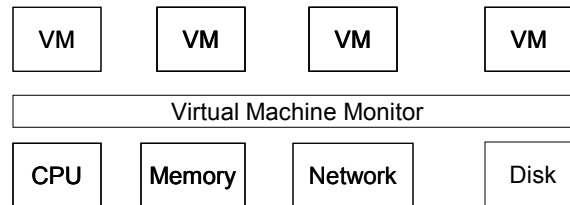


Virtualization – Resource Management

Virtual Machine Monitor

- A thin software layer providing the virtualization
- Designed to multiplex hardware resource efficiently among VMs
- Provide the illusion that a virtual machine completely owns hardware resources
- Also called hypervisor
- VMware ESX server as an example
 - Virtualizes Intel IA-32 architecture
 - On bare hardware
 - Binary translation – privileged code trapped
 - No need to change guest OS

Virtual Machine Monitor



- Manages the physical resources on the hardware:
 - Memory
 - Physical processors
 - Storage controllers
 - Networking
 - Keyboard, video, and mouse
- Includes schedulers for CPU, memory, and disk access, and has full-fledged storage and network stacks.

VMM Resource Management – VMware ESX Server

- Two most important resources
 - CPU
 - Memory

CPU Scheduling

- The execution context for scheduling
 - On conventional operating systems, a process or a thread
 - For VMM, the execution context is a guest OS (VM).
- Guarantee each VMs a certain share of CPU
 - Fairness is important
- Proportional-share based scheduling

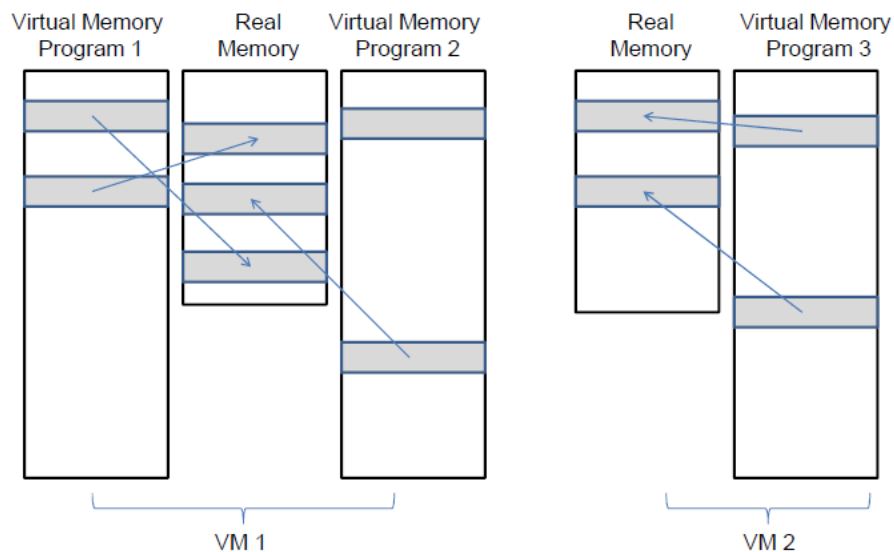
Proportional-Share Based Algorithm

- Different from conventional OS (e.g., Unix)
 - UNIX CPU scheduler uses a priority-based scheme
 - Priority is given by user
- CPU consumption / Share is used as the priority
 - Less than 1 : high priority, larger than 1 : low priority
 - Priority is determined by the VMM
 - VMs with larger share will get more CPU
- Accurately control the CPU allocation of VMs
 - VM0's share: VM1's share = 2:1
 - E.g. VM0 gets twice more CPU than VM1

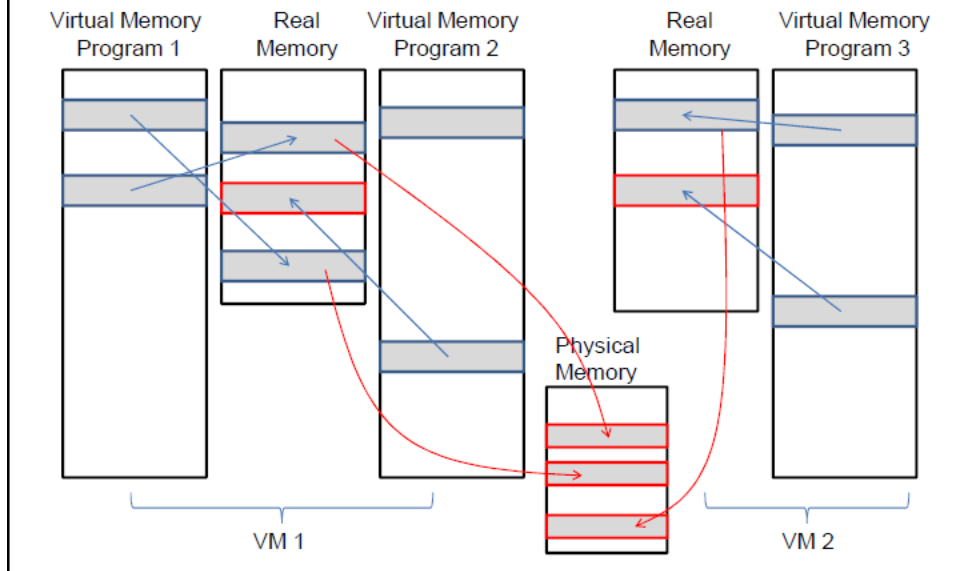
Memory Management

- Each VM is allocated memory based on configured memory size
- *Reservation*
 - Guaranteed lower bound on the amount of memory
- *Limit*
 - Upper limit on memory
- *Shares*
 - Specify the relative priority for a virtual machine
 - VMM determines allocation of memory (additional to reservation) to each VM based on shares

Memory Virtualization



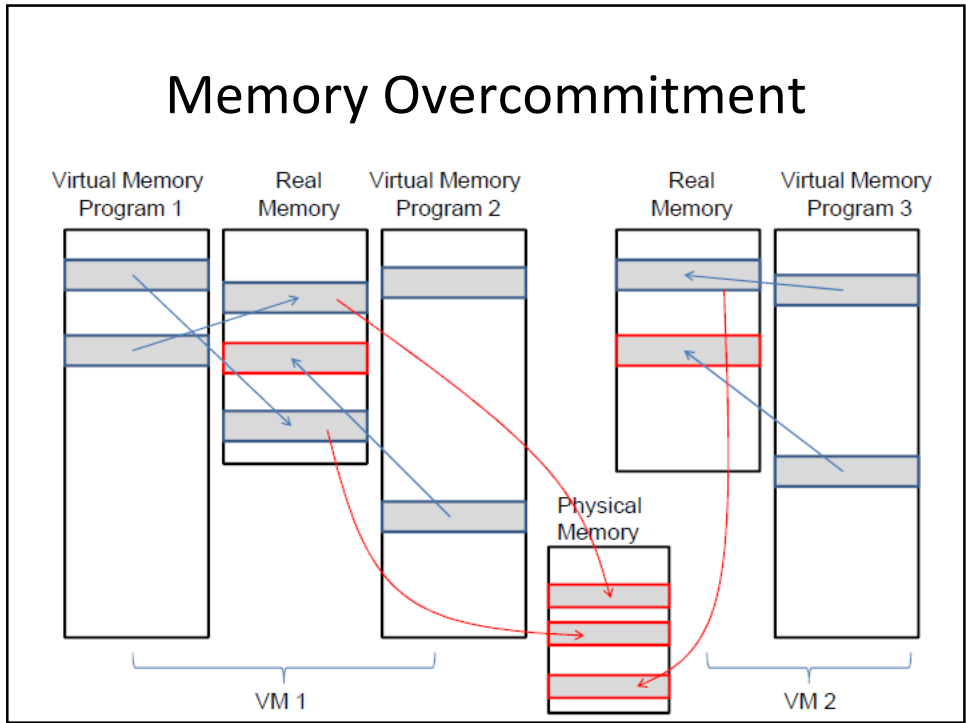
Memory Virtualization



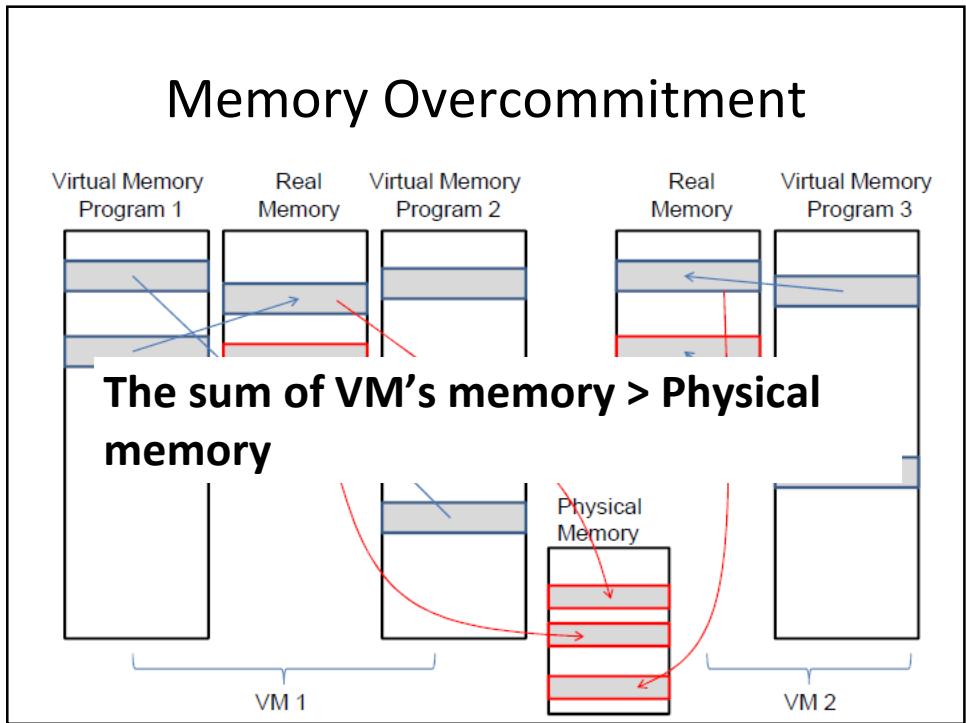
Memory Overcommitment

- Some VMs are lightly loaded while others are more heavily loaded, and relative activity levels vary over time
 - The sum of instantaneous memory usage is relatively smaller than the sum of maximum memory usage
- Use overcommitment
 - Allocate more memory for VMs than physical memory
 - For example, you can have a host with 2GB memory and run four virtual machines with 1 GB memory each configured
- Effective for maximizing memory use
 - Can run more VMs

Memory Overcommitment



Memory Overcommitment



Memory Reclamation

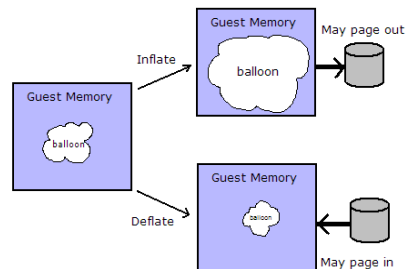
- Instantaneous memory usage change
 - Memory requirement for VMs changes over time
- Transfers memory from idle virtual machines to virtual machines that need more memory
 - Clear memory page prior to the reallocation
 - Enforcing the isolation between VMs.
- How to do memory reclamation ?

Memory Reclamation

- Traditional: add transparent swap layer
 - VMM choose memory pages to claim
 - Need page replacement decisions
 - But best decision can be made by guest OS
 - Guest and meta-level policies may clash
 - What happens when a guest OS tries to page out memory pages already swapped out by VMM?
- Alternative: implicit cooperation
 - Coax guest into doing page replacement

Balloon Driver

- Goal:
 - Do the memory reclamation with no information from VM OS
 - Let VM OS choose victims
- A balloon module or driver is loaded into VM OS
- Memory owned by balloon driver never swapped out
 - The balloon works on pinned real memory pages in the VM
- “Inflating” the balloon reclaims memory
- “Deflating” the balloon releases the allocated pages



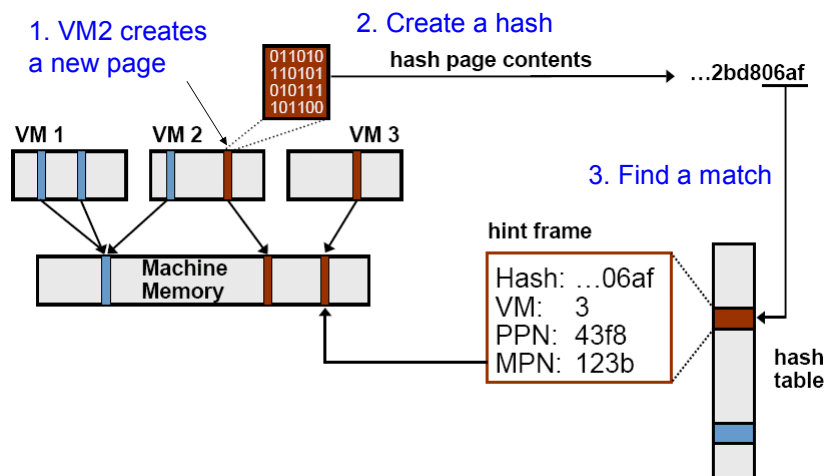
Memory Sharing

- Opportunities for sharing memory across virtual machines.
 - Several VMs may be running instances of the same guest operating system
 - Have the same applications or
 - Contain common data.
- Exploit the redundancy of data and instructions across several VMs
 - Reduce total memory usage
 - Increase the level of over-commitment available for the VMs: run more VMs!

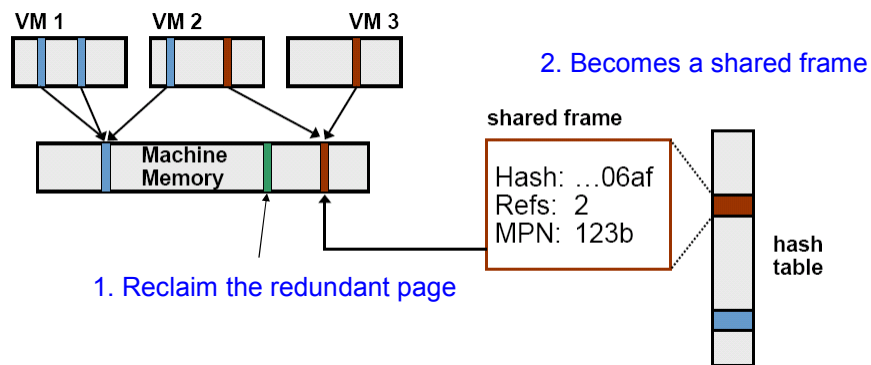
Page Sharing

- Compare page content to implement sharing
 - Does not need to modify guest OS to work
- A hash value is used to summarize page content
 - Reduce scan comparison complexity
- A hint entry for not yet shared pages
- COW (copy-on-write) when a shared page is written to

Page Sharing – Scan



Page Sharing – Successful Match



Proportional Allocation

- Memory is allocated to VMs through shares
 - Share specifies the relative priority or importance of a VM
- VMs with more shares get more memory
- Determines allocation of additional memory
- Pure proportional-share can be inefficient. Why?

Proportional Allocation

- Memory is allocated to VMs through shares
 - Share specifies the relative priority or importance of a VM
- VMs with more shares get more memory
- Determines allocation of additional memory
- Pure proportional-share can be inefficient. Why?
 - Doesn't consider active memory usage (or working set)!

Idle Memory Tax

- When memory is scarce, VMs with idle pages will be penalized compared to more active ones
- Charges tax for idle memory

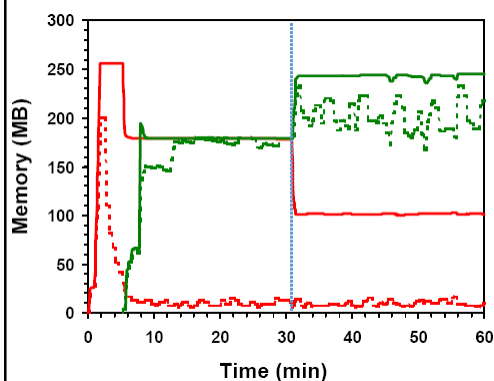
$$\rho = \frac{S}{P \cdot (f + k \cdot (1 - f))}$$

- where S and P are number of shares and allocated pages owned by a client, respectively, f is the fraction that is active
- Idle page cost: $k = 1/(1 - \text{tax_rate})$ with $\text{tax_rate}: 0 < \text{tax_rate} < 1$
- High shares-per-page ratio gives a high priority for memory allocation
 - Smaller allocation and smaller idle memory -> higher ratio

Idle Memory Tax

- Statically samples pages in each VM to estimate active memory usage (f)
- Has a default tax rate of .75
- By default samples 100 pages every 30 seconds

Idle Memory Tax



Experiment:

2 VMs, 256 MB, same shares.

VM1: Windows boot+idle. VM2: Linux boot+dbench.

Solid: usage, Dotted: active.

Change tax rate 0% → 75%

After high tax.

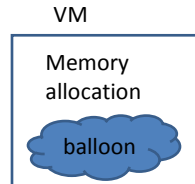
Redistribute VM1 → VM2.

VM1 reduced to min size.

VM2 given more memory

Memory Overbooking in Xen Using Control Theory

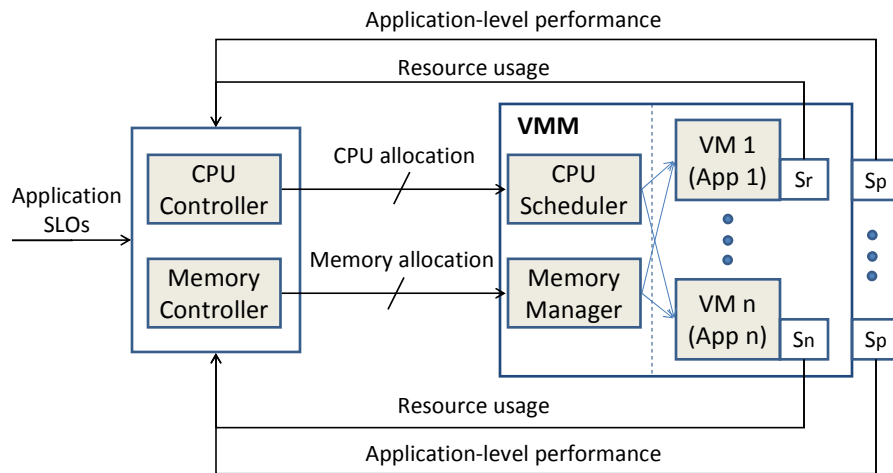
- VMware
 - Page sharing
 - Content-based hashing to identify identical pages
 - Reduces overall memory consumption
 - Ballooning
 - Mechanisms to reclaim unused memory
 - Supports dynamic memory allocation based on the importance of VMs and their actual memory usage → doesn't consider application performance
- Xen
 - Ballooning is supported
 - No dynamic allocation policy
 - "Self-ballooning" patch → doesn't consider application performance



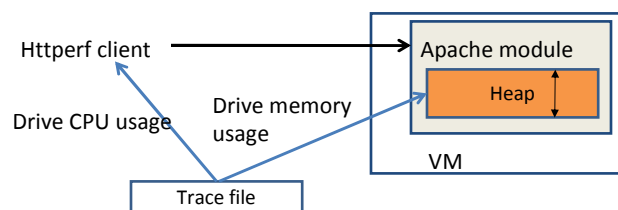
Memory Overbooking in Xen Using Control Theory

- **Enable memory overbooking by dynamically controlling memory allocation in Xen**
 - Consider application performance
 - Use a control-theory based approach

Resource Control System Architecture

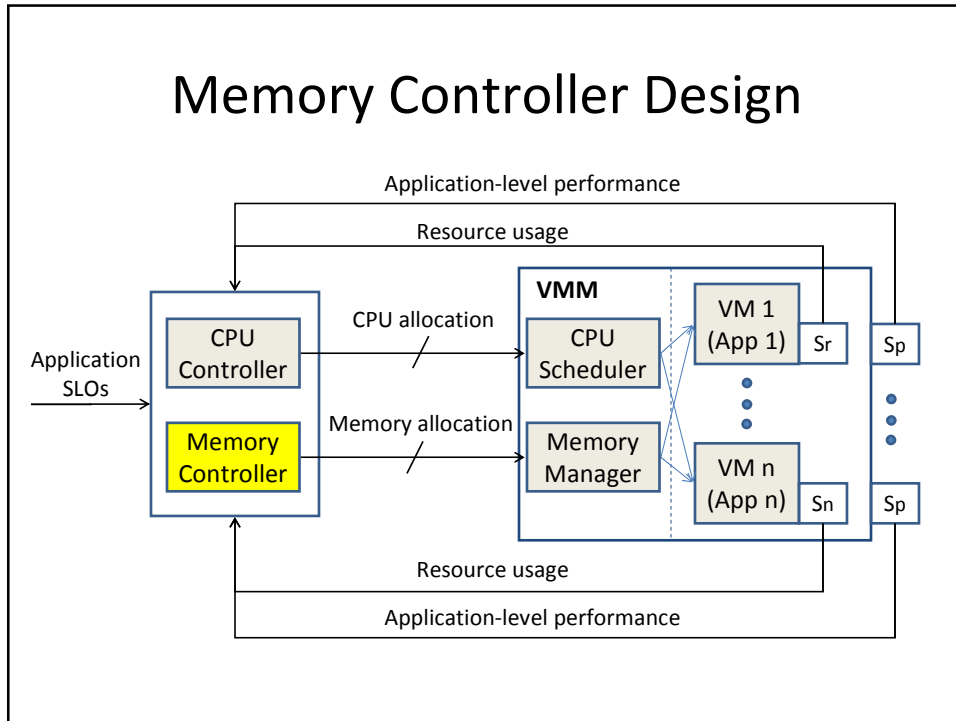


MemAccess



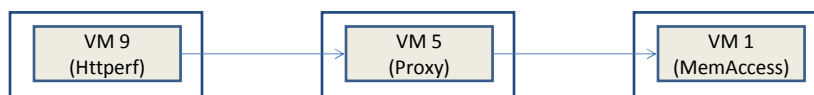
- Drive CPU and memory consumption from the traces
- Implemented as Apache module
- Every request, the apache module executes
 - CPU-intensive computation
 - Randomly access the allocated Heap memory
- Replicate usage pattern from traces files
 - CPU
 - Memory

Memory Controller Design

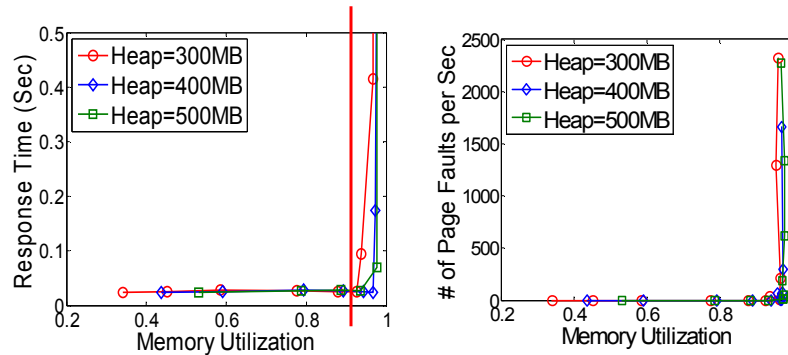


Memory Controller Design Cont.

- Design of memory controller
 - The relationship between memory allocation to a VM and application performance
 - How memory and CPU behave differently
- Experimental setup
 - One httperf client and one VM running a MemAccess application
 - Request is fixed: 30req/sec
 - VM's memory allocation varies from 230MB to 1GB with three different heap sizes of 300MB, 400MB and 500MB

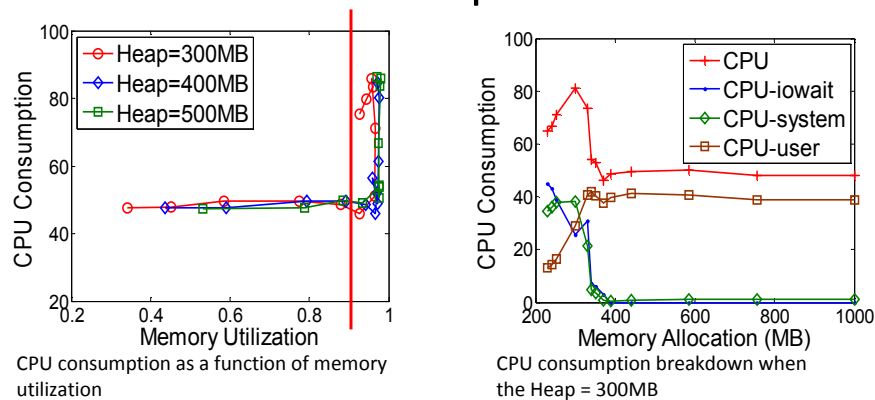


Response Time and Memory Utilization



Sharp increase in response time after a certain threshold, say 90%
 High correlation between the response time and the number of page fault
 Increase in response time caused by the increase memory pressure in the VM

Memory Utilization and CPU Consumption

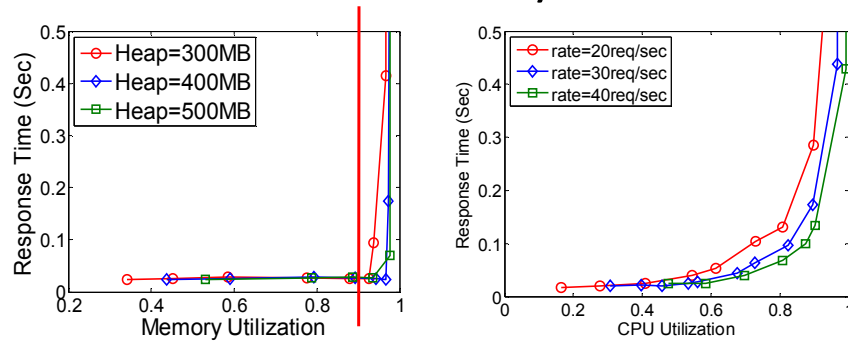


CPU consumption as a function of memory utilization

CPU consumption breakdown when the Heap = 300MB

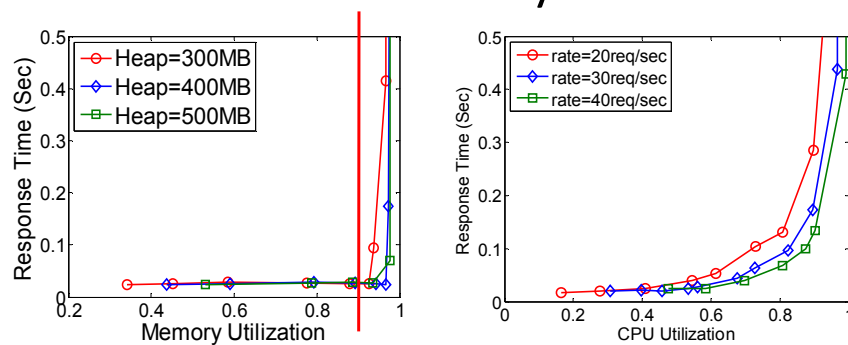
CPU not much affected by memory utilization, significant CPU overhead after the threshold
 Increase in CPU usage mainly caused by extra paging activities - demonstrated by increase in the CPU system time

How Memory and CPU Behave Differently



In both figures, the response time increase monotonically
The curve is much smoother and more differentiable for CPU
Almost binary function for memory

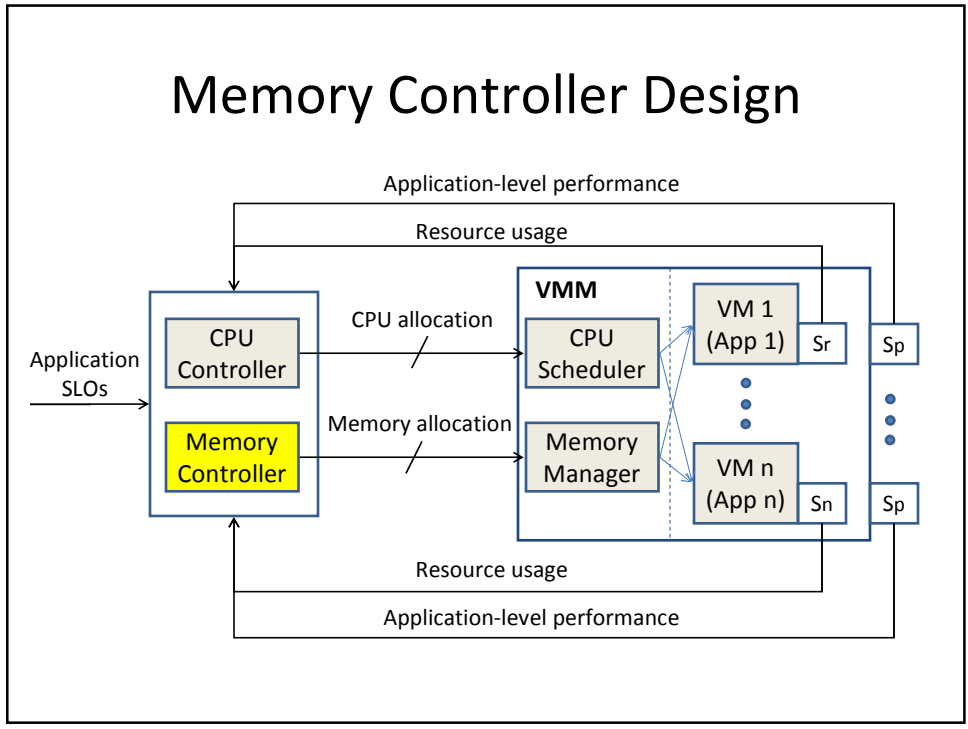
How Memory and CPU Behave Differently



In both figures, the response time increase monotonically
The curve is much smoother and more differentiable for CPU
Almost binary function for memory

Memory allocation doesn't affect response time up to a critical point!

Memory Controller Design



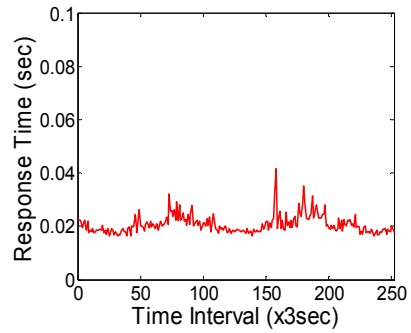
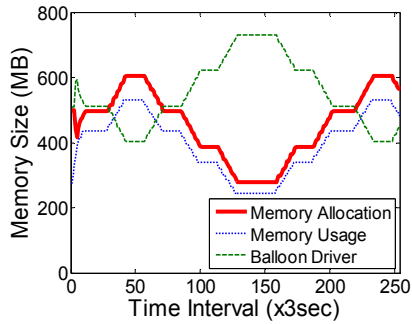
Memory Controller Design

- Maintain memory utilization below a critical point (90%)
- Employ a utilization controller for memory

$$u_{mem}(k+1) = u_{mem}(k) - \lambda_{mem} v_{mem}(k) (r_{mem}^{ref} - r_{mem}(k)) / r_{mem}^{ref}$$

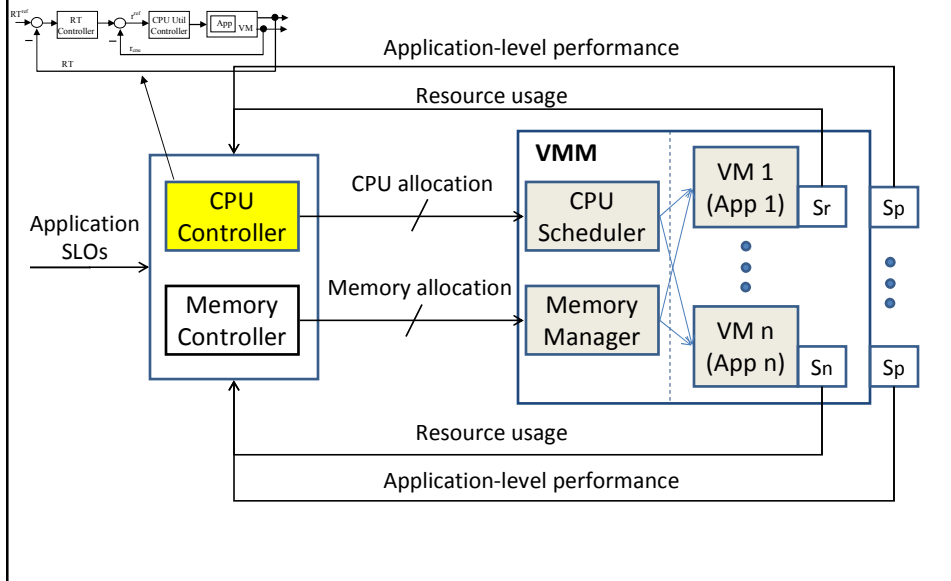
- Proven to be stable for $\lambda < 2.0$

Performance of Memory Controller



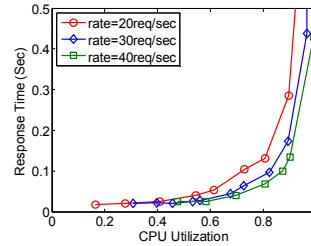
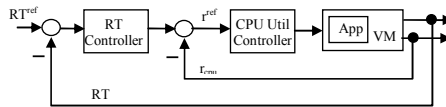
Memory controller successfully allocates memory according to varying demand
The response time is kept small.

(Joint) Resource Control System Architecture



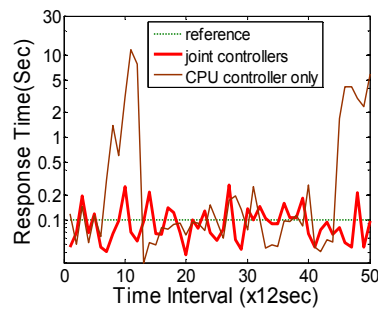
Joint CPU and Memory Controller Design

- CPU controller
 - Inner CPU utilization controller + outer response time controller

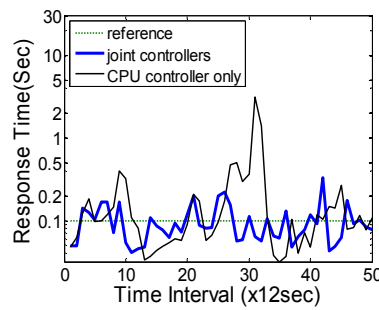


- Memory controller
 - Utilization controller
 - Set the reference utilization to 0.9

Performance of Joint Controllers with Synthetic Workload



VM1



VM2

Joint controller successfully keeps the response time around the target
 With CPU controller only, the response time violates the target severely

Summary of Memory Overbooking in Xen Using Control Theory

- Presented a case study for dynamic memory allocation on Xen-based platforms.
- Designed a joint resource controller for CPU and memory
- Showed that with the joint controller, applications achieved application SLOs, while achieving memory overbooking.