Automatic Memory Control of Multiple Virtual Machines on a Consolidated Server

Prof. M.V. Raut, Prof. A.K.Dere

Assistant Professor

Computer Department
raut.manju23@gmail.com, archu.dere@gmail.com

Abstract:

The virtualization means, multiple virtual machines can coexist and operate on one physical machine. When virtual machines compete for memory, the performances of applications deteriorate, especially those of memory-intensive applications. The aim is optimize memory control techniques using a balloon driver for server consolidation. To design and implement an automatic control system for memory based on a Xen balloon driver. To avoid interference with VM monitor operation, our system works in user mode; therefore, the system is easily applied in practice. To design an adaptive global-scheduling algorithm to regulate memory. This algorithm is based on a dynamic baseline, which can adjust memory allocation according to the memory used by the VMs.To evaluate our optimized solution in a real environment with 10 VMs and well-known benchmarks. Experiments confirm that our system can improve the performance of memory-intensive and disk-intensive applications by up to 500 and 300 percent.

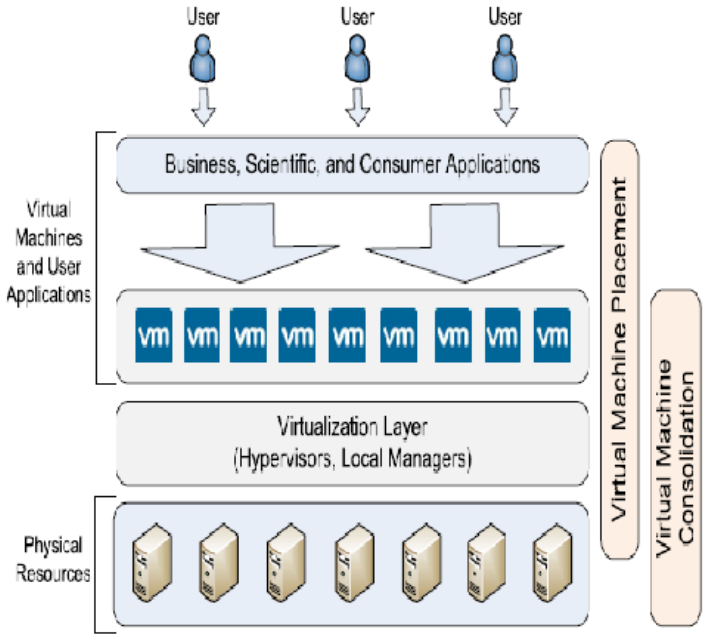
Keywords: *Virtual machine, server consolidation, memory control, global-scheduling.*

1. Introduction

Virtualization is a framework or methodology of dividing the resources of a computer into multiple execution environments, by applying one or more concepts or technologies such as hardware and software partitioning, time-sharing, partial or complete machine simulation, emulation, and quality of service. Although the resources of these virtual machine such as CPU and memory are isolated through virtual machine monitor (VMM) subsystems automatic control systems can reallocate the limited resources of the consolidated server dynamically, which can reduce the running time of applications and maximize resource utilization. Automatic control systems for CPU devices have been widely researched but time sharing for memory devices remains an open issue Normally, memory is statically allocated to each VM when the machine is booted, and memory size does not vary throughout the life cycle of the VM. Memory control in Virtual Machine face a minimum of three new challenges in the context of server consolidation:

1. Tools for automatic memory control at the application level require further investigation. To activate underlying mechanisms and to generate low-level interfaces, Xen , VMware, and KVM have implemented page sharing, virtual hot plugs, and balloon drivers in the virtual machine monitor. However, these mechanisms and interfaces only focus on the underlying methods in have implemented page sharing, virtual hot plugs, and balloon drivers in the virtual machine monitor. However, these mechanisms and interfaces only focus on the underlying methods in kernel mode to resize the memory for an individual virtual machine. They cannot specify which VM needs to reclaim/release its memory or how many pages it should take/give in a global perspective.

2. Memory scheduling algorithms need to be more adaptive to different scenarios, regardlessof when the global memory is sufficient or insufficient. Each VM can submit a memory value,called committed memory, which will be used in the future. The memory state is sufficient if the sum of the committed memories of all VMs is smaller than the available memory of the physical machine. Otherwise, the memory state is insufficient. Our previous work focused only on the sufficient state. Memory scheduling algorithms must be in a sufficient state, and self-ballooning was not observed with a global perspective.To avoid this local minimum and to attain optimal performance, additional algorithms should thus be developed for general global scheduling



Proposed solution to Multiple VMs on a Consolidated Server

3 The scale at which previous evaluations are performed is not coherent with the virtual Machine consolidation ratios used by large vendors. Although few cloud computing companies (Amazon) are willing to disclose the number of VMs they can host on a physical server, they on servatively estimate that one server contains at least 10 or 12 virtual machines However, previous experiments are limited to a maximum of two or four virtual machines. These experiments also adopt workloads that are synthetic and traces driven. Therefore, more tests should be conducted with with additional virtual machines.

2 Related work

In this current framework, Memory is then occasionally reallocated utilizing this inflatable driver. Be that as it may, our framework has three noteworthy favorable circumstances over MEB. In the first place, MEB alters the VMM piece to capture memory access and screen memory utilization. This procedure creates overwhelming extra over-burdens and break down VMM execution. In any case, our framework is lightweight and can be totally joined into client space without meddling with VMM activity. Second, MEB utilizes a brisk guess calculation to keep add up to page misses from achieving a neighborhood least. Our framework decides the ideal distribution of worldwide memory by presenting dynamic baselines and fathoming straight conditions. At last, MEB checks the viability of the calculation utilizing constrained assets, e.g., two and four VMs, though our framework can scale up to 10 VMs. Our framework can decide the distribution by tackling direct conditions with the dynamic standard, which fits both adequate and deficient physical memory.

3 Purposed work

We planned two booking calculations for the server: self-planning and worldwide booking. The booking calculation of Server at that point decides the space that requires extra pages, and in addition the area that gives these additional pages. The planning calculation likewise ascertains the ideal target pages for allotment to every space. The errand asks for in accessible assets to check the need in ideal arrangement undertaking in working of united server. The working framework utilized the inflatable driver blow up and collapses the demand pages best and first need of assignment in the worldwide booking. Assets asks for the ideal pages ascertain to put away in each virtual machine to be screen at a combined server powerfully time played out the season of assignment, without inertness of work thus keep on starting a next solicitations process.

4 System Model

1. The ballooning mechanism aims to over commit memory. In this process, physical memory can be allocated to all active domains, although the amount allocated is more than the total physical memory in the system. In 2002, Waldspurger first introduced the ballooning mechanism for the VMware ESX Server. In 2003, Xen also implemented this mechanism to allocate memory from one domain to others . As a result, memory from idle VMs or from domains that useless memory canbe committed to newly created VMs or to domains requiring additional memory. Virtual memory in Xen decouples the virtual address space from the physical address space.

Following figure shows the overall description of this system.

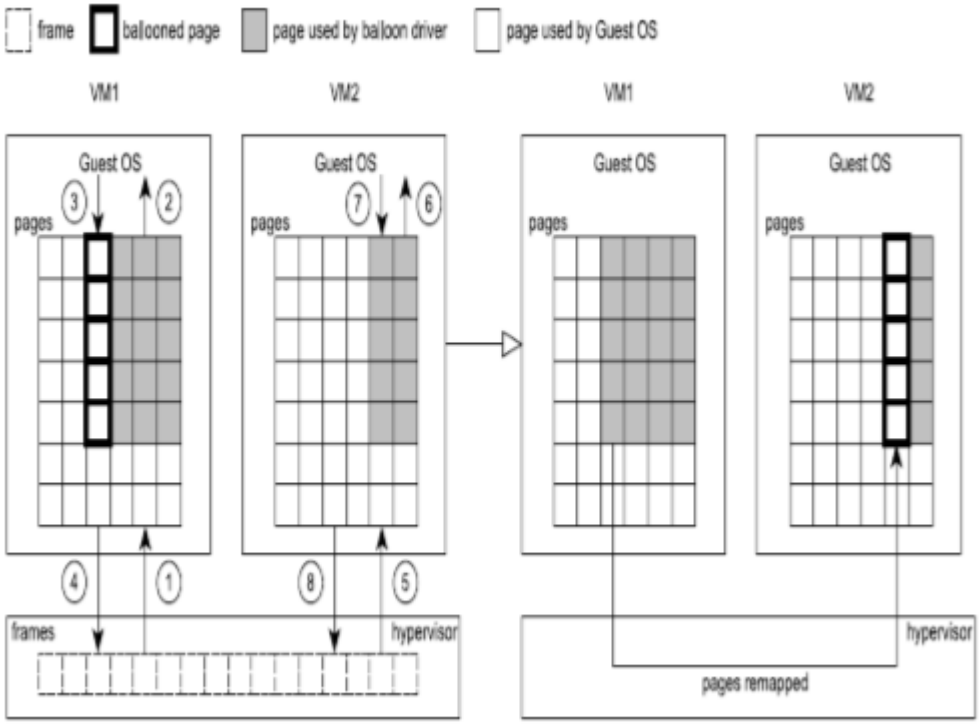


Figure 3.1.1:Block Diagram

2. The virtual memory in VMs and the physical memory in the actual machine are divided

Automatic Memory Control of Multiple Virtual Machines on a Consolidated Server into pages and frames, respectively. The pages are addressed by their Guest Physical Frame Numbers, or GPFNs. The frames are addressed by their Machine Frame Numbers, or MFNs. Every VM has a physical to machine translation table, which maps the GPFNs to MFNs. The Xen balloon driver resides in the domain but is controlled by the hypervisor . Fig. depicts its working process of inflation and deflation, with two VMs (VM1 and VM2) as examples. The left side of Fig. represents the initial page allocation of the VMs. The right side represents the remapped page allocation.

3. To in flate the balloon, first ,the hypervisors ends an inflation request to the balloon driver in VM1(phase1).Then, the balloon driver requests free pages from its Guest OS(phase2). After acquiring the pages, it records their corresponding GPFNs (phase3). It then notifies the hypervisor to replace the behind these GPFNs with invalid entry. Finally, the hypervisor puts the sereclaimed MFNs on its own free list , which is given toVM2(phase4). To deflate the balloon, the balloon driver in VM2 receives a deflation request from the hypervisor (phase 5). Then, the balloon driver releases pages to its Guest OS (phase 6). If the Guest OS is allowed to increase its page numbers and if free frames are available (phase 7), the hypervisor will allocate MFNs behind the GPFNs to increase the pages used by the Guest OS inVM2(phase8). NotethatfromVM1sperspective, the ballooned pages appear to still be in use by its balloon driver. In fact, the frames behind these pages have been reclaimed by the hypervisor and remapped to the ballooned pages in VM2.

5 System Architecture

(a) Domain :A domain is a VM that is operating on a system. On boot, the Xen hypervisor activates the first. Mechanism of the Xen balloon driver. domain (Domain0), on which a Guest OS runs. Through Xen control tools, Domain0 is privileged to access the hardware and to manage other domains. These other domains are referred to a DomainUs and are unprivileged; they can thus run on any Guest OS that has been ported to Xen.

(b) Balloon : The Xen balloon driver is the basis of and supports our system of automatic memory control technically. They can thus focus on efficiently allocating the memory pages across various domains.

Following figure shows the overall description of this system architecture.

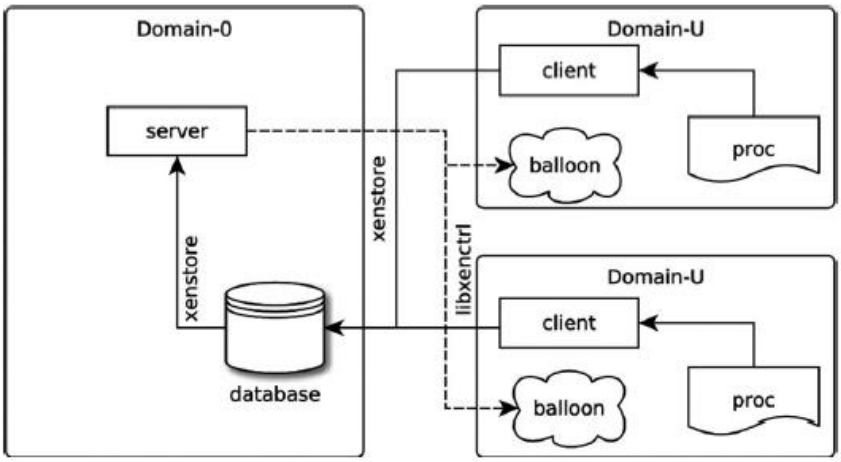


Figure 4.1:System Architecture

(c) XenStore : This is a hierarchical namespace shared between domains This is a hierarchical namespace shared between domains, which stores the running information of domains . It also provides primitives to either read or Automatic Memory Control of Multiple Virtual Machines on a Consolidated Server write a key, enumerates a directory, and generates notifications when a key changes value. XenStore is categorized into three branches: /vm stores configuration information about domains; /local/domainstores information regarding the domain in the local node. Its key (¡domid¿/memory/ target) contains the target page number of the domain; /tool stores information for various tools. XenStore can be accessed by virtual input/output (I/O) drivers using the in-kernel application programming interface (API) XenBus.

(**d) proc:**A process file system is a virtual file system in the Guest OS layer that contains dynamic information related to kernel and system processes. The directory /proc/meminfo stores the memory information. MemTotal is the total page number, MemFree is the size of unallocated pages, Buffers denotes the buffer size for files, and Cached is the size of the pages used by caches. The total free pages of the system includes MemFree, Cached, and Buffers. SwapTotal is the total size of swap memory, and SwapFree is the size of the free swap memory.

(e) Libxenctrl : This is a C interface that can be called by libraries or applications in the domains to interact with the hypervisor.

(**f) Database :**The database is hosted by Domain0 and functions in the application layer, which stores page information from DomainUs. Database contains the following records

1) the total GPFNs of the domain, which is derived from /proc/meminfo/MemTotal;
2) the maximal GPFNs of used memory Mem- Used, which is calculated as follows:

MemUsed = MemTotal - MemFree - Cached - Buffers Where, MemTotal, MemFree, Cached, and Buffers are obtained from /proc/meminfo.

(g) Client: This collects memory information from DomainUs and periodically passes this information over to the Database. It is hosted by DomainUs and functions in the application layer. Memory and swap space information are gathered from the proc of DomainUs. These data are stored in Database with the APIs of XenStore. Client also collects the total and free MFNs of the physical machine.

(h) Server: As the core of the system, it acquires memory information from Database. It resides in Domain0 and functions in the application layer. The scheduling algorithm of Server then determines the domain that requires additional pages, as well as the domain that provides these extra pages. The scheduling algorithm also calculates the optimal target pages for allocation to each domain. According to the system state, different scheduling algorithms in Server may be utilized.

6 . Algorithm :Global-Scheduling Algorithm

The self-scheduling algorithm is applied if the free frames in the physical machine can satisfy the total pages requested by all VMs. In this case, the self-scheduling algorithm can directly map MFNs to GPFNs through the balloon driver for each domain. It also deploys a driver in the hypervisor, which monitors available frames in the physical machine and they trigger the global-scheduling algorithm when the available frames run out. The global-scheduling algorithm is utilized if the physical machine lacks free frames and cannot meet the total pages requested by all VMs. In this case, VMs compete for memory. The global-scheduling algorithm is used to over commit memory globally.

Summary of Key Notations

n	Number of VMs
N	Total memory of n VMs
f	Reserved free memory of VMs
N_i	Total memory of the ith VM, $1 \leq i \leq n$
N_{ti}	Target memory allocated to the ith VM, $1 \leq i \leq n$
F_i	Free memory of the ith VM, $1 \leq i \leq n$
A_i	Used memory of the ith VM, which equals to $(N_i - F_i)$, $1 \leq i \leq n$
A	$\{A_i \mid i = 1 \ldots . n\}$
$\bar{A}$	$\bar{A} = \frac{1}{n}\sum_{i=1}^{n} A_i$

The global-scheduling algorithm is used in the Server program of Domain0. In this algorithm, two sub-procedures, calculating-idle-memory-tax() and solve-linear-equation(), are es- sential. First, the total and free memory sizes for each VM are acquired using XenStore. Second, the parameter t (idle memory tax) is computed using the function calculating-idle-memory-tax().By solving the linear equations, we calculate the target memory (Nti) allocated to each VM. Finally, the target memory for each VM is sent to the balloon driver using Lib xen ctrl interfaces to reallocate the memory pages.

The *global-scheduling* algorithm is given as follows:

Algorithm 1. *Global-Scheduling* Algorithm

Input: N, n, N_i, A_i
Output: Nt_i

1. **while true do**
2. $A \leftarrow Null$
3. **for** $1 \leq i \leq n$ **do**
4. $N_i \leftarrow xs\_read$ (/local/domain/VM_i/mem/total);
5. $F_i \leftarrow xs\_read$ (/local/domain/VM_i/mem/free);
6. $A_i = N_i - F_i$;
7. $AppendTo(A, A_i)$;
8. **end**
9. $\tau \leftarrow calculating\_idle\_memory\_tax(A, f)$;
10. **for** $1 \leq i \leq n$ **do**
11. $Nt_i \leftarrow solve\_linear\_equation(N_i, A_i, \tau)$;
12. $xs\_write(Nt_i$, /local/domain/VM_i/mem/target);
13. $xc\_domain\_set\_pod\_target(VM_i, Nt_i)$;
14. **end**
15. sleep(*interval*);
16. **end**

7 Conclusion

The devise a system for automatic memory control based on the balloon driver in Xen Virtual Machines. System aims to optimize the running times of applications in consolidated environments by overbooking and/or balancing the memory pages of Xen Virtual Machines. Unlike traditional methods, such as MEB, our system is lightweight and can be completely integrated into user space without interfering with Virtual Machine Monitor operation. They also design a global-scheduling algorithm based on the dynamic baseline to determine the optimal allocation of memory globally.

8 Future Scope

We plan to extend our system to CPU and I/O devices in the future. Allows applications on multiple servers to share data without replication, decreasing total memory needs. Lowers latency and provides faster access than other solutions.

References

[1] A. Fox, R. Griffin, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "Above the clouds: A Berkeley view of cloud computing," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2009-28, p. 13, 2009,

[2] J. E. Smith and R. Nair, Virtual Machines: Versatile Platforms for Systems and Processes. Amsterdam, The Netherlands: Elsevier, 2005.

[3] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing performance isolation across virtual machines in Xen," in Proc. ACM Int. Conf. Middleware, 2006, pp. 342–362.

[4] P. Padala, G. S. Kang, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility computing environments," ACM SIGOPS Oper. Syst. Rev., vol. 41, no. 3, pp. 289– 302, 2007.

[5] W. Zhang, H. Zhang, H. Chen, Q. Zhang, and A. M. K. Cheng, "Improving the QoS of web applications across multiple virtual machines in cloud computing environment," in Proc. IEEE 26th Int. Parallel Distrib. Process.Symp. Workshops PhD Forum, 2012, pp. 2247–2253.

[6] W. Zhang, H. He,G. Chen, and J. Sun, "Multiple virtual machines resource scheduling for cloud computing," Appl. Math.Inf. Sci., vol. 7, no. 5, pp. 2089–2096, 2013.

[7] D. Magenheimer, "Memory overcommit. . . without the commitment," Xen Summit, pp. 1–3, 2008.

[8] Xen, the powerful opensource industry standard for virtualization. (2013).[Online]. Available: http://www.xenproject.org/