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)

<table>
<thead>
<tr>
<th>Pages</th>
<th>Initial</th>
<th>After Sharing</th>
<th>After Patching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique</td>
<td>191,646</td>
<td>191,646</td>
<td></td>
</tr>
<tr>
<td>Sharable (non-zero)</td>
<td>52,436</td>
<td>3,577</td>
<td></td>
</tr>
<tr>
<td>Zero</td>
<td>149,038</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>393,120</td>
<td>195,224</td>
<td>88,422</td>
</tr>
<tr>
<td>Reference</td>
<td></td>
<td>50,727</td>
<td>50,727</td>
</tr>
<tr>
<td>Patchable</td>
<td></td>
<td>144,497</td>
<td>37,695</td>
</tr>
</tbody>
</table>
Memory Structures for VMs

- **Guest “The Illusion” Page Table**
  - Guest Virtual Address: 6a b4 934
  - Guest Physical Page: b801000

- **Shadow “The Real” Page Table**
  - Guest Virtual Address: 6a b4 934
  - Host Physical Page: a74f000

Identical page flags needn’t be set in both tables!
Consider Identical Pages:

- Store only one copy

- Mark as read-only in *Shadow* Page Table
  (Guest Page Tables are Unchanged!)
1. Application on **Guest** executes an instruction to write to a shared page

![Guest](image)

![Write 5 to 89d9231f!](image)

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

![89d92000](image)

![Frame: 34234000 R/O=Y](image)

**Simplified Shadow Page Table Lookup**
Example: Writing to a Shared Page

3. The VMM receives the page fault and:
   a. Allocates a new page frame  Frame: 9453a000
   b. Copies data from the old page frame
      Frame: 34234000  Copy  Frame: 9453a000
   a. Updates the shadow page table so the new copy is used by the guest application
      Frame: 89d92000  Copy  Frame: 9453a000 R/O=N

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
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Mean execution time (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>share_pages</td>
<td>6.2</td>
</tr>
<tr>
<td>cow_break</td>
<td>25.1</td>
</tr>
<tr>
<td>compress_page</td>
<td>29.7</td>
</tr>
<tr>
<td>uncompressed</td>
<td>10.4</td>
</tr>
<tr>
<td>patch_page</td>
<td>338.1</td>
</tr>
<tr>
<td>unpatch</td>
<td>18.6</td>
</tr>
<tr>
<td>swap_out_page</td>
<td>48.9</td>
</tr>
<tr>
<td>swap_in_page</td>
<td>7151.6</td>
</tr>
</tbody>
</table>

CPU overhead of different functions.
Evaluation: Artificial Scenarios

Identical Pages
Evaluation: Artificial Scenarios

Random Pages

- Sharing
- Patching
- Compression
Evaluation: Artificial Scenarios

Similar Pages (95% similar)
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

The graph shows the total requests handled against the total offered load (number of clients) for different configurations:

- **Baseline 4VMs**
- **DE 5VMs**
- **DE 6VMs**
- **DE 7VMs**
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!