Estimating Working Set Size by Guest OS Performance Counters Means

Anna Melekhova
Parallels
Moscow, Russia
email: annam@parallels.com

Larisa Markeeva
Innopolis University
Kazan, Russia
email: l.markeeva@innopolis.ru

***Abstract*—Cloud infrastructures imply virtual machines dynamic hosting. The distribution of resources is performed in dependence on the volume and a pattern of used resources. An estimation of current load and prediction of future load are mandatory for effective resources management. The statistical processing of internal counters of a guest operating system gives good prospects. The disadvantage of the method is the complexity of data collection and its processing. The large number of parameters and variance in their behavior in different operating systems and configurations introduce extra complexity for the virtualization case. The present research covers the estimation of a virtual machine working set based on guest OS internal counters. The correction of the estimated value is done in accordance on the feedback of donor guest OS.**

Keywords-virtual machine; cloud computing; memory management; working set.

I. INTRODUCTION

Resources virtualization is one of the key topics in IT (Information Technology) industry due to resource optimization task. Most of the nowadays workloads on user resources come in peaks. That is, virtual machines mostly consume little resources; but, eventually, activity gets to a peak and the volume of required resources increases significantly. In this case, resources exhausting would bring a considerable performance loss. Virtualization is a popular solution to handle the issue and make the management more flexible. A widely used practice that increases the percentage of resources utilization is setting the size of virtual resources above the real hardware resources. This method is called overcommit or oversubscription. To make this mode effective a smart resources management is required. This management follows the principle "from each according to his ability, to each according to his needs" when resources are assigned basing on the real consumption level, not the assigned one.

CPU (Central Processing Unit) and memory are the most common overcommitted resources as their utilization is not too high in usual workloads [13][14]. Modern operating systems (OS) behave differently when these resources are not fully utilized. For instance, when OS does not need CPU resource, it sends a halt signal (hlt instruction) to CPU to reduce power consumption. A virtualization system handles this event and thus, it determines that the CPU resource can be diverted to another consumer. At the same time, there is no instruction to signal about unused memory in Intel x86 architecture. Thus, it is impossible to estimate real memory consumption by hardware means.

We define a working set as a set of memory pages used by a consumer (a process or a virtual machine) in a given time frame [1]. The problem of virtual set size estimation is not new; it is discussed in [2]-[6]. There is an analogy with a process in an operating system. A virtual machine does not know that the address space is not continuous, similar to how a process does not know that its allocated space is not continuous. Accesses to unmapped memory are handled by a virtualization engine transparently to the guest operating system, similar to how a page miss is handled transparently to a process. Processes, usually, do not inform operating systems about the required volume of memory. Operating systems do not inform the virtual machine about the required volume of memory as well.

Although a working set topic is elaborately investigated (the fundamental work of Denning dates back to 1970 [7]), the virtualization adds a new dimension. While a process memory is definitely a black box for an operating system, a virtual machine can use a paravirtualization to obtain information on the guest resources management or even improve guest OS for better interaction with the virtualization module. The present article unveils a method that reflects changes inside a guest OS by the means of internal counters statistics. This statistics can be used to predict peak loads as well.

There is a number of techniques to optimize RAM utilization:

- Content-based page sharing
- Ballooning
- Memory compression
- Page replacement algorithms

The detailed description of these techniques can be found at [13][14]. Section II.A of the present article describes ballooning in more details.

Each technique has its advantages and disadvantages, while most virtualization products use a combination of them [13] [14]. A content-based page sharing has an imperceptible performance impact on the host, but its memory utilization ability depends on the workload type. Memory compression decreases the cost of a page miss, but it doesn't influence the number of misses. Page replacement algorithms often don't satisfy an acceptable page miss rate due to so-called semantic gap. Ballooning dramatically decreases the page miss rate but in the case of overinflation it causes the guest OS lags and even falls.

 ISBN: 978-1-61208-388-9

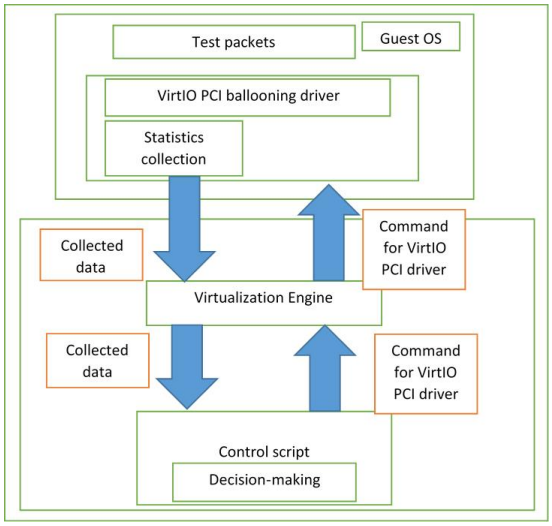

Figure 1. Balloon driver control diagram

The ballooning technique is implemented in all virtualization products – VMWare, Xen, KVM [15], Parallels, Microsoft Hyper-V, Sun VirtualBox Guest ballooning driver is designed for Linux and Windows OSes. But all known balloon implementation has no idea on the best balloon size – they just process external inflate/deflate commands [16].

Luiz Capitulino discusses auto-ballooning in [21]. The proposed concept fits Linux hosts only. The idea is to use three memory pressure types, LOW, MEDIUM, and HIGH and use these states to automatically decide whether to inflate or deflate the balloon.

Another technology used for autoballooning on KVM is Memory Overcommitment Manager (MOM) [22][23]. It also relies on the memory pressure state, same as for the previous technique. In contrast with the previous technique, this uses a set of scripts and the libvirt library [24], thus it does not require kernel modifications. But, MOM works only with Linux guests. A more detailed description of the MOM technique can be found in [23].

Both listed approaches share the same problems: the lack of adequate performance test sets [21] and the non-predictive nature of the techniques [21]-[23]. The latter means that these techniques are unable to forecast future loads and prepare the environment for such loads.

In this paper, we aim to build an effective working set size estimator and use it for an automatic balloon control algorithm. We present performance results for our algorithm as well.

Section II introduces basic definitions such as virtual machine, hypervisor, ballooning and internal counters. Section III describes the architecture of the data collecting subsystem, the data collecting itself, an efficiency test and its configuration. Section IV covers data collection required for the analysis, and introduces the estimation of a working set size. Section V presents performance testing results and methods for performance improvements. Section VI, as a conclusion, lists briefly the obtained results, and proposes further research topics.

II. BASIC DIFINITIONS

Virtual machine (VM) is an emulation of a particular computer system.

A hypervisor is a software, firmware, or hardware that creates and runs virtual machines. We use terms virtualization module, virtualization engine, engine to denote a hypervisor.

A computer that runs a hypervisor, which maintains one or more virtual machines, is called a host machine.

Each virtual machine is called a guest machine.

A. Ballooning

A ballooning is a technique of on-the-fly virtual machine random access memory size modification. Each guest OS gets an additional driver (balloon). The driver is managed by a virtualization engine. The engine issues commands of two types: increase the number of pages allocated by the driver (inflate balloon) or decrease this number (deflate balloon). The memory pages allocated by the driver are committed from the point of view of the guest OS and thus, they could not be allocated to other processes. The hypervisor on the host side gives the memory pages allocated by the balloon driver to other virtual machines running on the same host machine. This implements the principle “from each according to his ability, to each according to his needs”.

B. Operating system internal counters

Further, we present a description of operating system internal counters and how to use these counters.

Operating systems of Microsoft Windows family use system internal counters such as TotalMemory [8] and CommitMemory for computer resources management and statistics collection. The system updates these counters automatically. User space and kernel space programs can fetch these counters at any time with proper requests. The requests to get values of the counters are fast as usually the data is simply copied from the kernel memory to the specified area.

As shown in [9][10], even different versions of Microsoft Windows significantly vary in the policy of resources management. So, the existence of a single efficient resource management policy for the different OSes is unlikely.

This article investigates counters of two OSes within a single family: Windows 7 and Windows 8. This choice is based on the popularity of Microsoft's operating systems family in general [17] and specifically Windows 7 and Windows 8 that still have a mainstream support [18].

The approach illustrated on Figure 1 can be applied to other operating systems such as Linux. However, the differences in system API (Application programming interface), counters, and resource management algorithms [19][20] make modifications to the VirtIO PCI

 ISBN: 978-1-61208-388-9

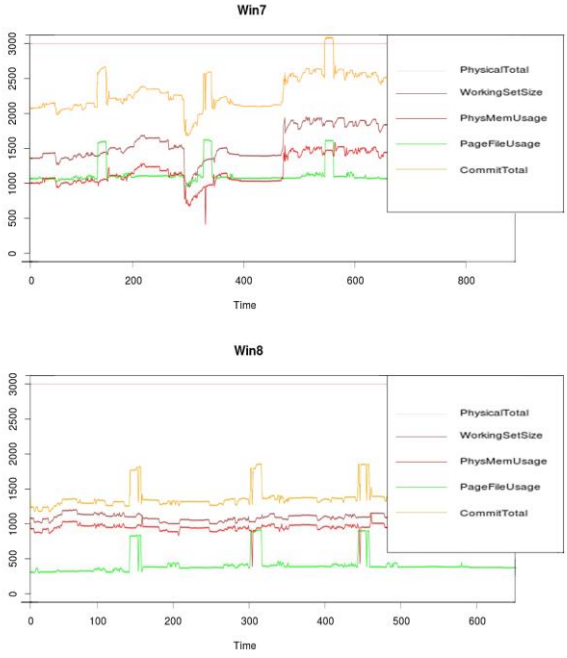

Figure 2. Internal counters dynamics for Windows 7 and Windows 8 operating systems.

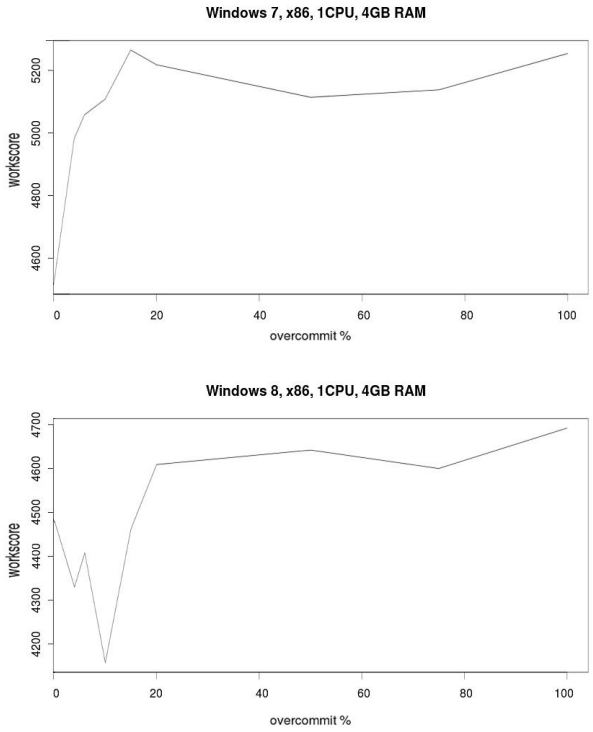

Figure 3. PCMart test results for different σ values in Windows 7 and Windows 8 operating systems.

ballooning driver necessary as well as a search for counters similar to TotalMemory and CommitMemory. The analysis of memory management and swapping algorithms in Linux is another topic for further research.

We deduce formulas (1)-(4) for the estimate of the working set size and test the efficiency of these estimates. The resulting value is applied by balloon means. That is we:

1) Estimate the value with guest OS counters.

2) Set balloon size to take away all potentially unused memory pages.

3) Correct the value in accordance to guest OS counters value.

III. THE DATA COLLECTING

A. An architecture of the data collecting subsystem

In order to minimize the cost of data acquisition, prototyping, and controlling the balloon driver we use the scheme from the Figure 1.

We have modified the VirtIO balloon driver so that in addition to the main goal the driver gets internal counters' values and delivers them to a hypervisor. The hypervisor sends this data to the Control script that analyzes the data and makes decision. Control script can send commands to the VirtIO balloon driver through the hypervisor interfaces.

B. Data collecting and efficiency test

The dependencies between internal counters and the working set size are deduced using special tests that can put virtual machines under specific workloads. These tests should be able to inflict different patterns of workload and measure the overall virtual machine's performance. The main point is that improvements in memory management shouldn't downgrade the performance.

We have used PCMark [11] that gives an aggregated view and atomics that test separate functions of virtual machine. The atomics tests are describe in Table I. The atomics tests measure system performance in a wide variety of workloads: CPU workload, network, HDD, Java programs, 2D and 3D graphics, and so on.

C. Test configuration

Host machine has the following configuration: Intel Core i5-4570 3.2GHz x 4, 16Gb RAM, 1Tb HDD, Intel Haswell Desktop, Linux Ubuntu 13.04.

Virtual machines have the following configuration: 1CPU, 4Gb RAM, 256Mb Video, 64Gb HDD. The operating systems used are Windows 7 x64 and Windows 8 x64.

Virtualization engine used is Parallels Desktop 10.

IV. VIRTUAL MACHINE WORKING SET CALCULATION

A. Collecting data for the analysis

The first modification of the VirtIO balloon driver sent 20 parameters to the hypervisor. During the experiments, some of the parameters turned out to be statistically irrelevant.

 ISBN: 978-1-61208-388-9

TABLE I. PERFORMANCE TESTS

Test	Windows 7, 1CPU, 4Gb	Windows 8, 1CPU, 4Gb
busyloop_test	-2.0%	-2.0%
system_syscall_test	-0.1%	-0.3%
process_exec_test	-2.0%	-2.3%
thread_create_test	-2.0%	-1.8%
io_hdd_seq_rand_rd _test	-99.8%	-99.2%
virtalloc_test	-1.1%	-0.2%
mem_read_test	-1.5%	-5.4%
mem_write_test	-1.6%	-2.9%
mem_pf_read_test	-0.2%	-10.0%
mem_pf_write_test	+0.4%	-9.1%
mem_copy_test	-1.2%	-1.6%

The following parameters remained:

- Physical total - the amount of actual physical memory.
- Commit total - the number of pages currently committed by the system.
- Working set size - current working set size.
- Commit available - the number of pages currently available to the system.
- Physical memory usage - the amount of physical memory currently in use.
- Page file usage - the amount of page file currently in use.
- Balloon size – the memory allocated by the VirtIO balloon driver

Figure 2 demonstrates the collected data for Microsoft Windows 7 and Microsoft Windows 8.

B. *Working set size estimation – the first approximation*

Figure 2 demonstrates that the working set size has a lower bound equal to the Physical memory usage and an upper bound equal to the Commit total.

The lack of random access memory causes memory swapping. Read/write operations become extremely slow, so underestimating the working set size leads to a significant drop in performance of the overall guest operating system. Thus, the Physical memory usage should not be estimated as its lower bound.

We denote the estimate of the working set size by W.

Evidently, the Commit Total value can not be less than the working set size as it includes the volume of used page file pages plus the volume of memory allocated in the RAM. So, it is greater than or equal o the size of a working set.

When the VirtIO balloon driver is disabled, the Commit total contains all the memory pages allocated by the system, hence

$$W=CommitTotal \quad (1)$$

TABLE II. PERFORMANCE RESULTS FOR CACHE SIZES, VARYING FROM 256MB TO 768MB. WINDOWS 7 OS

Test	Windows 7, Compared to a VM with no memory management	Windows 7, Compared to the VM with memory management without cache control
busyloop_test	+1.1%	+3.2%
system_syscall_test	-1.0%	-0.9%
process_exec_test	-3.3%	-1.4%
thread_create_test	+3.1%	+5.0%
io_hdd_seq_rand_rd _test	-99.8%	+32.0%
virtalloc_test	-0.4%	+0.7%
mem_read_test	-2.4%	-0.9%
mem_write_test	-2.4%	-0.8%
mem_pf_read_test	-0.3%	-0.0%
mem_pf_write_test	-0.1%	-0.6%
mem_copy_test	-0.9%	+0.3%

When VirtIO balloon driver is enabled, the Commit total is increased by the size of the balloon. So,

$$W=CommitTotal-BalloonSize \quad (2)$$

C. *Working set size estimation – the second approximation*

The working set size as we calculated it in (2) does not take into account the size of system caches. This raw result would drop the system performance since on disk cache exhausting the number of actual I/O operations increases. Thus, it is reasonable to provide additional memory for OS caches.

We denote the cache memory estimate by O and effective working set size as E, so

$$E=W+O \quad (3)$$

We assume that O depends on the total physical RAM of a virtual machine, hence

$$O=\sigma \cdot PhysicalTotal \quad (4)$$

According to [9][10] policies of memory managements vary, so σ is a constant defined by the operating system version.

The constant σ from (4) is evaluated as follows:

1) Modify the Control Script. In addition to the data collection it calculates the E in accordance to the formula (3).

2) The Control Script sends inflate/deflate balloon commands to the balloon driver via the hypervisor interface. The size is set to *PhysicalTotal-E* where *PhysicalTotal* is the total size of the assigned random access memory.

 ISBN: 978-1-61208-388-9

TABLE III. PERFORMANCE RESULTS FOR CACHE SIZES, VARYING FROM 256MB TO 768MB. WINDOWS 8 OS

Test	Windows 8, Compared to a VM with no memory management	Windows 8, Compared to the VM with memory management without cache control
busyloop_test	-1.6%	+0.4
system_syscall_test	+1.1%	+1.4%
process_exec_test	+2.7%	+5.1%
thread_create_test	+3.1%	+5.0%
io_hdd_seq_rand_rd _test	-98.8%	+53.1%
virtalloc_test	+0.2%	+0.4%
mem_read_test	-0.2%	+5.5%
mem_write_test	+6.1%	+9.2%
mem_pf_read_test	-4.9%	+5.7%
mem_pf_write_test	-4.0%	+5.7%
mem_copy_test	+7.7%	+9.4%

3) The guest machine runs PCMark tests with different values of σ.

A considerable decrease of performance (more than by k percent) can be observed for Windows 7 operating system when $\sigma<0.2$.

In case of Windows 8, the performance drops by more than k percent when $\sigma<0.15$.

The testing results are presented in Figure 3.

V. PERFORMANCE EVALUATION

We used the special tests described in Section III-B to test the performance of our estimation. The results are provided in Table II.

Performance changes significantly in tests: io_hdd_seq_rand_rd_test, for Windows 7 and io_hdd_seq_rand_rd_test, mem_pf_read_test, mem_pf_write_test for Windows 8.

A. Improving VM performance with the required memory size estimates

Operating systems use free memory for file caches, and it is the main cause for the performance downgrade analyzed in Section IV-C. All free memory is occupied by the VirtIO balloon driver, so no memory is left for caches. The following techniques can be used to circumvent this problem:

1) Enable LargeCacheFile to change the cache control policy [12].

2) Set the limits on the file cache using system calls.

The file cache size must be chosen individually, depending on user's needs. Tables III and IV contain performance results for cache sizes varying from 256Mb to 768Mb.

A considerable increase of performance in the case of controlled cache file size opposed to the uncontrolled cache size with the same estimate is obvious. Still, a significant drop of performance compared to a virtual machine with no memory management persists. But the resource gain on the system is significant – about 25% of memory in case of Windows 7 and about 30% of memory in case of Windows 8 were retained, assuming that the declared memory size in a VM was 4096Mb.

VI. CONCLUSION AND FUTURE WORK

We have obtained estimates on the working set size of virtual machines. These estimates were modified to include cache size in order to gain back the performance. Such an approach gave an acceptable performance while achieving the significant resource economy. Unfortunately, the current approach considers recalculation of the value for each operating system.

We propose a possible solution of this problem. One can collect lots of statistical data for different guest operating systems and build a predicting system such as an autoregressive model and/or a neural network. These systems are promising in terms of forecasting operating system working sets. Additional research in this area is required to make this idea practical, including the analysis of statistical properties of the memory utilization and internal counters time series.

As further research topics we propose improvements to the obtained estimate and a more elaborate search for dependencies between the working set size and internal counters.

ACKNOWLEDGMENT

This work has been supported by the Russian Ministry of education and science with the project "Development of new generation of cloud technologies of storage and data control with the integrated security system and the guaranteed level of access and fault tolerance" (agreement: 14.612.21.0001, ID: RFMEFI61214X0001).

REFERENCES

[1] A. Tanenbaum, Modern Operating Systems Third Edition, pp. 209 – 210, 2009.

[2] P. J. Denning, “The working set model for program behavior”, Commun. ACM 11, 5 May 1968.

[3] P. J. Denning, “Working Sets Past and Present”, IEEE Trans. Softw. Eng. 6, 1, 1980, pp. 64-84,.

[4] Weiming Zhao, Xinxin Jin, Zhenlin Wang, Xiaolin Wang, Yingwei Luo, and Xiaoming Li, “Low cost working set size tracking”. In Proceedings of the 2011 USENIX conference on USENIX annual technical conference (USENIXATC’11). USENIX Association, Berkeley, CA, USA, 2011, pp. 17-17,.

[5] Working set, [retrieved: January, 2015], from http://msdn.microsoft.com/enus/library/windows/desktop/cc441804(v=vs.85).aspx.

[6] D. R. Slutz, I. L. Traiger, “A note on the calculation of average working set size”. Commun. ACM 17, 10, 1974, pp. 563-565.

[7] P. J. Denning, “Virtual Memory”, ACM Comput. Surv. 2, 3, 1970, pp. 153-189.

[8] Performance_information structure, [retrieved: January, 2015], from http://msdn.microsoft.com/en-us/library/windows/desktop/ms684824(v=vs.85).aspx.

[9] L. B. Markeeva, A. L. Melekhova, A. G. Tormasov, "Odnorodnost' virtualizacionnih sobytij, porojdennyh razlichnymi operacionnymi sistemami", Trudi MFTI, vol. 6, N3(23), 2014, pp. 57-64 [in Russian].

[10] S. Sinofsky, Reducing runtime memory in Windows. [retrieved: January, 2015], from http://blogs.msdn.com/b/b8/archive/2011/10/07/reducing-runtime-memory-in-windows-8.aspx.

[11] PCMark. [retrieved: January, 2015], from http://www.futuremark.com/benchmarks/pcmark8.

[12] M. Friedman, O. Pentakalos, Windows 2000 Performance Guide, pp. 280-293, 2002.

[13] S. D. Lowe, Best Practices for Oversubscription of CPU, Memory and Storage in vSphere Virtual Environments, pp. 3-10.

[14] A. Melekhova, "Machine Learning in Virtualization: Estimate a Virtual Machine's Working Set Size", Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference, June 28 2013-July 03 2013.

[15] Main Page - KVM, [retrieved: January, 2015], from http://www.linux-kvm.org/page/Main_Page.

[16] Projects/auto-ballooning – KVM, [retrieved: January, 2015], from http://www.linux-kvm.org/page/Projects/auto-ballooning.

[17] Usage share of operating systems, [retrieved: January, 2015], from http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0.

[18] Windows lifecycle fact sheet, [retrieved: January, 2015], from http://windows.microsoft.com/en-us/windows/lifecycle.

[19] R. Love, Linux Kernel Development Third Edition, pp. 231-260, 2010.

[20] M. E. Russinovich, D. A. Solomon, and A. Ionescu, Windows Internals, Part 2: Covering Windows Server® 2008 R2 and Windows 7, Sixth Edition, pp. 233-402, 2012.

[21] L. Capitulino, Automatic ballooning, [retrieved: January, 2015], from http://www.linux-kvm.org/wiki/images/f/f6/Automatic-ballooning-slides.pdf.

[22] Memory Overcommitment Manager, [retrieved: January, 2015], from https://github.com/aglitke/mom.

[23] A. Litke, Automatic Memory Ballooning with MOM, [retrieved: January, 2015], from https://aglitke.wordpress.com/2011/03/03/automatic-memory-ballooning-with-mom/

[24] Libvirt virtualizetion API, [retrived: January, 2015], from http://libvirt.org

 ISBN: 978-1-61208-388-9