2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)
Dec 20-22, 2013, Shenyang, China

A Dynamic Memory Management Model on Xen Virtual Machine

Guilin Zhang, Huiqiang Wang, Hongwu Lv, Guangsheng Feng, Zhanbo He
College of Computer Science and Technology
Harbin Engineering University
Harbin, China
crosslandy@126.com

Abstract*—Xen virtualization has been widely used in cloud computing, data center and cyber physical systems. Because memory can not be used by time-sharing as other devices, how to efficiently allocate memory has become the bottleneck of performance improvement on Xen. In this paper, we propose a dynamic memory management model on Xen virtual machine with the goal of availability growth, which includes a fast memory adjustment strategy with memory recycling, memory increasing, and memory adjustment. Experimental results show that, in order to satisfy the memory requirements virtual machines need, this memory management model can recycle a large amount of free memory in virtual machines rapidly; in the aspect of increasing memory, using exponential growth mode can also satisfy high memory requirements in a short time, which has improved availability of the virtual machine requiring memory urgently; and in the case of shortage of memory resources, the model can also give priority to meet the demand of virtual machine requiring memory urgently, which has avoided availability falling due to shortage of memory resources.

Keywords-xen; virtual machine; memory management

I. INTRODUCTION

With the expansion of computer systems, virtualization technology has been widely used in cloud computing, data center and cyber physical systems[1][2]. As an important representative of the virtual technology, Xen virtualization technology allows multiple commodity operating systems to share conventional hardware in a safe and resource managed fashion, but without sacrificing either performance or functionality[3]. But as a special part, memory is usually statically allocated to each virtual machine [4], and size does not vary while the virtual machine is up running, which becomes a bottleneck in the performance of virtual machines.

To solve this problem, Xen Cloud Platform provide Dynamic Memory Control[5], which can change the amount of host physical memory assigned to any running virtual machine without rebooting it, but it cannot make a flexible response to a virtual machine which has a serious of complicated tasks. Zhao W [6] showed a prediction model. It uses two effective memory predictors, which respectively, estimate the amount of memory available for reclaiming without a notable performance drop, and additional memory required for reducing the virtual machine paging penalty. And Huazhong University[7] also presented a prediction model by analyzing historical Data of each virtual machine to relocate its memory.

This work is supported by the National Natural Science Foundation of China under Grant Nos. 61370212; Special Fund for Basic Scientific Research of Central Colleges under Grant Nos. HEUCFZ1213, HEUCF100601; the Research Fund for the Doctoral Program of Higher Education of China 20122304130002. the Natural Science Foundation of Heilongjang Province under Grant No. ZD 201102, F201217.

All these methods are based on forecast accuracy, therefore, a wrong prediction will lead to a negative effect. Besides, Zhang W[8] provided a method by using tax rate to relocate memory of different virtual machines, but it requires a complex calculation, which may cause additional overhead.

In this paper, we present a dynamic memory management model on virtual machines. The method of memory management in the model is designed by using threshold control, which takes the usage of memory under control in a limited range towards different guest virtual machines, instead of prediction mode. The framework developed on Xen combines initialization configuration module, memory monitoring module and memory allocation module, which could keep memory resources in a high efficient utilization among all the guest virtual machines.

The rest of this paper is organized as follows: Section 2 presents the architecture and the modules. Section 3 focuses on the policy of core module. The experimental results of memory management model are presented in Section4. And Section 5 concludes.

II. ARCHITECTURE OF MODEL

As illustrated in Fig. 1, the architecture based on Xen can be divided into three modules: initialization configuration module (ICM), memory monitoring module (MMM) and memory allocation module (MAM).

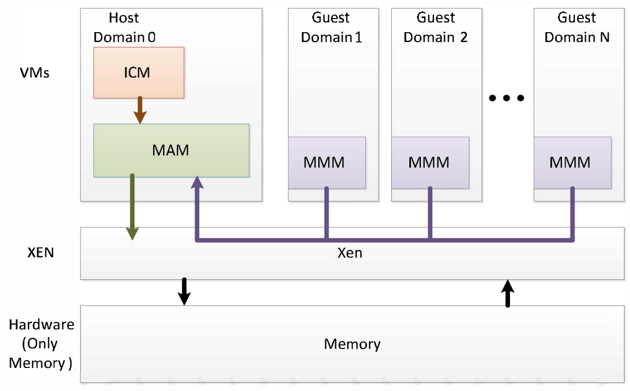

Figure 1. Architecture

(1)ICM: it is responsible for initializing the relevant parameters and a related configuration file when creating a new guest virtual machine. Each guest virtual machine has a configuration file, which includes $U_{\max}$ (the maximal memory utilization of guest domain) $U_{\min}$ (the minimal memory utilization of guest domain) and other parameters associated with it, in order to control its memory in a different range.

(2)MMM: it is responsible for monitoring the memory information, which consists of three parts: VM_{id} (the ID of guest domain), M_{total} (the total memory of guest domain) and U_{used} (the memory utilization of guest domain) in each guest virtual machine. It sends data to MAM which is in host domain by using XenStore.

(3)MAM: it is the core of this model by using fast memory allocation strategy (FMAS) to manage all the guest virtual machines` memory. MAM will recycle a certain amount of memory from a guest domain, when there is quite a lot of free memory in that guest domain, and it will increase a guest domain's memory in an opposite situation. Even in the absence of usable memory, MAM will squeeze some memory from guest domains whose memory utilization is in a low level but still in the rational range. The details of FMAS are shown in next section.

III. FAST MEMORY ALLOCATION STRATEGY

Different guest domain always has different requirement about memory in the usual. FMAS can control a guest domains` memory in a stable range, so that it can reduce the bad influence caused by the frequent memory adjustment. The flowchart of FMAS is shown in Fig. 2.

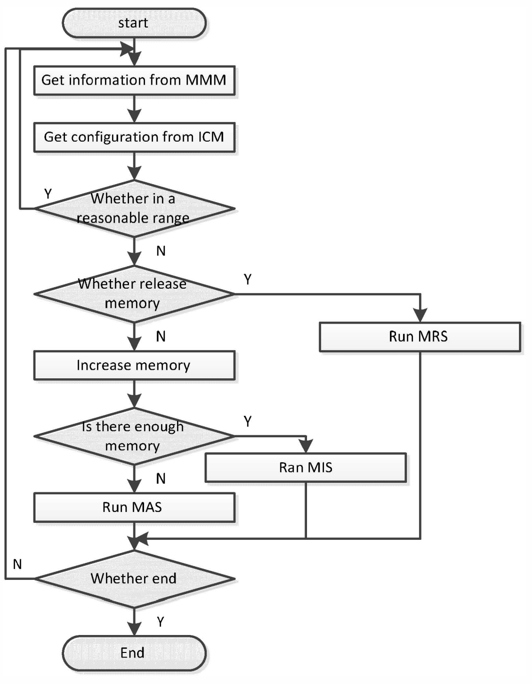

Figure 2. Flowchart of FMAS

Step1. Get memory information of guest domain i from MMM. The information consists of three parts: $VM_i, i = 1..N$ (VM_i is the ID of guest domain i, N is the number of all the guest domains) M_{i_total} (M_{i_total} is the total memory of guest domain i), and U_{i_used} (U_{i_used} is the memory utilization of guest domain i).

Step2. Get the maximum memory utilization $U_{i\_\max}$ and the minimum memory utilization $U_{i\_\min}$ from VM_i `s configuration file provided by ICM. If $U_{i\_\min} \le U_{i\_used} \le U_{i\_\max}$, do nothing and take a short sleep, then go back to step 1, else go to step 3.

Step3. If $U_{i\_used} < U_{i\_\min}$, means there are a plenty of free memory needs to be recycled, then MAM will run the Memory Recycling Strategy (MRS). Else go to step 4.

Step4. If $U_{i\_used} > U_{i\_\max}$, means VM_i is lacking of memory, then MAM will increase the guest domain's memory. At this point, there are two possible situations: if there is enough physical memory for VM_i, then run Memory Increasing Strategy (MIS), else go to step 5.

Step5. In this step, it means that the available memory can not meet the current needs of VM_i, so it is necessary to adjust the memory allocation for the other guest domains. Then run Memory Adjustment Strategy (MAS).

Step6. After executing above process, MAM will return to step1 for the next round.

Next, Memory Recycling Strategy (MRS), Memory Increasing Strategy (MIS) and Memory Adjustment Strategy (MAS) will be discussed in detail.

A. Memory Recycling Strategy

MRS is responsible for releasing the extra memory in VM_i, using direct release mode, as shown in figure 3.

```
Algorithm: Memory Recycling
Input:  $U_{i\_\max}$  $U_{i\_\min}$  $M_{i\_total}$  $U_{i\_used}$
Output: $M_{i\_new}$ //new total memory of $VM_i$
begin
    Get $U_{i\_\max}$ and $U_{i\_\min}$
    Calculate the new utilization of $VM_i$ do
        $U_{i\_new} = (U_{i\_\min} + U_{i\_\max})/2$
    Calculate the new total memory of $VM_i$ do
        $M_{i\_new} = (M_{i\_total} * U_{i\_used})/U_{i\_new}$
    Set a new memory for $VM_i$
end
return $M_{i\_new}$
```

Figure 3. Algorithm of MRS

Firstly, MRS gets the $U_{i\_\max}$ and $U_{i\_\min}$ from the configuration file of VM_i. And then calculate the new utilization U_{i_new}, which is the arithmetic mean value of $U_{i\_\max}$ and $U_{i\_\min}$.

Secondly, MRS calculates a new value of total memory M_{i_new} with the parameter M_{i_total} U_{i_used} and U_{i_new} as presented in figure 3. Finally, MRS executes the system call of Xen to set a new memory for VM_i.

For example, suppose that the current total memory of VM_i is 512MB, the memory utilization now is 60%, the bound of utilization is $U_{i\_\min} = 70\%$ and $U_{i\_\max} = 90\%$. With the algorithm of MRS the new memory of VM_i should be $M_{i\_new} = (512 * 0.6)/0.8$, and it's 384MB.

B. *Memory Increasing Strategy*

MIS is responsible for increasing the memory allocation of VM_i when needed, using exponential growth mode, which is similar to the sliding window of TCP congestion control as shown in Fig.4.

```
Algorithm: memory increasing
Input:   m, $M_{free}$ $M_{i\_total}$
Output: $M_{i\_new}$
begin:
    if the first time to increase then
        m=init
        else m=m+1
    end
    Set expected incremental memory $2^m$
    if $M_{free} < 2^m$ then
        $M_{i\_add} = M_{free}$
        else $M_{i\_add} = 2^m$
    end
    Calculate a new memory $M_{i\_new} = M_{i\_total} + M_{i\_add}$
end
return: $M_{i\_new}$
```

Figure 4. Algorithm of MIS

Firstly, Judge whether it is the first time to increase the memory of VM_i, if so set the index factor $m = init$, else $m = m + 1$.

Secondly, set the expected incremental memory equals to 2^m MB, and judge whether there is enough physical memory in the machine, if $M_{free} < 2^m$, then the increasing memory will be M_{free}, else it will be 2^m (M_{free} free memory in physical machine).

Finally, execute the system call of Xen to set a new memory for VM_i.

C. *Memory Adjustment Strategy*

MAS is responsible for adjusting memory among all the guest domains when there is a guest domain is lacking of memory and there is no free memory in the physical machine. The algorithm of MAS is shown in figure 5.

When it is in MAS, it means there is no free memory in physical machine anymore, and VM_i is lacking of memory at this time. So MAS will squeeze memory from the other guest domains whose memory utilization is in a low level but still in the rational range.

Firstly, for all the guest domains, MAS will set a new memory utilization $U_{j\_present} = U_{j\_\max}$ ($U_{j\_\max}$ is the maximal memory utilization of VM_j, and $j = 1..N, (j \neq i)$), except VM_i which is lacking of memory.

Secondly, calculate a new memory allocation for VM_j. It is:

$$M_{j\_changed} = \frac{M_{j\_total} * U_{j\_used}}{U_{j\_present}} + \Delta M \qquad (1)$$

In the formula 1, M_{j_total} is the allocated total memory of VM_j, and U_{j_used} is the present utilization of VM_j, $U_{j\_present}$ is a new utilization of VM_j which has been got in the first step, and ΔM is an additional increment of memory which will prevent a new memory utilization approaching to its maximum value, and then causes another memory adjustment.

Finally, execute the system call of Xen to set a new memory for all the VM_j which allowed to be changed.

```
Algorithm: Memory Adjustment
Input:   $M_{j\_total}$, $U_{j\_\max}$, $U_{j\_used}$, $\Delta M$, $M_{total}$
Output: $M_{free}$
begin
    if $M_{free} = 0$ then
        foreach $j \in 1..N$, $j \neq i$ do
            Get $VM_j$ info from $VM_j$ config file
            Calculate the new memory of $VM_j$
            Set a new memory for $VM_j$
        end
        reset $M_{free}$
    end
end
return $M_{free}$
```

Figure 5. Algorithm of MAS

IV. Implementation and Performance Analasys

The dynamic memory management model is implemented on Xen4.0 on a Sugon server. The server has 16 cores and 4GB 133MHZ Dual Ranked memory. There are 5 domains running in the server, one is host domain and the others are guest domains. Linux 3.6.11 runs on all the domains as their OS. As shown in figure 1, MMM has been deployed in guest domain, ICM and MAM run in host domain.

To exclude the impact of CPU resource contention when multiple guest domains are running, each domain is assigned a dedicated CPU core. The maximum memory limit for each guest domain is 1GB and the minimum memory limit is 256MB which means the memory of a guest domain will not be larger than1GB or less than 256MB. In order to simplify the experiment the maximum and minimum utilization are set to 90% and 70%, and the initial value of index factor m is 4. For the host domain, its memory is fixed on 2GB during the whole experiment.2

A. *Appropriate Using of Physical Memory*

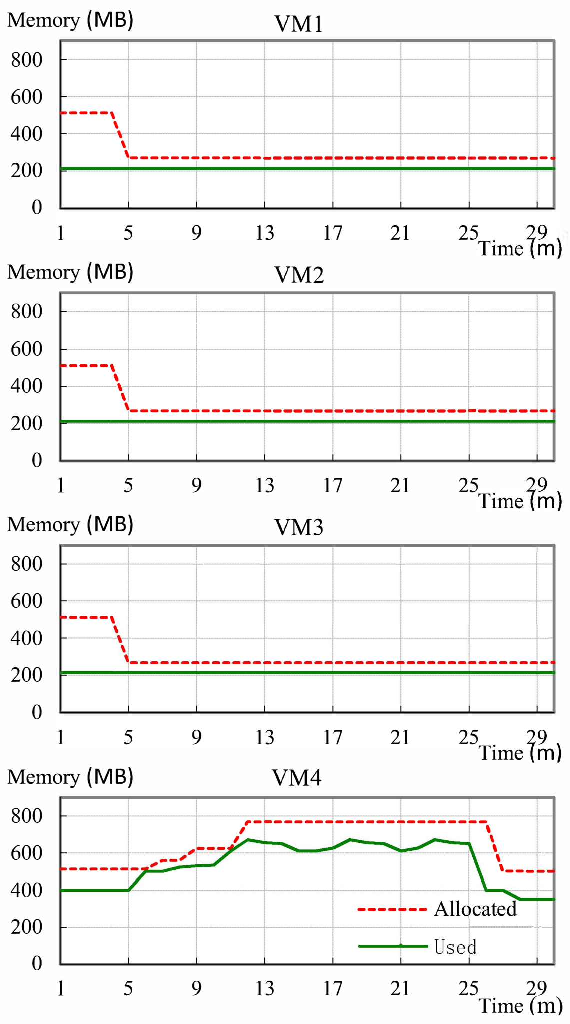

Figure 6. Appropriate Using of Physical Memory

In this experiment, there are four guest domains running with an initial memory of 512MB. VM1 VM2 and VM3 only run an operating system without any task, and VM4 runs a task of compiling Linux kernel. In the running virtual machines, there is 298MB memory unoccupied in each empty load guest domain. For VM4, there is not too much memory unoccupied. That will be a giant waste of memory without any relocation.

It will be more appropriate by using FMAS as shown in figure 6. Starting from the fifth minute, FMAS is activated. And each empty load guest domain's memory reduces to 268MB by MRS, so that their memory utilization can be maintained in a reasonable range. And for VM4, FMAS will keep increasing VM4's memory until the memory utilization of VM4 is less than 90%. In this experiment, the increasing sequence of VM4 is 16MB, 32MB, 64MB and the last increase is 128MB, after that, VM4's memory will be under control. In the minute of 26 the task is over, then the allocation memory of VM4 will return to a low level, but still reasonable.

B. *High Availability under Huge Memory Consuming*

FMAS will show its high availability in the situation of huge memory consuming, in this experiment. We design a huge memory consuming task to test the performance of FMAS by using memtester4 as shown in figure 7. The upper figure is a guest domain without FMAS and the under figure is with it. And the initial memory of the two guest domain is 256MB.

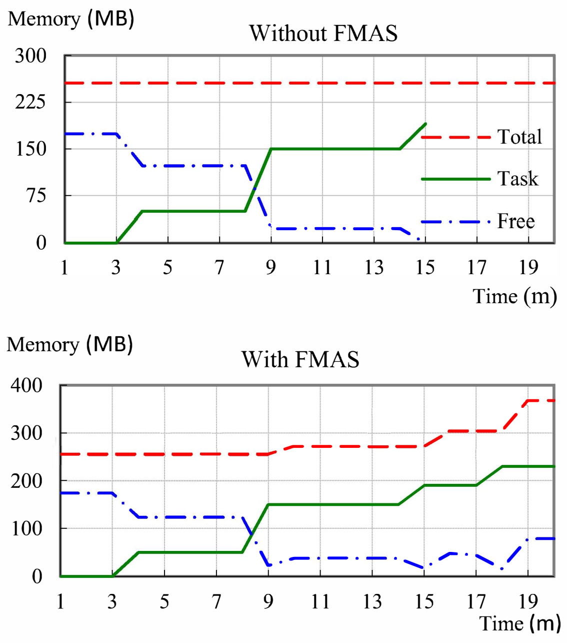

Figure 7. High availability under huge memory consuming

The free memory in both domain is become less with the increase of memory consumption. Starting from the third minute, the huge memory consuming task has been eaten 50 MB memories, and the two guest domain is still in a good condition. In the minute of 5, memory task has been consumed 150 MB, and the utilization of both guest domain is greater than 90%, so the one with FMAS will increase memory immediately to a reasonable range, in order to allow more tasks to run on it. However, in the minute of 15, the task has eaten 190MB memories, the guest domain that without FMAS is out of memory and then halt down, but the one with FMAS can still run, even in the minute of 18 when the memory task has consumed over 230MB memories.

C. *Performance of MAS*

As shown in figure 8, there are four guest domains running with an initial memory of 512MB. The memory utilization of VM1, VM2 and VM3 are kept in a stable range of 85%, 75%, and 89%, which are all in the reasonable range, running different tasks. However, VM4 is lacking of memory with a high memory utilization of 98% at this moment. But there is no enough physical memory for it, because the total memory for the guest domains is only 2GB. Therefore, MAS will squeeze the memory from the guest domains whose memory utilization is in a low level but still reasonable. In this experiment, the additional increment of memory ΔM is set to 10MB.

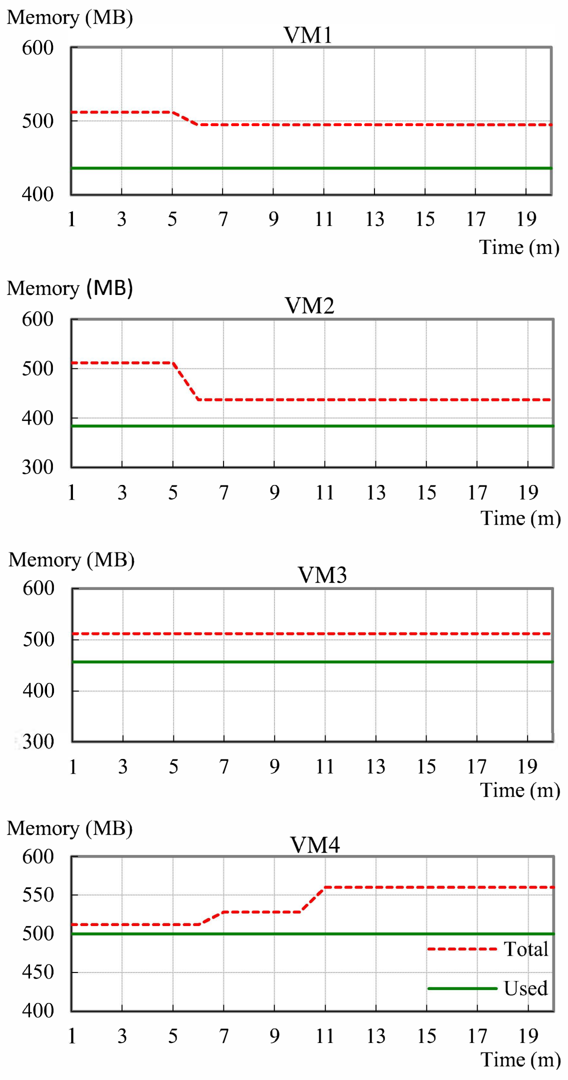

Figure 8. Performance of Memory Adjustment

The process of memory adjustment is shown as figure 8. Starting from the 6 second, FMAS is activated. The allocated memory of VM1 and VM2 reduce to 495MB and 437MB. And nothing happened to VM3, because the memory utilization of VM3 is too close to its upper bound. It may cause a new round of memory adjustment if we keep reducing VM3's memory. After the adjustment, MAS retrieve 92MB physical memory for VM4 which is lacking of memory. And then MIS will take over the rest. As this experiment shows, MAS isn't a blinding "squeeze" but according to the specific circumstances of the guest domains.

V. CONCLUSION AND FUTURE WORK

In this paper, we present a dynamic memory management model on Xen virtual machine with the goal of availability growth, which includes a fast memory adjustment strategy with memory recycling, memory increasing, and memory adjustment. Our memory management model provides a flexible memory space for different guest domains depending on the tasks they are responsible for. The MIS in our model will meet the memory demand rapidly by using exponential growth mode in several times and MDS will recycle the unused memory only in one time, which could reduce the number of memory adjustment. Furthermore, the MAS in our model will squeeze memory from other guest domains whose memory utilization is in a low level but still in rational range, which will improve the availability of virtual machine.

In the next step of work, we plan to add the model to the open source Xen, to improve its performance and universality.

REFERENCES

[1] Ya-Qiong L, Ying S, Yong-Bing H. A memory global optimization approach in virtualized cloud computing environments[J]. Chinese Journal of Computers, 2011, 34(4): 684-693.

[2] Rochwerger B, Breitgand D, Levy E, et al. The reservoir model and architecture for open federated cloud computing[J]. IBM Journal of Research and Development, 2009, 53(4): 4: 1-4: 11.

[3] Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization[J]. ACM SIGOPS Operating Systems Review, 2003, 37(5): 164-177.

[4] Dan Magenheimer. Memory Overcommit without the commitment. Boston, USA: Xen Summit, 2008.

[5] Tosatto D. Citrix XenServer 5.6 Administration Guide[M]. Packt Publishing, 2012.

[6] Zhao W, Wang Z, Luo Y. Dynamic memory balancing for virtual machines[J]. ACM SIGOPS Operating Systems Review, 2009, 43(3): 37-47.

[7] Huazhong University of Science and Technology. Dynamic memory management system based on memory hot plug for virtual machine. Application 201110162615[P]. 2011-6-16.

[8] Zhang W, Cheng T, He H, et al. LVMM: A lightweight virtual machine memory management architecture for virtual computing environment [C] Uncertainty Reasoning and Knowledge Engineering (URKE), 2011 International Conference on. IEEE, 2011, 1: 235-238.