

Performance of Memory Virtualization Using Local Memory Resource Balancing

P.V.S.S.Gangadhar 1,

*1(Scientist-D at NIC & Research Scholar at IT Department, GIT,
GITAM University, Visakhapatnam, India.*

Dr. Ashok Kumar Hota 2

*2(Scientist-F at National Informatics Centre, MEITY, Govt Of India,
Bhubaneswar,India,*

Dr. M.Venkateswara Rao 3

3(Professor, Dept. of IT, Gitam University,Visakhapatnam, India,

Dr. V.Venkateswara Rao 4

4(Professor, Dept. of CSE,Sri Vasavi Engg. College,Tadepalligudem,India.

Abstract

Virtualization has become a universal generalization layer in contemporary data centers. By multiplexing hardware resources into multiple virtual machines and therefore facilitating several operating systems to run on the same physical platform at the same time, it can effectively decrease power consumption and building size or improve security by isolating Virtual Machines. In a virtualized system, memory resource supervision acts as a decisive task in achieving high resource employment and performance. Insufficient memory allocation to a Virtual Machine will degrade its performance radically. On the opposing, over allocation reasons ravage of memory resources. In the meantime, a Virtual Machine's memory stipulates may differ drastically. As a consequence, effective memory resource management calls for a dynamic memory balancer, which, preferably, can alter memory allocation in a timely mode for each Virtual Machine based on their present memory stipulate and therefore realize the preeminent memory utilization and the best possible overall performance. Migrating operating system instances across discrete physical hosts is a helpful tool for administrators of data centers and clusters: It permits a clean separation among

hardware and software, and make easy fault management. In order to approximate the memory stipulate of each Virtual Machine and to adjudicate probable memory resource disagreement, a extensively planned approach is to build an Least Recently Used based miss ratio curve which provides not only the current working set size but also the correlation between performance and the target memory allocation size. Regrettably, the cost of constructing an MRC is nontrivial. In this paper, we initially present a low overhead LRU-based memory demand tracking scheme, which includes three orthogonal optimizations: AVL based Least Recently Used association, dynamic hot set sizing. Our assessment outcome confirm that, for the complete SPEC CPU 2006 benchmark set, subsequent to pertaining the three optimizing techniques, the mean overhead of MRC construction are lowered from 173% to only 2%. Based on current WSS, we then predict its trend in the near future and take different tactics for different forecast results. When there is a adequate amount of physical memory on the host, it locally balances its memory resource for the VMs. Once the local memory resource is insufficient and the memory pressure is predicted to sustain for a sufficiently long time, a relatively expensive solution, VM live migration, is used to move one or more VMs from the hot host to other host(s). Finally, for transient memory pressure, a remote cache is used to alleviate the temporary performance penalty. Our experimental results show that this design achieves 49% center-wide speedup.

Keywords - virtualization, virtual machine, cloud computing, data centric, physical memory, load balancing, cluster, resource allocation.

1. INTRODUCTION

Virtualization is becoming persistent in massive data centers, cloud computing, and enterprise infrastructure, motivated by a number of significant benefits, such as theatrical cost reduction, enlarged application availability and further well-organized IT management. According to Gartner (16), today, 25% of installed server workloads are virtualized. IDC even forecasts that, by 2014, more than 70% of applications on newly distributed servers will run in virtual machines (29). However, in a virtualized environment, competent and effectual memory resource management is silent a demanding problem. In this paper we recommend a memory resource balancing method to develop performance and memory resource consumption for center-wide virtualized computing. We show that our elucidation can correctly monitor memory demand of each virtual machine with very low operating cost and can successfully improve overall system performance. Virtualization technologies like Xen(9), VMware(54), and Denali(56) have turn into a common generalization layer in contemporary data centers. They facilitate multiple operating systems to run on their own virtual machines separately. Figure 1.1 illustrates an example, where the hypervisor multiplexes the hardware of a single physical machine with several virtual machines and a guest Operating System executes inside each virtual machine separately. One of the major benefits of using virtualization is server consolidation. It

is not unusual to achieve a 15-to-1 or even higher *consolidation ratio* (11), which is the ratio of virtual to physical machine without troublesome performance impact. For a data center that hosts a large number of servers, this can successfully save power consumption, floor space possession and air conditioning costs. In addition, virtualization can advance ease of use by live migration (15). When one physical server fall short or wants maintenance, the virtual machines it hosts can be clearly migrated to another physical machine with insignificant application *downtime*. The core of virtualization is the *virtual machine monitor* which is also called *hypervisor*. VMM is accountable for building and organization multiple instances of virtual hardware platforms. A bundle of physical resources like CPUs or network interface cards can be multiplexed in a time-sharing manner, which is like to how multiple processes of a native Operating System would split them. However, the memory system is shared all the way through address space partitioning. That is, each virtual machine is allocated with a fixed amount of address space of physical memory. However, conflicting from how a resident Operating System administers virtual memory and physical memory for its processes, for the purpose of fidelity, the VMM is not actively involved in memory management of each Virtual Machine. More particularly, when created, each VM is allocated with a fixed amount of physical memory. Then, it is the guest Operating System's job to supervise that amount of physical memory without the involvement of the hypervisor. As a result, the hypervisor is unaware of memory demand of VMs and thus powerless to dynamically balance memory resources. In our solution, we first design a low cost but accurate Least Recently Used based working set size tracking scheme as the basis of memory resource balancing. The Least Recently Used based working set size form associates memory allocation size and performance impact. Based on the form, we propose a local memory balancing method, which dynamically amend memory allocation amount via ballooning (34, 9) on a single physical machine. Then it is unmitigated to a global surroundings, where the physical memory of all interconnected machines is balanced via live migration and remote caching. To the best of our knowledge, our work uniquely coordinates the global memory balancing practices with a local balancing scheme. Without effective local memory balancing, the efficiency of global memory balancing will be significantly weakened. Figure 1.2 shows the overview of our solution.

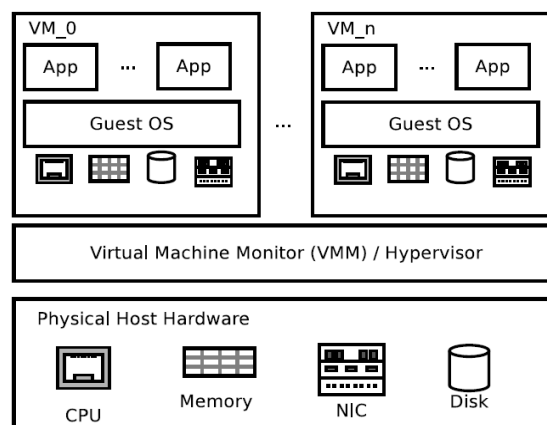
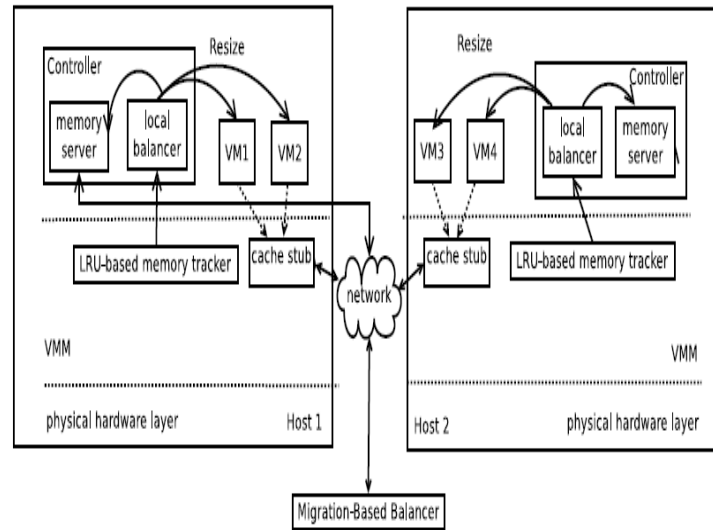


Figure 1.1 Organization of virtualization**Figure 1.2** Solution overview

2. RELATED WORK

2.1 Memory Management

In this section, we first briefly introduce how memory is managed in a native operating system. Then we describe how memory management is adapted for virtualization and the challenges that virtualization brings.

2.1.1 Memory Management in A Native OS

Virtual memory was first described in 1960s (21) and it has become a standard feature of modern general-purpose operating systems. It gives each process an illusion that it is running on a standalone and contiguous memory address space, called *virtual address space*. The size of a virtual address space can be larger than the amount of available real, physical memory. Typically, memory is allocated and reclaimed at a fixed granularity, called *page*, whose size is determined by processor architecture. Both virtual address space and physical address space are measured at the unit of page size. Inside the operating system, it maintains a translation table for each process, named *page table*, which maps a virtual page number to a physical page frame number. Inside a processor, there is a *memory management unit* (MMU) that dynamically translates virtual addresses that an application references to the corresponding physical addresses by looking up the current page table. When a system runs out of physical memory, and if some process requests for a free page, the OS selects some physical frame, saves its contents on a secondary storage (called *swapping-out* or *paging-out*) and allocates the page to the process. Later on, if the

previously paged out data is needed, it is loaded into a free physical memory frame, which is called *paging-in* or *swapping-in*. This page swapping scheme is called *demand-paging*. The strategy that determines which pages are swapped out is called *page replacement policy*. Since the latency of disk accesses is usually thousands of times longer than that of memory accesses, frequent page swapping will significantly damage the performance. A theoretically optimal page replacement policy should select a page whose next use will be the farthest in the future (2). However, in a general purpose OS, it is impossible to precisely predict the future memory access behaviors. In real world, a commonly used algorithm is the *least recently used (LRU)* replacement policy or its approximation. It works based on the property of program locality. That is, the pages that have been heavily and recently used are also most likely to be heavily used in near future. For an OS that uses demand paging and the LRU page replacement policy, a process that heavily uses a set of pages will automatically secure those pages in physical memory.

2.1.2 Memory Management with Virtualization

To create a virtualized environment, the hypervisor runs at the most privileged level that a native OS runs at. For CPUs without virtualization support, guest OSes and its processes run at the non-privileged level. Those privileged operations in the guest OS, such as page table setup, I/O instructions and etc. are either statically replaced with calls to the hypervisor or dynamically trapped and emulated via binary rewriting by the hypervisor. With hardware virtualization support, an unmodified guest OS runs on the CPU directly without intervention by the VMM until it tries to execute a restricted instruction. At this point, the hypervisor takes the control to emulate the instruction. When an operating system runs on the top of hypervisor, another level of memory address space, *guest physical address (GPA)* space, is introduced. To avoid confusion, in virtualization, the real physical address is specifically referred as *machine physical address (MPA)*. GPA is used by guest OSes in their physical address space. The purpose of using GPA is to provide the guest OS the impression that it is running on a real machine with certain amount of contiguous physical memory starting from address 0 because there is no guarantee that every guest OS will be allocated with contiguous machine memory and most OSes do not support fragmented physical address space. The size of the GPA space of an OS is the same as the amount of machine memory that the hypervisor statically allocates to the guest OS when it is created. Meanwhile, the hypervisor maintains a mapping table to map the contiguous GPA space to the possibly scattered MPA space. Figure 2.1 contrasts the address spaces between a native OS and a virtualized OS. To support the translation from GPA to MPA, currently there are two kinds of approaches based on whether the processor supports MMU virtualization or not (1, 53, 39): two software-based techniques and a hardware-based approach.

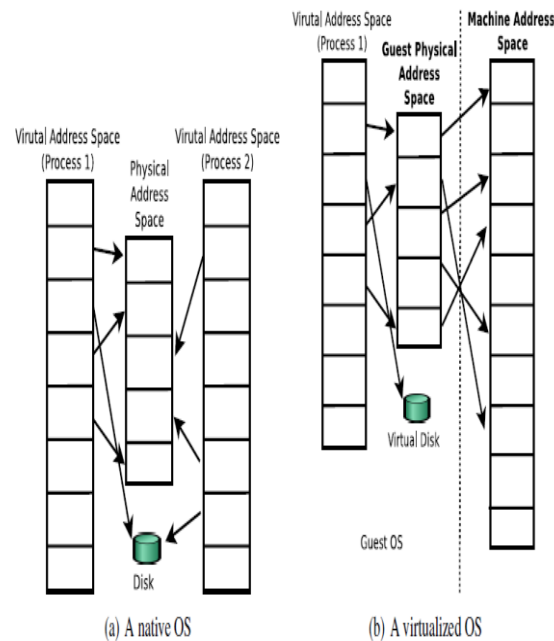


Figure 2.1 Address spaces in a native OS and a virtualized OS

2.2 Working Set Size Estimation

In this section, we first introduce the concept of working set. We then discuss various techniques to estimate the size of a working set.

2.2.1 Working Set

Denning (17) first defined the working set as the set of memory pages referenced by a process during a time interval. The size of the working set (WSS) is the amount of memory that a process needs without paging. Even if the process has very large memory footprint, those pages not in its working set can be reclaimed without performance penalty. The same idea can be extended to VMs. Assuming a guest OSes can well utilize their allocated physical memory, and if the memory allocation amount of each VM is exactly its WSS, then the physical memory allocation on the host is optimal. Therefore, working set size estimation provides a necessary metric that ensures the system to reach the maximum performance as expected.

2.2.2 Miss Ratio Curve Based Working Set Size Estimation

The miss ratio curve (MRC) based WSS estimation is a widely proposed technique (38, 13, 67, 60, 30, 51). A page miss ratio curve plots the page miss ratios against various amount of memory allocation. When the allocation size is no less than a system's WSS, the miss ratio is 0, which means all accesses will hit in main memory.

When the allocation size is less than its WSS, the MRC tells the ratio of many accesses that will cause page swapping. With an MRC, we can redefine WSS as the size of memory that results in less than a predefined tolerable page miss rate. Since an MRC models the performance and memory allocation size, it is especially suitable for memory resource arbitration. For example, when two applications compete for memory resources, in order to achieve optimal overall performance, the arbitration scheme needs to evaluate how the performance would be impacted by varying their allocation sizes. A commonly used method to calculate MRC is Mattson's stack algorithm (38). It was initially proposed to reduce the time of trace-driven cache simulation. The algorithm uses an LRU stack to store the page numbers of accessed page frames. For each entry of the LRU stack, its distance to the top of the stack is called *stack distance* or *LRU distance*. Each stack entry i is associated with a counter, denoted as $Hist(i)$. When a page is referenced, the algorithm first searches the page number in the stack and computes its stack distance, $dist$. It then increments the hit counter $Hist(dist)$ by one. Finally, it updates the stack by moving the page number to the top of the stack. One can plot an *LRU histogram* by relating each counter value to its corresponding LRU distance. If there is a stack with depth D and we reduce it to depth d , then the expected miss ratio can be calculated as follows:

$$Miss_ratio(d) = \frac{\sum_{i>d}^D Hist(i)}{\sum_{i=0}^D Hist(i)}$$

For example, given a system with only four pages of physical memory, the top half of Figure 2.4 shows the hit counts for some application that makes a total of 200 memory accesses. The histogram indicates that 100 accesses hit the most recently used (MRU) page, 50 accesses hit the second MRU slot, and so on. Apparently the page hit rate is $(100+50+20+10)/200 = 90\%$. We can tell that if we reduce the system memory size by a half, the hits to the first two MRU slots are still there while the hits to the next two MRU slots now become misses. The hit rate becomes $(100+50)/200 = 75\%$. The LRU histogram thus can accurately predict miss rate with respect to the LRU list size. The bottom part of Figure 2.3 is the miss ratio curve corresponding to the histogram on the top.

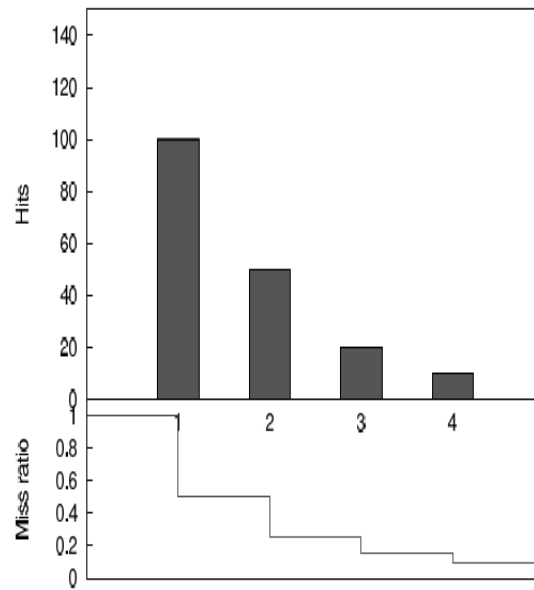


Figure 2.3 LRU histogram example

3. LOCAL MEMORY RESOURCE BALANCING

With the LRU miss ratio curves of all VMs on a physical machine, we can dynamically

Adjust each VM's memory allocation size. We call this scheme *local memory resource*

Balancing. In this chapter, we first present our memory resource balancing and arbitration scheme in Section 4.1. When there is no sufficient physical memory to meet all VMs' memory requirement, our arbitration algorithm is able to quickly find an allocation plan that aims for the overall performance.

3.1 Local Memory Resource Balancing And Arbitration

Once the working set sizes of all VM are estimated, the balancer needs to determine the target allocation sizes for them. Assume that P is the size of all available host machine memory when no guest is running, and V is the set of all VMs. For QoS purposes, each $VM_i \in V$ is given a lower bound of memory size L_i . Let $E_i = \max(L_i, WSS_i)$ be the *expected memory size* of VM_i . When $P \geq \sum E_i$, all VMs can be satisfied. We call the residue of allocation ($P - \sum E_i$) as *bonus*. The *bonus* can be spent flexibly. In our implementation, we aggressively allocate the bonus to each VM proportionally according to E_i . That is $T_i = E_i + \text{bonus} \times \frac{E_i}{\sum E_i}$

where T_i is the final target memory allocation size. When $P < \sum E_i$, at least one VM cannot be satisfied. Here we assume all VMs have the same priority and the goal is to minimize system wide page misses. Let $mrc_i(x)$ be the miss ratio curve and nri be

number of memory accesses in a recent epoch of VM_i . Given a memory size m , the number of page misses is $miss_i(m) = m \times cri(m) \times nri$. The balancer tries to find an allocation $\{T_i\}$ such that $\sum_i miss_i(T_i)$, the total penalty, is a minimum. Since ballooning adjusts memory size on a page unit, a brute force search takes nearly $O(M|V|)$ time, where M is the maximum number of pages a VM can get. We propose a quick approximation algorithm. For simplicity of discussion, we assume that there are two VMs, VM1 and VM2, to balance. Choosing an increment/decrement unit size S ($S \leq G$), the algorithm tentatively reduces the memory allocation of VM1 by S ,

Increases the allocation of VM2 by S , and calculates the total page misses of the two VMs based on the miss ratio curves. We continue this step with increment/decrement strides of $2S$, $3S$, and so on, until the total page misses reach a local minimum. The algorithm then repeats the above process but now reducing allocation of VM1 while increasing allocation for VM2. It stops when it detects the other local minimum. The algorithm returns the allocation plan based on the lower of the two minima. This algorithm can run recursively when there are more than two VMs. It is possible that the two minima are close to each other in terms of page misses but the allocation plans can be quite different. For example, when two VMs are both eager for memory, one minimum suggests $(VM1 = 50MB, VM2 = 100MB)$ with total page misses of

1000, while the other one returns $(VM1 = 100MB, VM2 = 50MB)$ with total page misses of 1001. The first solution wins slightly, but the next time, the second one wins with a slightly lower number of page misses and this phenomenon repeats. The memory adjustment will cause the system to thrash and degrade the performance substantially. To prevent this, when the total page misses of both minima are close (e.g. the difference is less than 10%), the allocation plan that is closer to the current allocation is adopted. It is also necessary to limit the extent of memory reclaiming. Reclaiming a significant amount of memory may disturb the target VM because the inactive pages may not be ready to be reclaimed instantly. So during each balancing, we limit the maximum reduction to 20% of its current memory size to let it shrink gradually.

4. EXPERIMENTAL EVALUATION

In our implementation, the local balancer is written in Python and runs in domain-0, a privileged guest domain. It communicates with hypervisor via hyper calls to acquire LRU histograms and resize memory allocation. We set the balancing frequency as every 5 seconds, an empirical value, which allows the WSS estimator to collect enough information but not too long to miss optimizing opportunities.

To evaluate the effect of automatic memory resource balancing, we first start from balancing for two VMs, each runs different workloads. We evaluate various workload combinations, including a simple scenario of CPU intensive + memory intensive workloads where there is no memory contention, a workload combination of DaCapo and SPEC WEB that has occasional memory competition, a combination of two memory intensive workloads.

4.1 Balancing For Two VMs (CPU Intensive + Memory Intensive Workloads)

Our evaluation starts with a simple scenario where memory resource contention is rare. The workloads include the DaCapo benchmark suite and 186.crafty, a program with intensive CPU usage but low memory load. On VM1, 186.crafty runs 12 iterations followed by the DaCapo benchmark suite. Meanwhile, on VM2, the DaCapo suite runs first, followed by the same number of iterations of 186.crafty.

Figure 4.1(a) displays the actual allocated memory size and expected memory size on both VMs respectively. Note that the VMrunning 186.crafty gradually gives up its memory to the other VM. When both VMs are running 186.crafty, *bonus* is gradually allocated to the two VMs.

To show the performance that an ideal memory balancer could deliver, we measure the *best case* performance on two VMs, each with 380 MB fixed memory, the peak memory allocation that a VM could own during balancing.

Table 4.1 lists the number of major page faults and execution time for both VMs. With memory balancing, the number of total major faults is reduced by a factor of 25.

Table 4.1 Major page faults and execution time

	Baseline		Balancing		Best	
	VM1	VM2	VM1	VM2	VM1	VM2
Major page faults	158,810	110,965	4102	6499	1428	538
Execution time (DaCapo)	1,619	1,267	147	153	127	110
Execution time (186.crafty)	32.5	32.5	32.7	32.9	32.5	32.7

Figures 4.1(b) and 4.1(c) show the execution time of each benchmark in the three settings

respectively: baseline, best case, and balancing. With memory balancing, the performance of 186.crafty is nearly the same, but DaCapo gains a speedup of 11 and 8.3 on the two hosts, respectively. Most notable improvements are from Eclipse and Xalan, whose average execution time on the two VMs is cut into 1/18 and 1/32, respectively. These two benchmarks require around 350 MB memory, resulting in a large number of page faults without memory balancing. Eventually, using memory balancing, it achieves an overall speedup of the two VMs of 8.05.

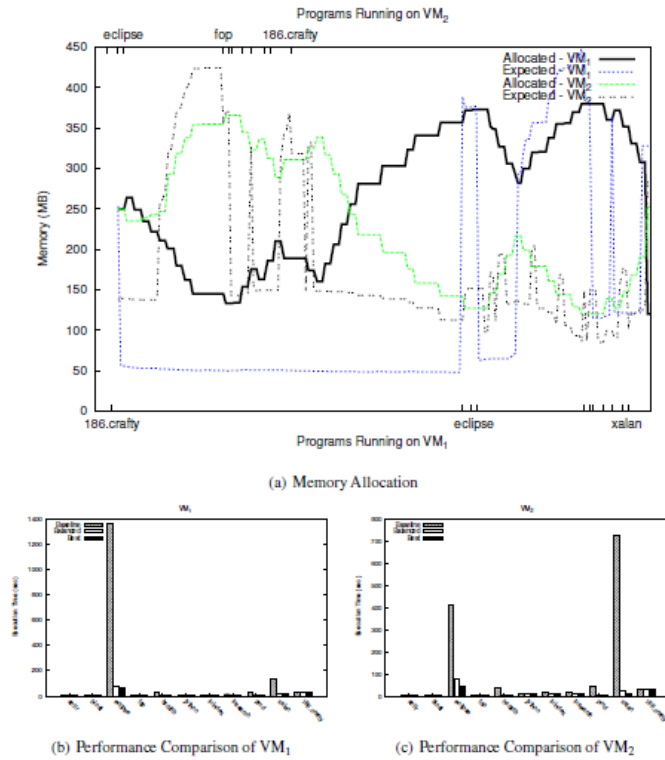


Figure 4.1 Local memory resource balancing: DaCapo + 186.crafty.

For readability, in Fig. 4.1(a), only a few program names of DaCapo are labeled.

Fig. 4.1(b) and 4.1(c) show the complete program names in the order of execution.

4.2 Mixed Workloads Of Four VMs

To simulate a more realistic setting in which multiple VMs are hosted and diverse applications are deployed, four VMs are created and different workloads are assigned to each of them. VM1 runs the DaCapo suite, VM2 runs the DaCapo suite in reverse order, VM3 runs 186.crafty for 12 iterations, and VM4 runs SPEC Web 2005.

As shown in Figure 4.2, with memory balancing, the performance of DaCapo and DaCapo are boosted by a factor of 8.17 and 10.72, respectively, with the cost of a 70% slowdown for Banking. The performance of 186.crafty and Ecommerce are slightly impacted by 3% and 5%. The overall mean speedup of using memory balancing is 1.72.

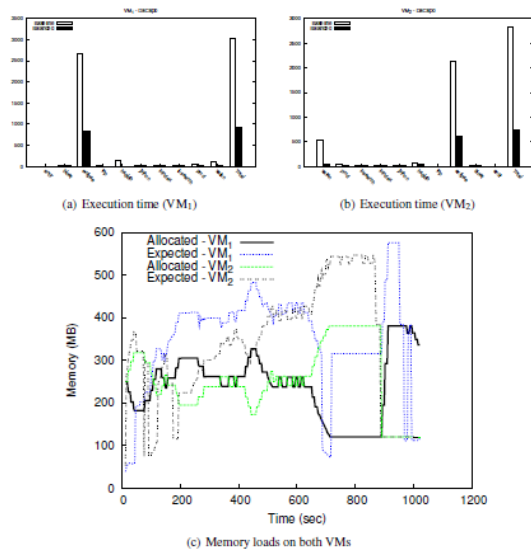


Figure 4.2: DaCapo + DaCapo'

5. CONCLUSION

As demonstrated by the experimental results, based on the WSS tracking scheme, our local memory resource balancer can effectively improve overall system performance. Even for the case with heavy memory resource competition, our arbitration algorithm still boosts the overall performance by a factor of 3. And for programs with large working set sizes, the experimental results show that the WSS tracking scheme is able to guide memory balancing with low cost and eventually boots the overall performance by 1.85 times. The 4-VM setting shows that our balancing algorithm can balance memory resources for multiple virtual machines. To achieve better overall performance, the algorithm may sacrifice the performance of some VMs, but after applying higher priority to important VMs, the quality of service of those VMs can be guaranteed. Though the local balancing scheme improves the memory utilization of a single host, performance penalty still exists when the total memory demand of all VMs exceeds the host's available physical memory or a spike of memory demand occurs.

References

- [1] On Strategies for Dynamic Resource Management in Virtualized Server Environments. **Kochut, A and Beaty, K.** Washington, DC, USA : IEEE Computer Society, 2007. Proceedings of the 2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. pp. 193--200. 978-1-4244-1854-1.
- [2] **Ou, George.** Introduction to server virtualization. *TechRepublic*. [Online] May 22, 2006. <http://www.techrepublic.com/article/introduction-to-server-virtualization/6074941>.


- [3] *Live Migration of Virtual Machines*. **Clark, Christopher, et al., et al.** 2005. In Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI).
- [4] **Rajan, J. P.** How Live Migration works in Hyper-V R2. [Online] March 31, 2010. <http://blogs.technet.com/b/ranjanajain/archive/2010/03/31/how-live-migration-works-in-hyper-v-r2.aspx>.
- [5] **VMware, Inc.** VMware vMotion. [Online] 2009. www.vmware.com/files/pdf/VMware-VMotion-DS-EN.
- [6] J. Almeida, V. Almeida, D. Ardagna, C. Francalanci, and M. Trubian, "Resource Management in the Autonomic Service-Oriented Architecture," in Proceedings of the 2006 IEEE International Conference on Autonomic Computing (ICAC 2006), June 2006, pp. 84–92.
- [7] J. Anselmi, E. Amaldi, and P. Cremonesi, "Service Consolidation with End-to-End Response Time Constraints," in Proceedings of 34th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2008.), September 2008, pp. 345–352.
- [8] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica et al., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.
- [9] B. Arnold, S. A. Baset, P. Dettori, M. Kalantar, I. I. Mohomed, S. J. Nadgowda, M. Sabath, S. R. Seelam, M. Steinder, M. Spreitzer, and A. S. Youssef, "Building the ibm containers cloud service," *IBM Journal of Research and Development*, vol. 60, no. 2-3, pp. 9:1–9:12, March 2016.
- [10] M. Assuncao, M. Netto, B. Peterson, L. Renganarayana, J. Rofrano, C. Ward, and C. Young, "CloudAffinity: A framework for matching servers to cloudmates," in Proceedings of the 2012 IEEE Network Operations and Management Symposium (NOMS 2012), April 2012, pp. 213–220.
- [11] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-event system simulation*. Prentice Hall, 2010.
- [12] P. Barham et al., "Xen and the art of virtualization," in Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003), October 2003, pp. 164–177.
- [13] L. A. Barroso and U. Hölzl, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007.
- [14] C. L. Belady and D. Beaty, "Roadmap for datacom cooling," *ASHRAE journal*, vol. 47, no. 12, p. 52, 2005.
- [15] A. Beloglazov and R. Buyya, "Energy efficient allocation of virtual machines in cloud data centers," in Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid 2010), May 2010, pp. 577–578.


- [16] Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers,” in Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science, December 2010, pp. 4:1–4:6.
- [17] F. Caglar, S. Shekhar, and A. Gokhale, “A performance Interference-aware virtual machine placement strategy for supporting soft realtime applications in the cloud,” Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA, Tech. Rep. ISIS-13-105, 2013.
- [18] P. DELFORGE, “Energy efficiency, data centers—NRDC,” <http://www.nrdc.org/energy/data-center-efficiency-assessment.asp>, (Accessed on 02/18/2016).
- [19] B. H. Li, X. Chai, and L. Zhang, “New advances of the research on cloud simulation,” in *Advanced Methods, Techniques, and Applications in Modeling and Simulation*, vol. 4 of *Proceedings in Information and Communications Technology*, pp. 144–163, 2012.
- [20] . Jafer, Q. Liu, and G. Wainer, “Synchronization methods in parallel and distributed discrete-event simulation,” *Simulation Modelling Practice and Theory*, vol. 30, pp. 54–73, 2013.
- [21] R. Fujimoto, A. Malik, and A. Park, “Parallel and distributed simulation in the cloud,” *SCS Modeling and Simulation Magazine*, pp. 1–10, 2010.
- [22] A. W. Malik, A. J. Park, and R. M. Fujimoto, “An optimistic parallel simulation protocol for cloud computing environments,” *SCS M&S Magazine*, vol. 4, 2010.
- [23] A. Javor and A. Fur, “Simulation on the Web with distributed models and intelligent agents,” *Simulation*, vol. 88, no. 9, pp. 1080–1092, 2012.
- [24] IEEE Std 1516.1-2010, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)*, Framework and Rules Specification, 2010.
- [25] IEEE Std 1516.2-2010, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)*, Object Model Template (OMT) Specification, 2010.
- [26] IEEE Standard, *1516.1-2010—IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Federate Interface Specification*, 2010.
- [27] Google, "Google App Engine", (2012), [online]. Available: cloud.google.com [Nov 1, 2012].
- [28] Amazon, "Amazon Elastic Compute Cloud (Amazon EC2)", (2012), [online]. Available: aws.amazon.com/ec2/ [Nov 1, 2012].
- [29] Microsoft, "Windows Azure.", (2012), [online]. Available: windowsazure.com [Nov 1, 2012].

- [30] IBM, "SmartCloud." (2012), [online]. Available: ibm.com/cloudcomputing [Nov 1, 2012].
- [31] P. Mell and T. Grance, "The NIST definition of cloud computing(draft)," *NIST special publication*, vol. 800, p. 145.
- [32] A. Desai, "Virtual Machine." (2012), [online]. Available: <http://searchservervirtualization.techtarget.com/definition/virtualmachin>
- [33] VMWare, "vSphere ESX and ESXi Info Center.", (2012), [online]. Available: vmware.com/products/vsphere/esxi-and-esx [Nov 1, 2012].
- [34] Microsoft, "Windows Virtual PC.", (2012), [online]. Available: <http://www.microsoft.com/windows/virtual-pc/> [Nov 1, 2012].
- [35] Xen, "Xen Hypervisor.", (2012), [online]. Available: <http://www.xen.org/products/xenhyp.html> [Nov 1, 2012].
- [36] Microsoft, "Hyper-V Server 2012.", (2012), [online]. Available: microsoft.com/server-cloud/hyper-v-server/ [Nov 1, 2012].
- [37] KVM, "Kernel-based Virtual Machine.", (2012), [online]. Available: linux-kvm.org [Nov 1, 2012].
- [38] Oracle, "VirtualBox.", (2012), [online]. Available: virtualbox.org [Nov1, 2012]
- [39] Nagpurkar P, Krintz C, Hind M, Sweeney PF, Rajan VT. Online phase detection algorithms. In Proceedings of the International Symposium on Code Generation and Optimization. 2006;(CGO '06):111–123.
- [40] Newhall T, Finney S, Ganchev K, Spiegel M. Nswap: A network swap module for linux clusters. In Proceedings of the 9th European Conference on Parallel Processing (Euro-Par). 2003:1160–1169.
- [41] Pinter SS, Aridor Y, Shultz S, Guenender S. Improving machine virtualization with 'hotplug memory'. In Proc. 17th International Symposium on Computer Architecture and High Performance Computing SBAC-PAD 2005. 24–27 Oct. 2005:168–175.
- [42] Romer TH, Ohlrich WH, Karlin AR, Bershad BN. Reducing tlb and memory overhead using online superpage promotion. SIGARCH Computer Architecture News. 1995;23(2):176–187.
- [43] Sapuntzakis CP, Chandra R, Pfaff B, Chow J, LamMS, RosenblumM. Optimizing the migration of virtual computers. In In Proceedings of the 5th Symposium on Operating Systems Design and Implementation. 2002:377–390.
- [44] Seiden SS. A guessing game and randomized online algorithms. In Proceedings of the thirty-second annual ACM symposium on Theory of computing. 2000;(STOC '00):592–601.
- [45] Shen X, Zhong Y, Ding C. Locality phase prediction. In Proceedings of the

- 11th International Conference on Architectural Support for Programming Languages and Operating Systems. 2004.
- [46] Sherwood T, Perelman E, Calder B. Basic block distribution analysis to find periodic behavior and simulation points in applications. In Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques.2001;(PACT '01):3–14.
- [47] Sherwood T, Perelman E, Calder B. Basic block distribution analysis to find periodic behavior and simulation points in applications. In Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques. 2001;(PACT '01):3–14.
- [48] Sherwood T, Sair S, Calder B. Phase tracking and prediction. In Proceedings of the 30th International Symposium on Computer Architecture. 2003.
- [49] Shirazi BA, Hurson AR, Kavi KM. Scheduling and Load Balancing in Parallel and Distributed Systems. IEEE Computer Society; 1995.
- [50] Sugumar RA, Abraham SG. Efficient simulation of caches under optimal replacement with applications to miss characterization. In Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement & Modeling Computer Systems. May 1993:24–35.
- [51] Tam DK, Azimi R, Soares LB, Stumm M. RapidMRC: Approximating l2 miss rate curves on commodity systems for online optimizations. In Proceeding of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems. 2009:121–132.
- [52] Uhlig R, Neiger G, Rodgers D, Santoni A, Martins F, Anderson A, Bennett S, Kagi A, Leung F, Smith L. Intel virtualization technology. Computer. may 2005;38(5):48 – 56.
- [53] Waldspurger CA. Memory resource management in VMware ESX server. SIGOPS Operating Systems Review. 2002;36(SI):181–194.
- [54] Werstein P, Jia X, Huang Z. A remote memory swapping system for cluster computers. In Parallel and Distributed Computing, Applications and Technologies, 2007. PDCAT '07. Eighth International Conference on. dec. 2007:75 –81.
- [55] Whitaker A, Shaw M, Gribble SD. Denali: Lightweight virtual machines for distributed and networked applications. In In Proceedings of the USENIX Annual Technical Conference. 2002.
- [56] Williams D, Jamjoom H, Liu YH, Weatherspoon H. Overdriver: handling memory overload in an oversubscribed cloud. In Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. 2011;(VEE '11):205–216.
- [57] Wood T, Shenoy P, Venkataramani A, Yousif M. Black-box and gray-box strategies for virtual machine migration. In Proceedings of the 4th USENIX

- conference on Networked systems design & implementation. 2007;(NSDI'07):17–17.
- [58] Yang T, Hertz M, Berger ED, Kaplan SF, Moss JEB. Automatic heap sizing: taking real memory into account. In Proceedings of the 4th international symposium on Memory management. 2004;(ISMM '04):61–72.
- [59] Yang T, Berger ED, Kaplan SF, Moss JEB. CRAMM: virtual memory support for garbage collected applications. In Proceedings of the 7th Symposium on Operating Systems Design and Implementation. 2006:103–116.
- [60] Zayas E. Attacking the process migration bottleneck. SIGOPS Operating Systems Review. 1987;21(5):13–24.
- [61] Zeileis A, Kleiber C, Kramer W, Hornik K. Testing and dating of structural changes in practice. Computational Statistics & Data Analysis. 2003;44:109–123.
- [62] Zhang X, Dwarkadas S, Shen K. Towards practical page coloring-based multi-core cache management. In Proceedings of the 4th ACM European Conference on Computer systems. 2009.
- [63] Zhao W, Wang Z. Dynamic memory balancing for virtual machines. In Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. 2009:21–30.
- [64] Zhao W, Jin X, Wang Z, Wang X, Luo Y, Li X. Low cost working set size tracking In Proceedings of the 2011 annual conference on USENIX Annual Technical Conference. 2011.
- [65] ZhaoW, Jin X,Wang Z, XiaolinW, Yingwei L. Efficient LRU-based working set size tracking. Technical Report CS-TR-11-01, Houghton, MI, USA, 2011.

	<p>P.V.S.S.GANGADHAR is Scientist-D in NIC & Ph.D Scholar. Presently studying Ph.D, Department of Information Technology at Gitam Institute of Technology, Gitam University,Vishakapatnam, AndhraPradesh, India. His Research interests e-governance, Cloud computing, fuzzy logic and Data mining ORCID:0000-0002-8548-8492</p>
	<p>Dr. Ashok Kumar Hota is Scientist-F at NIC, MEITY,OSU, Bhubaneswar, Govt of India. His research interests include e-governance, Tribal informatics, Cloud Computing, Data Mining, and Big Data Analytics. He published several papers in International conferences journals.</p> <p>ORCID:0000-0002-9117-1504</p>

	<p>Dr. Mandapati Venkateswara Rao is Professor in Department of Information Technology at Gitam Institute of Technology, Gitam University, Vishakhapatnam, India. He has received M.Tech in CST and PhD in Robotics from Andhra University. His Research Interests includes Robotics, Cloud Computing and Image processing. He published several papers in International conferences and journals. ORCID:0000-0002-7598-3473</p>
	<p>Dr. Vedula Venkateswara Rao is Professor in the Department of Computer Science Engineering at Srivasavi Engineering College, tadepalligudem, India. He received Masters Degree in Computer Science Engineering from JawaharLalNehru Technological University Kakinada, Masters Degree In Information Technology from Punjabi University, Patiyala, India and PhD from Gitam University. His research interests include Cloud Computing and Distributed Systems, Data Mining, Big Data Analytics and Image Processing. He published several papers in International conferences and journals.</p> <p>ORCID:0000-0003-0131-4944</p>