Memory virtualization overhead mitigation using contiguous memory virtual machines

Peterson Yuhala

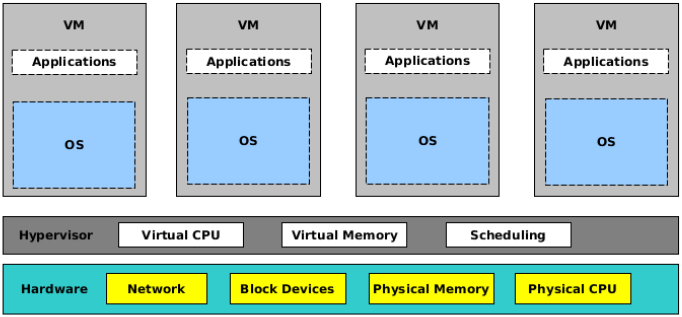
Supervised by : Prof. Alain Tchana

University of Neuchâtel - June 2018

Introduction

- Address translation represents a significant part of virtualization overhead [ISCA'16,ISCA'17].
- Our main goal is to nullify that overhead in virtualized systems.

Virtualized System Architecture



Memory virtualization

- Memory virtualization depends on VM type. Three main mechanisms exist.

Memory virtualization

- Memory virtualization depends on VM type. Three main mechanisms exist.
- In paravirtualised systems, VMs use a technique called direct paging [Xen'15].
- Here the guest uses a Physical to Machine mapping maintained within the hypervisor to map virtual addresses directly to physical addresses in its page table [Xen'15].

Memory virtualization

- Memory virtualization depends on VM type. Three main mechanisms exist.
- In paravirtualised systems, VMs use a technique called direct paging [Xen'15].
- Here the guest uses a Physical to Machine mapping maintained within the hypervisor to map virtual addresses directly to physical addresses in its page table [Xen'15].
- For hardware assisted virtualised systems, we have two main techniques used: shadow paging and nested paging.

Memory virtualization

- Memory virtualization depends on VM type. Three main mechanisms exist.
- In paravirtualised systems, VMs use a technique called direct paging [Xen'15].
- Here the guest uses a Physical to Machine mapping maintained within the hypervisor to map virtual addresses directly to physical addresses in its page table [Xen'15].
- For hardware assisted virtualised systems, we have two main techniques used: shadow paging and nested paging.
- Our main focus here is hardware assisted virtualised systems as HVMs are becoming the norm [Amazon AWS].

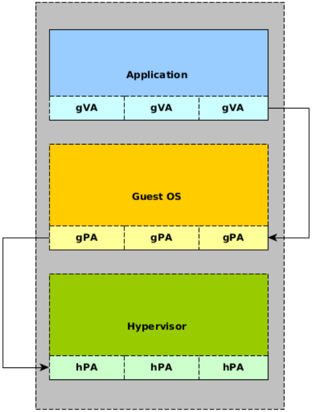
Main Issue

- The main issue here is Memory Address Translation in vitualized systems. ie Obtaining the corresponding machine address in host address space given a virtual address in guest address space.

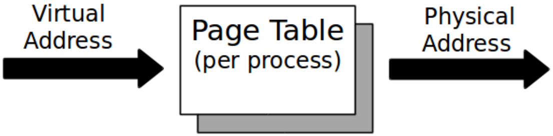
Institut de Recherche
en Informatique
de Toulouse

Paging
Basic Definitions

- Main memory addresses are called host physical addresses (hPA) and process addresses are called guest virtual addresses (gVA). Addresses in the guest address space are referred to as guest physical addresses (gPA).



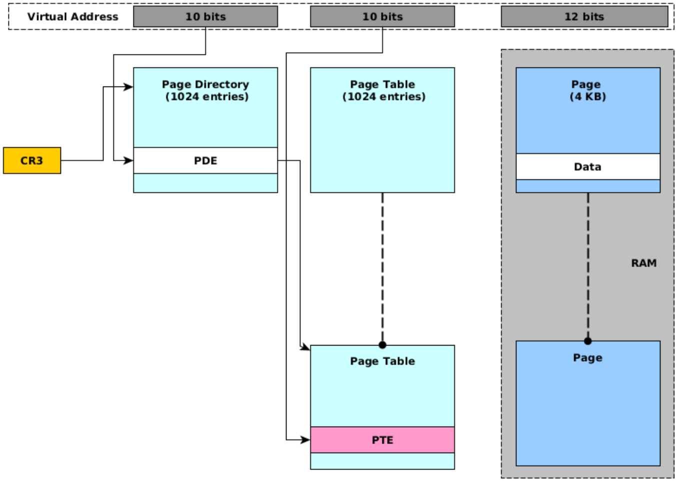
- Native systems (No Virtualization) require one-level address translations. ie virtual address $\rightarrow$ physical address.



Address Translation

Native Systems

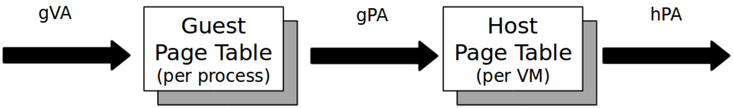
- The following figure illustrates the page walking mechanism in a two-level page table.



Address Translation

Virtualized Systems

- Virtualized systems on the other hand require two-level address translations. ie gVA $\rightarrow$ gPA AND gPA $\rightarrow$ hPA.



- Two approaches are used : **Shadow Paging** and **Nested Paging**.

Shadow Paging

- In shadow paging the hypervisor maintains a shadow page table for gVA $\rightarrow$ hPA mappings and the **CR3** register points to the shadow page table.
- The hypervisor captures any attempts by the VM to update its page tables and uses gVA $\rightarrow$ gPA and gPA $\rightarrow$ hPA to construct the shadow page table.

Shadow Paging

- In shadow paging the hypervisor maintains a shadow page table for gVA $\rightarrow$ hPA mappings and the **CR3** register points to the shadow page table.
- The hypervisor captures any attempts by the VM to update its page tables and uses gVA $\rightarrow$ gPA and gPA $\rightarrow$ hPA to construct the shadow page table.
- On a TLB miss, the hardware performs a 1D page walk on the shadow page table to obtain the machine address corresponding to a guest virtual address.

Shadow Paging
Advantages

- Only 4 memory references (same as base native) required for a complete page walk on a TLB miss [ISCA’16].

Shadow Paging
Advantages

- Only 4 memory references (same as base native) required for a complete page walk on a TLB miss [ISCA'16].
- No need for any extra hardware support for page walks.

Shadow Paging
Disadvantages

- Every page table update requires a costly trap in the hypervisor to update shadow page table entries [ISCA'16].

Shadow Paging
Disadvantages

- Every page table update requires a costly trap in the hypervisor to update shadow page table entries [ISCA'16].
- Costly traps in the hypervisor on context switches [ISCA'16].

Nested Paging

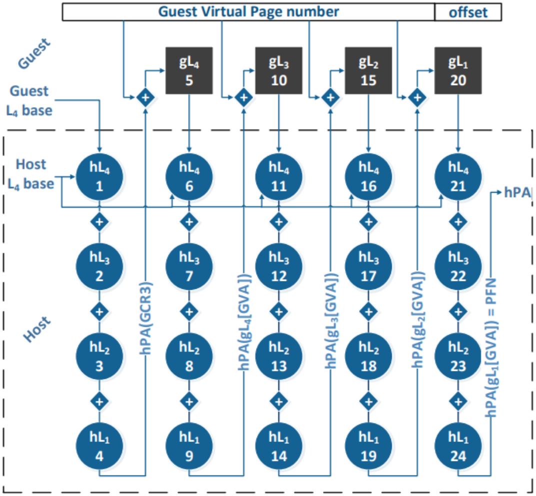
- Nested paging is a widely used hardware technique to virtualize memory and here the MMU a two dimensional page table consisting of the **guest page table** (gPT) and **host page table** (hPT).

Nested Paging

- Nested paging is a widely used hardware technique to virtualize memory and here the MMU a two dimensional page table consisting of the **guest page table** (gPT) and **host page table** (hPT).
- On a TLB miss the processor performs a 2D page walk for any gVA $\rightarrow$ hPA translation.

Nested Paging

2D page walk



Figure

Nested Paging
Advantages

- Allows fast direct updates to both page tables (gPT and hPT) without any traps in the hypervisor (ie no VMM intervention).

Nested Paging
Disvantages

- On a TLB miss the MMU performs a 2D page walk that requires up to 24 memory references. Figure 1 shows how up to 24 memory references can be obtained for a complete 2D page walk.

Trade Offs

Summary

	Base Native	Nested Paging	Shadow Paging
TLB Hit	fast	fast	fast
Max. memory access on TLB miss	4	24	4
Page table updates	fast direct	fast direct	slow mediated by VMM
Hardware support	1D page walk	2D page walk	1D page walk

Our Solution

- From the table, Nested and Shadow paging both present significant overhead due to memory accesses and page table updates respectively.

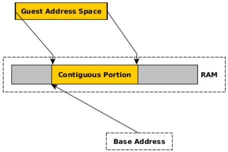
Our Solution

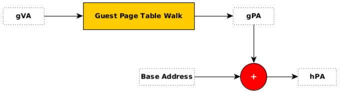
- From the table, Nested and Shadow paging both present significant overhead due to memory accesses and page table updates respectively.
- We propose a new mechanism of address translation which maintains the advantages of both Nested and Shadow paging, and at the same time nullifies the overhead of both mechanisms.

Our Solution

Basic Idea

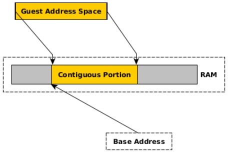
- Our basic idea is to give contiguous machine memory to VMs and introduce a hardware feature to store the base address of the **contiguous chunk**.

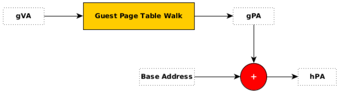




Our Solution
Basic Idea

- Our basic idea is to give contiguous machine memory to VMs and introduce a hardware feature to store the base address of the contiguous chunk.





- Using our solution, the **gPT** and the **base address** are sufficient for a complete gVA $\rightarrow$ hPA translation.

Our Solution

Comparison of Trade Offs

	Base Native	Nested Paging	Shadow Paging	Our Solution
TLB Hit	fast	fast	fast	fast
Max. memory access on TLB miss	4	24	4	4
Page table updates	fast direct	fast direct	slow mediated by VMM	fast direct
Hardware support	1D page walk	2D page walk	1D page walk	1D page walk

Proof of Concept
Simulator

- We made a **simulator** to show that it is possible to give contiguous memory to a high percentage of VMs in a datacenter.
- We simulated VM traces from Microsoft Azure [SOSP'17] and Bitbrains.
- The VM traces contain information about VM creation and destruction timestamps, CPU utilization and VM memory.

Proof of Concept
Simulator

- The simulator simulates 2 VM placement algorithms :
  - **Traditional placement algorithm** which takes into account resource availability (spread and stack).
  - **Contiguity-Aware placement algorithm** which prioritizes servers which can provide contiguous memory for VMs.

Proof of Concept
Simulator

- The simulator simulates 2 VM placement algorithms :
  - **Traditional placement algorithm** which takes into account resource availability (spread and stack).
  - **Contiguity-Aware placement algorithm** which prioritizes servers which can provide contiguous memory for VMs.
- It takes as input VM traces and then calculates the percentage of VMs with contiguous memory at the end of the simulation.

Proof of Concept
Simulator

- We use different generations of MS Azure servers in our simulator. ie. generation 3 to 6 and HPC servers so as to reflect the reality in real datacenters.
- A mix of servers with the following composition was used in our simulator :

Gen.	**%**
Gen3	20
HPC	20
Gen4	20
Gen5	20
Gen6	10
Godzilla	10

Proof of Concept

Simulation results

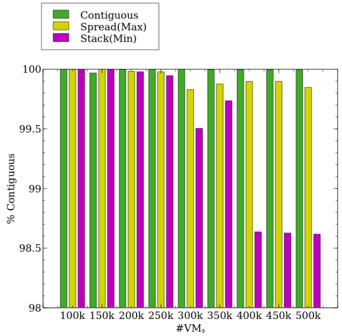


Figure : % contiguous-memory VMs using different placement algorithms

Proof of Concept

Simulation results

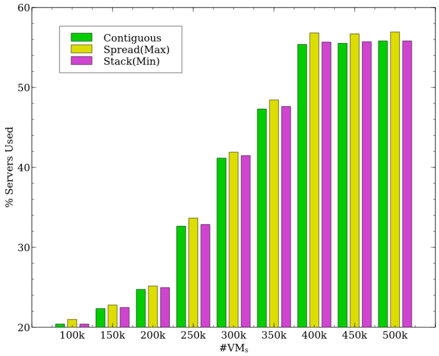


Figure : Server consolidation rate for both contiguous-aware and traditional placement algorithms.

Simulation Results

- We have a high % of VMs with contiguous memory because VMs sizes are relatively small compared to server sizes [SOSP'17].
- The relatively small and discrete size of VMs reduces the impact of memory fragmentation and prevents resource sprawl in the datacenter.

Implementation
Xen

- We implement our solution in Xen hypervisor. Here we modify the default Xen memory allocator so it allocates contiguous chunks of machine memory to VMs.

Implementation
Xen

- We implement our solution in Xen hypervisor. Here we modify the default Xen memory allocator so it allocates contiguous chunks of machine memory to VMs.
- To provide contiguous memory to a VM in Xen, we calculate the **order** corresponding to the VM's memory demand. ie **VM Total Memory =** 2^{order} *** 4 KB** .
- This order is then passed to the **Xen heap allocator** which provides a contiguous chunk of memory containing 2^{order} contiguous 4kb pages.

Next Steps

- We succeed already in giving contiguous memory to PV VMs. We would then eliminate unneccessary hypercalls knowing that VMs have contiguous memory.

Next Steps

- We succeed already in giving contiguous memory to PV VMs. We would then eliminate unneccessary hypercalls knowing that VMs have contiguous memory.
- We succeed in mapping 99% of HVM memory to contiguous machine memory. We would modify the Xen heap allocator to give 100% contiguity. The memory translation algorithm would then be modified as per our basic idea.

Next Steps

- We succeed already in giving contiguous memory to PV VMs. We would then eliminate unneccessary hypercalls knowing that VMs have contiguous memory.
- We succeed in mapping 99% of HVM memory to contiguous machine memory. We would modify the Xen heap allocator to give 100% contiguity. The memory translation algorithm would then be modified as per our basic idea.
- The last step would be the implementation of the whole solution in a simulator and processing of final results.

Challenges

- **Memory isolation** : Each VM must be confined within its memory region. How do we ensure isolation knowing that the hypervisor has minimal interference.

Challenges

- **Memory isolation** : Each VM must be confined within its memory region. How do we ensure isolation knowing that the hypervisor has minimal interference.
- How to **enforce contiguity**.

Challenges

- **Memory isolation** : Each VM must be confined within its memory region. How do we ensure isolation knowing that the hypervisor has minimal interference.
- How to **enforce contiguity**.
- **Cohabitation** of our solution with Nested Paging or Shadow Paging. ie Contiguous memory VMs use our solution and other VMs use Nested Paging/Shadow Paging.

Recap of our contributions

Memory Isolation

Memory Isolation :

- Introduction of a base address (**gBA**) for each VM (included within the VM Control Structure (VMCS)).

Recap of our contributions
Memory Isolation

Memory Isolation :

- Introduction of a base address (**gBA**) for each VM (included within the VM Control Structure (VMCS)).
- Once scheduled, the VM's **gBA** and VM's memory size (**gMS**) are stored within dedicated processor registers.

Recap of our contributions

Memory Isolation

Memory Isolation :

- Introduction of a base address (**gBA**) for each VM (included within the VM Control Structure (VMCS)).
- Once scheduled, the VM's **gBA** and VM's memory size (**gMS**) are stored within dedicated processor registers.
- To ensure isolation, the TLB is enhanced with a **security module** such that during insertion into the TLB, the security module checks if $hPA \leq gBA + gMS$

Recap of our contributions

Memory Contiguity Enforcement on a server

Memory Contiguity :

- At boot time, we allocate contiguous memory to VMs if possible.

Recap of our contributions
Memory Contiguity Enforcement on a server

Memory Contiguity :

- At boot time, we allocate contiguous memory to VMs if possible.
- For VMs with discontinuous memory, the hypervisor reorganizes the VM's memory to be contiguous. This can be done either by migrating the VM's pages during idle periods or by migrating the VM to another host.

Conclusion
Take Away Message

- Memory virtualization overhead can be very high when using traditional techniques (Nested and Shadow Paging).

Conclusion
Take Away Message

- Memory virtualization overhead can be very high when using traditional techniques (Nested and Shadow Paging).
- We propose to eliminate the overhead of these traditional techniques using contiguous-memory VMs.

Conclusion

Take Away Message

- Memory virtualization overhead can be very high when using traditional techniques (Nested and Shadow Paging).
- We propose to eliminate the overhead of these traditional techniques using contiguous-memory VMs.
- Through simulation we showed that it is possible to give contiguous memory to a high % of VMs in datacenters.

Conclusion
Take Away Message

- Memory virtualization overhead can be very high when using traditional techniques (Nested and Shadow Paging).
- We propose to eliminate the overhead of these traditional techniques using contiguous-memory VMs.
- Through simulation we showed that it is possible to give contiguous memory to a high % of VMs in datacenters.
- Our solution is being prototyped in the Xen hypervisor; we would then evaluate its performance to show that it is indeed better than Nested and Shadow paging.

Thank you for listening !