ols2006v2-pages-313-320.pdf> [accessed 5 August 2010]

[6] Dan Magenheimer, 'Project: Transcendent Memory', *Oracle Corp.*, 22 July 2010, <http://oss.oracle.com/projects/tmem/> [accessed 5 August 2010]

[7] Eric Horschman, 'Cheap Hypervisors: A Fine Idea -- If You Can Afford Them', *VMware: Virtual Reality*, 11 March 2008. <http://blogs.vmware.com/virtualreality/2008/03/cheap-hyperviso.html> [accessed 9 August 2010]

[8] Pasi Karkkainen, 'Best Practices for Xen', *Citrix Systems, Inc*, 13 May 2010. <http://wiki.xensource.com/xenwiki/XenBestPractices> [accessed 9 August 2010]