Difference Engine:

Harnessing Memory Redundancy in Virtual Machines

D. Gupta, S. Lee, M. Vrable, S. Savage, A. Snoeren, G. Vargese, G. Voelker, A. Vhadat

Presented by: Benjamin Prosnitz

Motivation

- A typical server has only 5-10% resource utilization
- Servers have high memory requirements:
  - Operating system
  - Applications
  - Caching Data
- Memory is the bottleneck for high consolidation

Reducing Memory Usage: Strategy

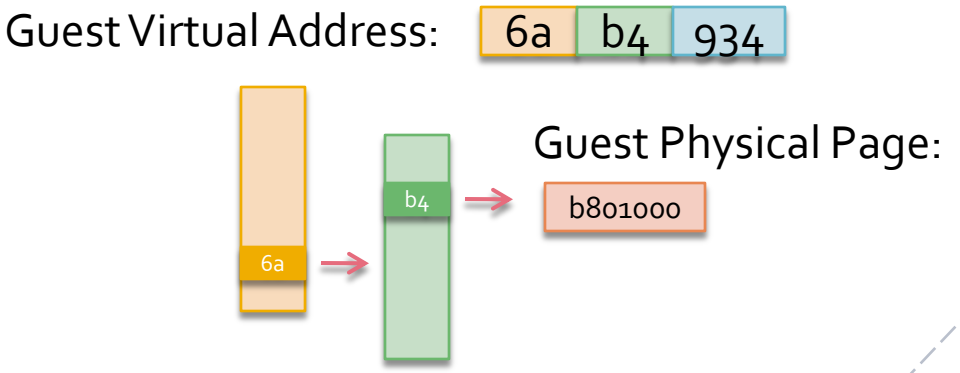
- Identify identical "Sharable" pages, store only one copy
- Identify similar "Patchable" pages, store a copy and patches for that copy
- Compress other infrequently used pages

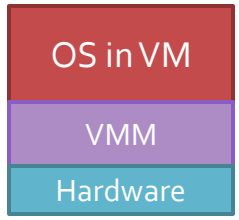
Strong Potential (from VM Snapshot)

Pages	Initial	After Sharing	After Patching
Unique	191,646	191,646	
Sharable (non-zero)	52,436	3,577	
Zero	149,038	1	
Total	393,120	195,224	88,422
Reference		50,727	50,727
Patchable		144,497	37,695

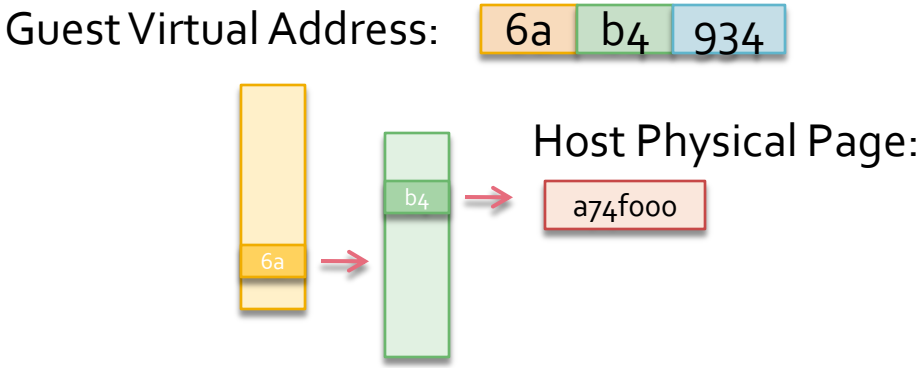
Memory Structures for VMs

- Guest "The Illusion" Page Table





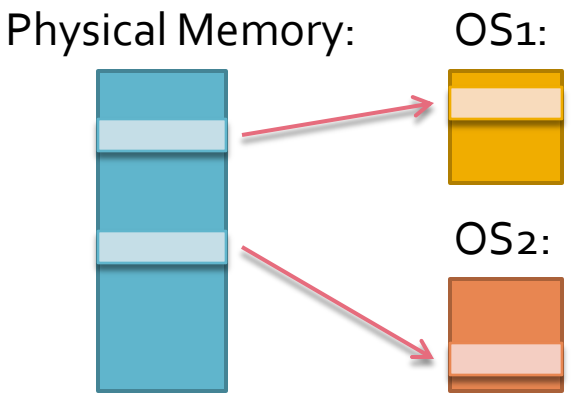
- Shadow "The Real" Page Table



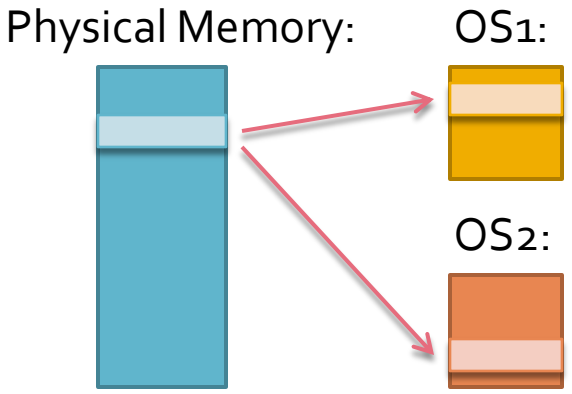
Identical page flags needn't be set in both tables!

Copy-on-Write and VMMs

- Consider Identical Pages:



- Store only one copy



- Mark as read-only in *Shadow* Page Table
  (Guest Page Tables are Unchanged!)

Example: Writing to a Shared Page

1. Application on Guest executes an instruction to write to a shared page

2. Because the Shadow page table has page 89d92000 marked as *read-only*, a page fault occurs which the VMM must handle

Simplified Shadow Page Table Lookup

Example: Writing to a Shared Page

3. The VMM receives the page fault and:[1]
   a. Allocates a new page frame — Frame: 9453a000
   b. Copies data from the old page frame

   a. Updates the shadow page table so the new copy is used by the guest application

4. The Guest finishes writing, oblivious to what the VMM did

1. A step was skipped: If the VMM discovers the page is marked R/O in the Guest OS, it lets the guest OS handle the page fault

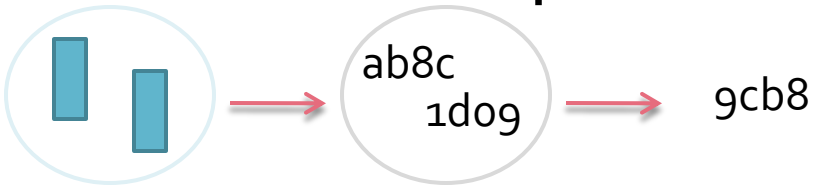
Page Sharing (for Identical Pages)

1. Hash all interesting pages
2. Identify pages with matching hashes
3. Confirm that they are identical using byte-by-byte comparison
4. Use copy-on-write to reduce memory consumption

Patches (for Similar Pages)

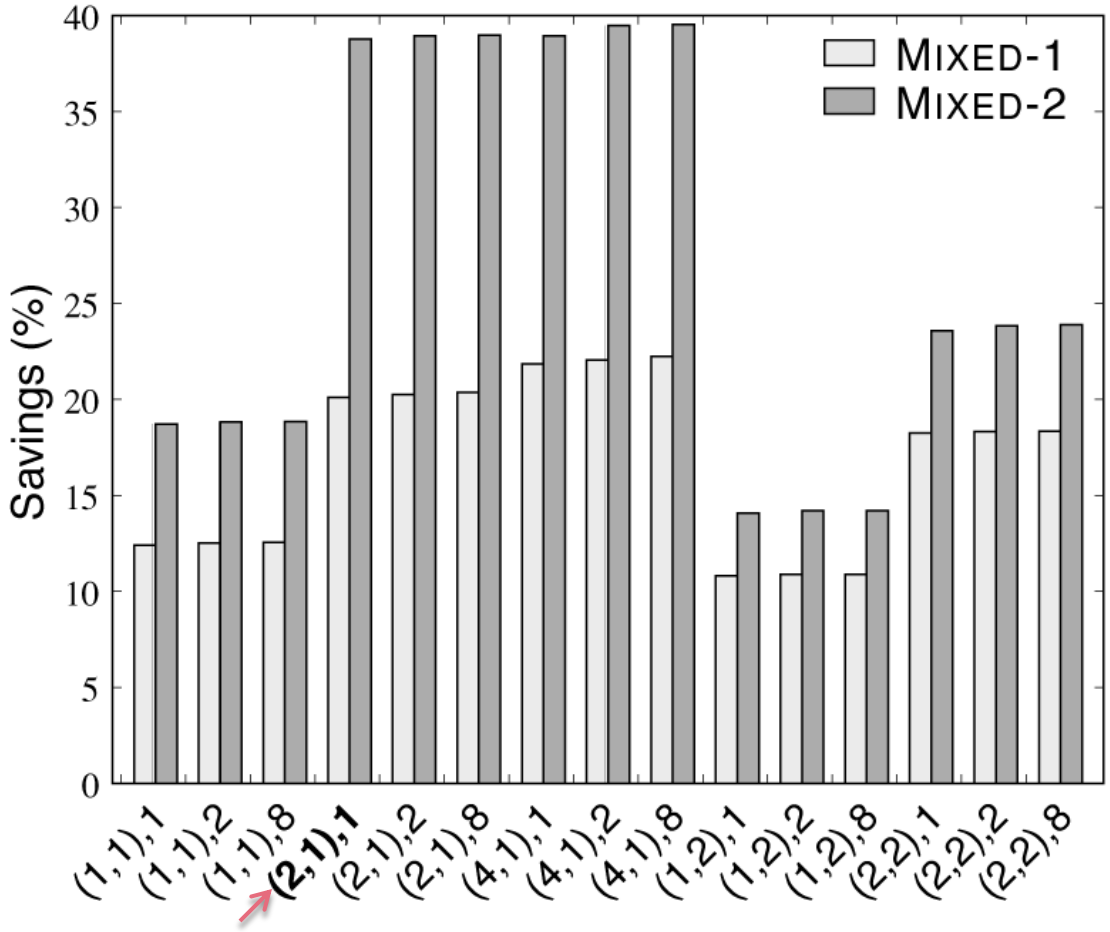
1. Randomly choose *k* fixed comparison points in a page

2. Hash a 64-byte block in each of the *k* locations
3. Compute a secondary hash by combining the hash codes for each possible *s*-block group



4. Create patches for *c* candidates and store the best candidate as copy-on-write

Savings with Different Patching Schemes



(k,s),c=(# hashes, # hashes per group), # candidates for patch

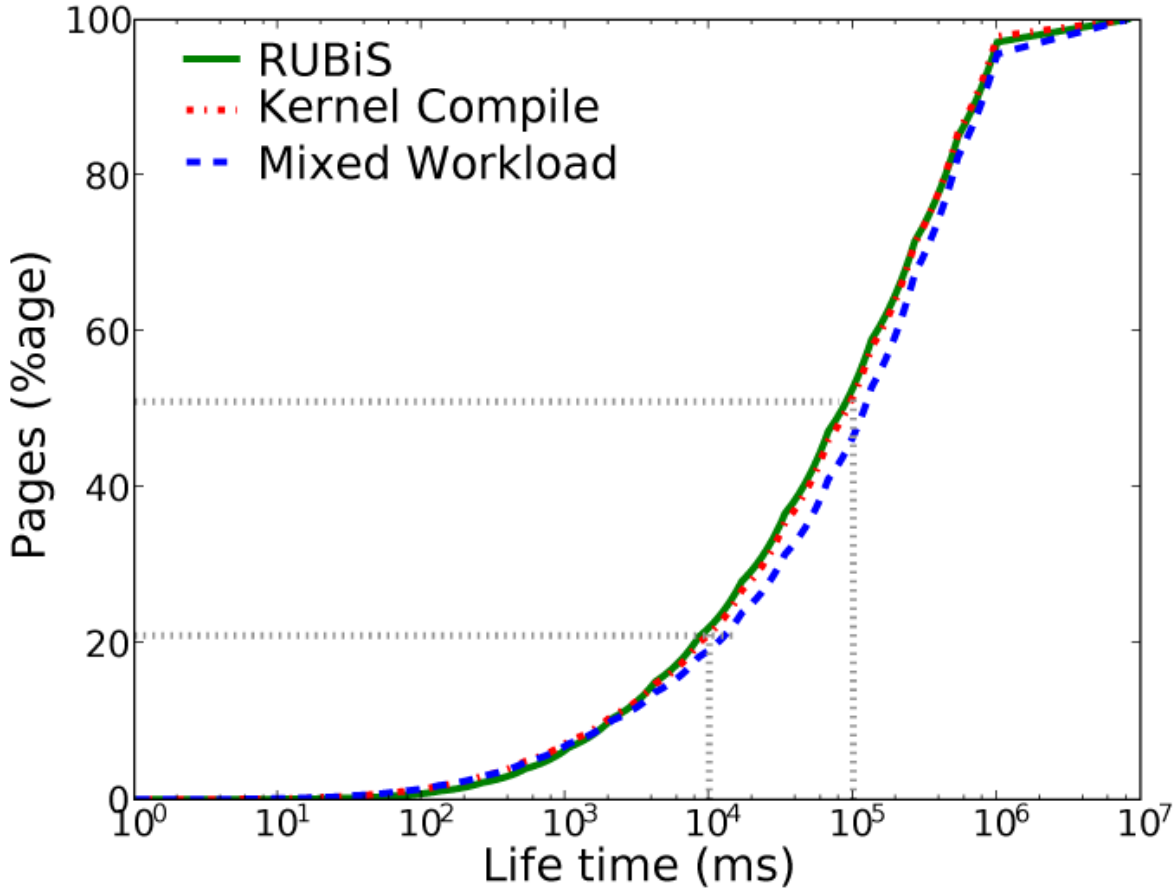
Compression

- Compression is applied to pages that:
  - Are infrequently used
  - Have high compression ratios
  - Have low similarity to other pages

Identifying Infrequently Used Pages

- Uses a Not-Recently-Used (NRU) policy
- Periodically scans modified and referenced flags to identify pages as:
  - Recently Modified (C1) – Stored as normal
  - Not Recently Modified (C2) – Used for sharing and as reference pages for patching
  - Not Recently Accessed (C3) – Used for sharing and patching
  - Not Accessed for an Extended Period (C4) – Used for sharing, patching and compression

Evaluation: NRU Policy



Lifetime of Patched and Compressed Pages for Three Different Workloads

Other Considerations

- Need memory management functionality to store patches and compressed pages
- Need to support paging to disk since there may be lower-than-expected memory redundancy

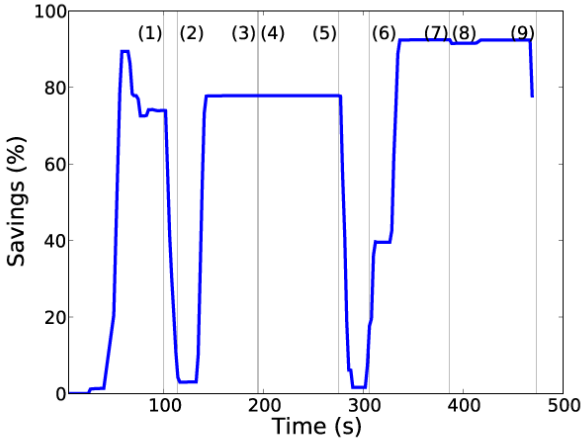
Evaluation: Micro-Benchmarks

Function	Mean execution time (μs)
share_pages	6.2
cow_break	25.1
compress_page	29.7
uncompress	10.4
patch_page	338.1
unpatch	18.6
swap_out_page	48.9
swap_in_page	7151.6

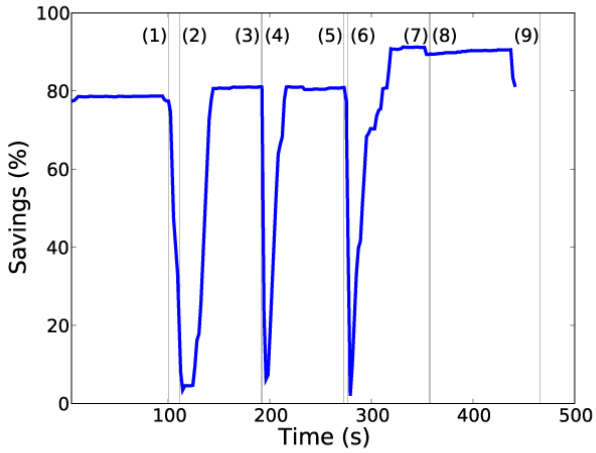
: CPU overhead of different functions.

Evaluation: Artificial Scenarios

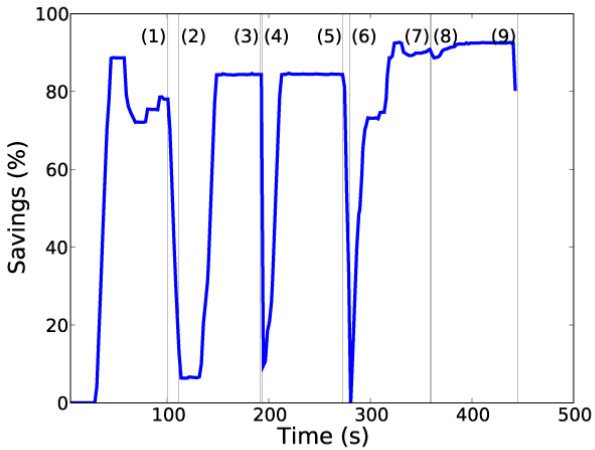
Identical Pages



Sharing



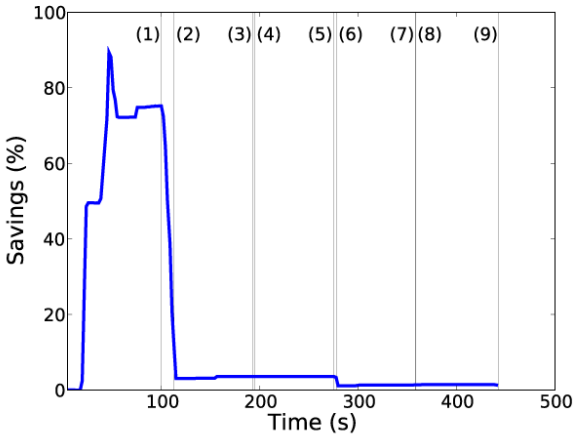
Patching



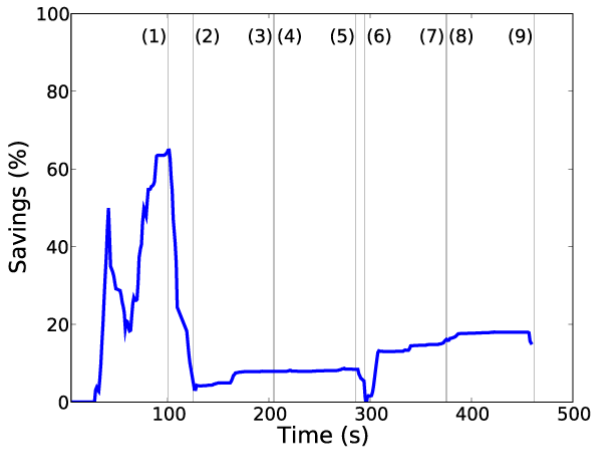
Compression

Evaluation: Artificial Scenarios

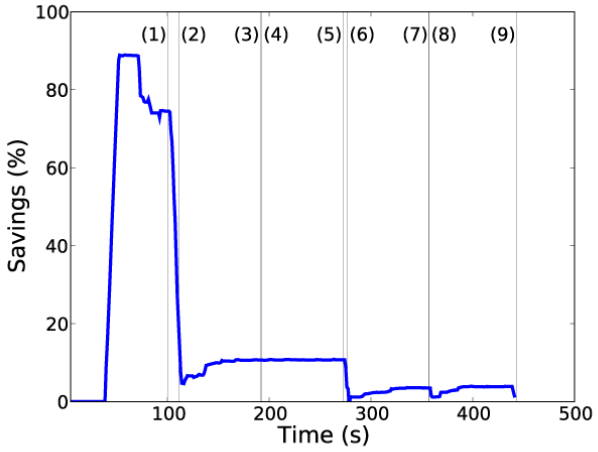
Random Pages



Sharing



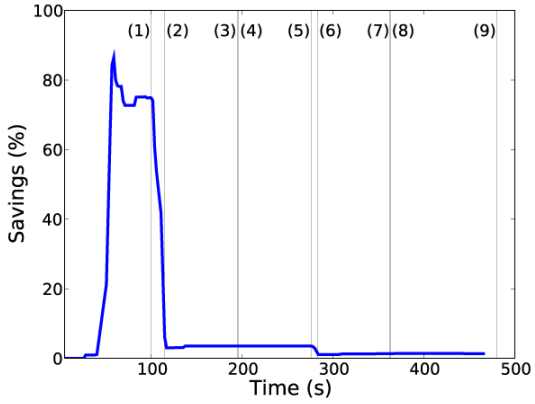
Patching



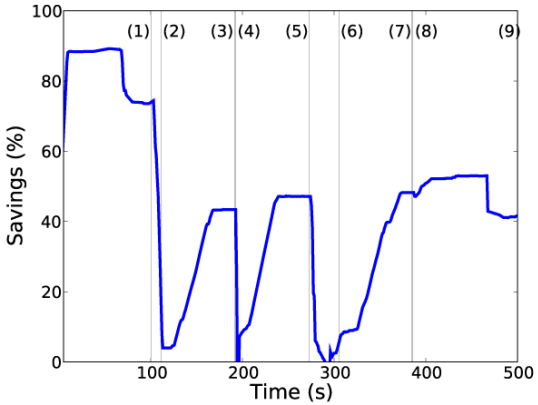
Compression

Evaluation: Artificial Scenarios

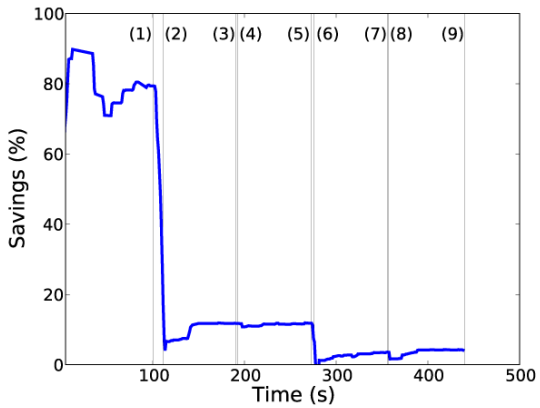
Similar Pages (95% similar)



Sharing

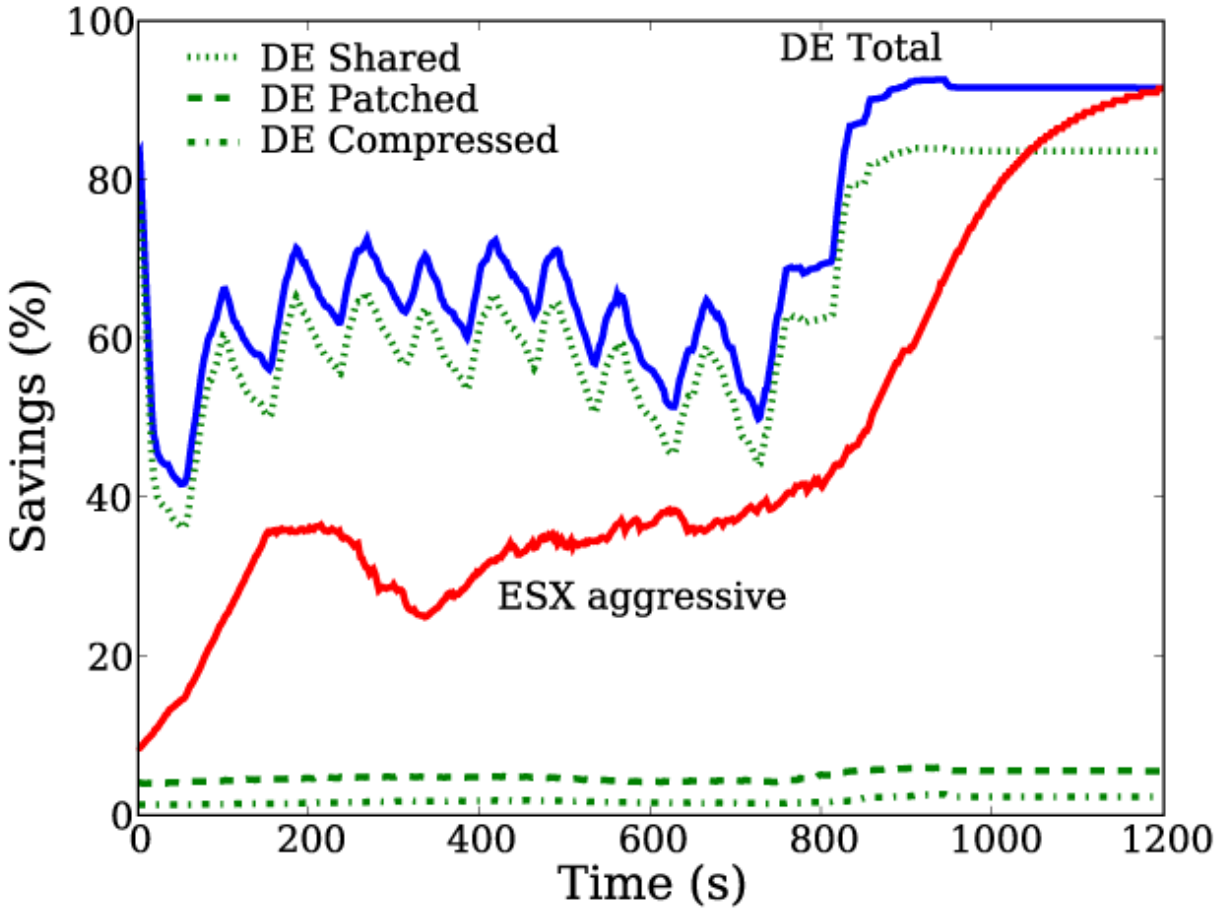


Patching



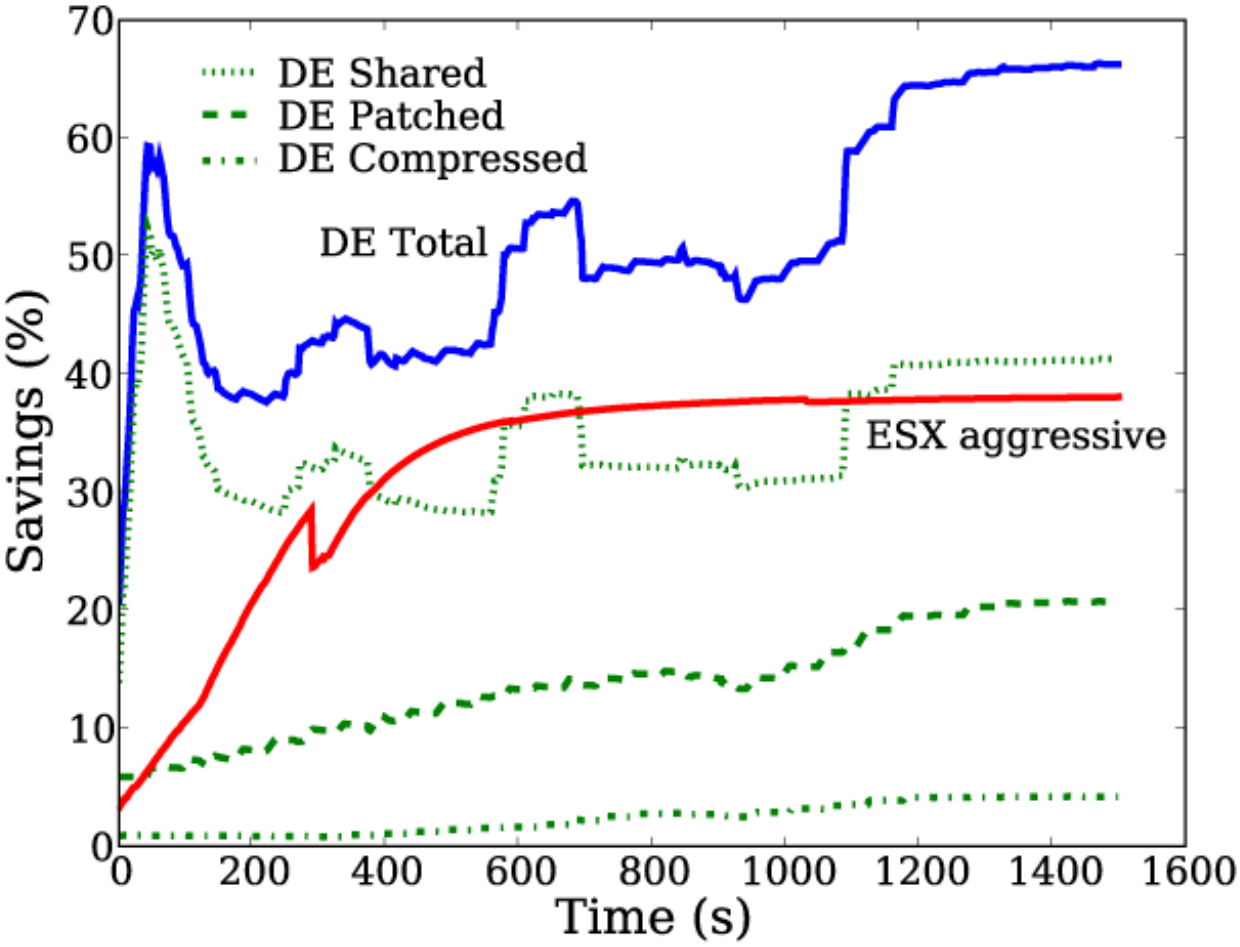
Compression

Evaluation vs. ESX: Homogenous Workload



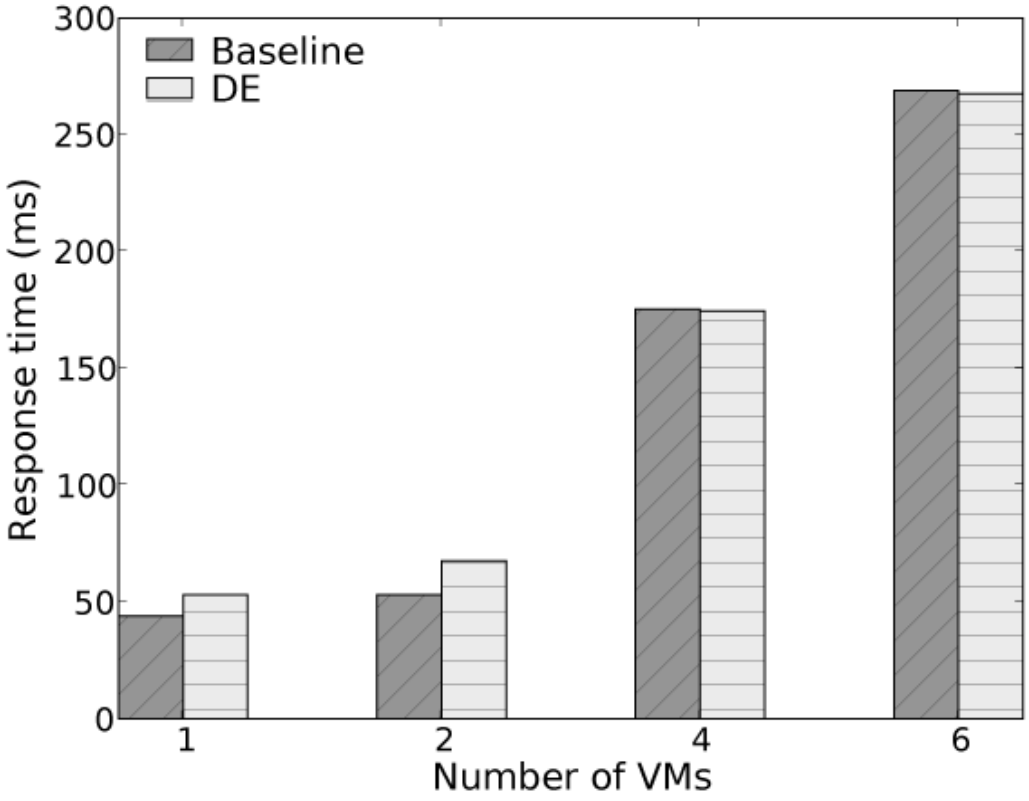
Four identical VMs Execute dbench

Evaluation vs. ESX: Heterogeneous Workload



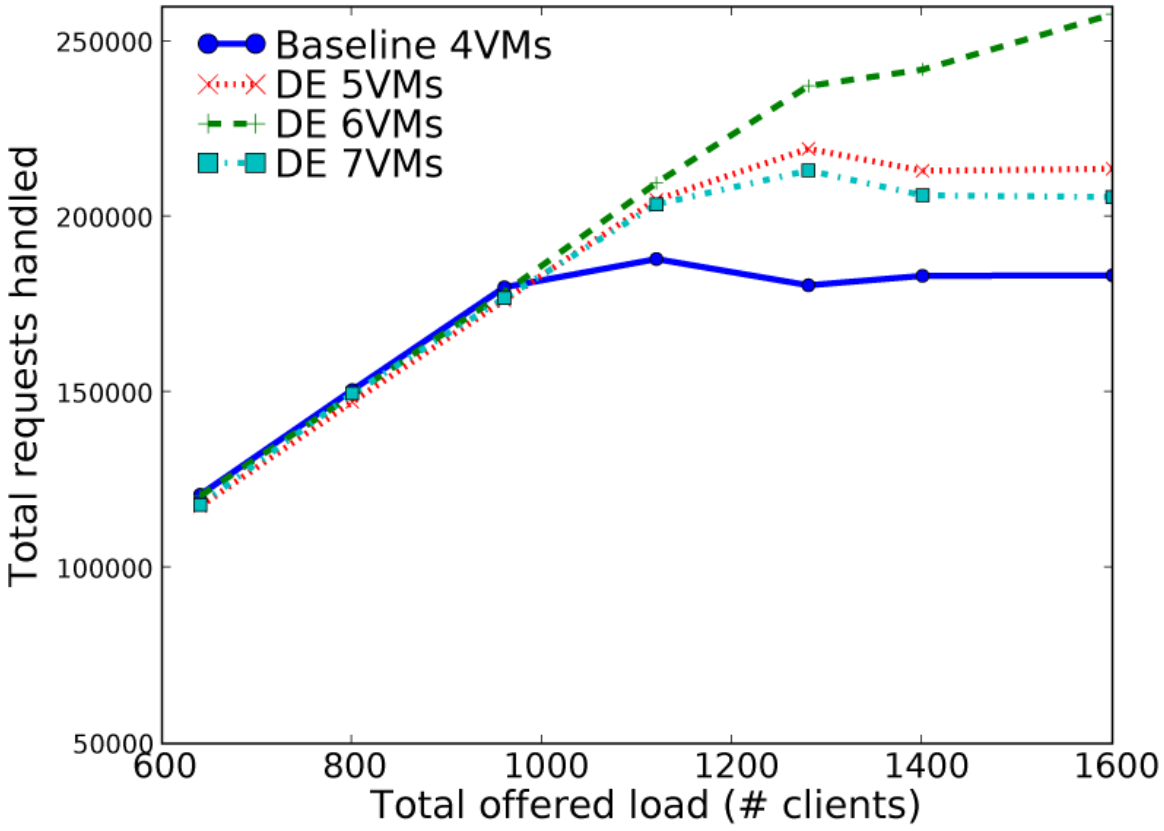
Memory Savings with the Mixed-1 Configuration

Evaluation: Performance



Average response time

Evaluation: Performance



Total requests handled

Issues (part 1)

- No evaluation of *variance* in performance or response time
  - Can one expect a certain response time from servers using the DE?
  - Is it slow when doing its periodic "clock" iterations?
  - Is it slower for certain tasks, like creating processes?

Issues (part 2)

- A *shift* in data would not allow for either sharing or patching (this could be due to an OS kernel security update adding a few instructions, etc.)
- What is the source of memory redundancy in heterogeneous configurations?

Thank you!