Difference Engine: Harnessing Memory Redundancy in Virtual Machines (D. Gupta et al)

Presented by: Konrad Gołuchowski
What is Virtual machine monitor (VMM)?

- Guest OS
- Guest OS
- Guest OS

Virtual machine monitor

Host OS

Hardware
A few samples of VMM

- Xen
- VMware
- VirtualBox
Why use virtualization in business?

- Low CPU utilization by individual services (e.g. 5-10%)
- Need isolation between services
  - (services may require different configuration)
- Deploy a new server within minutes

And your competitors are already using it!

Source: http://brandingbrand.com/
Who uses VMM?

- Anyone?
- Any problems with it?
- Maybe performance problems?

Source: www.jtgraphic.net
Motivation for this paper

- Server configuration (students):
  - 78 GB RAM
  - 24 CPU cores (Intel Xeon @ 2.66GHz)

- How many virtual machines may work on this server?
  - 100?
  - 200?

- But, even with 100 machines, each machine could use less than 1GB RAM
Motivation for this paper

- Hardware upgrade is complicated operation
- High-capacity memory chips are expensive
- And they consume power

- And... competitors are already doing it!

Source: http://news-libraries.mit.edu/
Competitors?

- **VMware ESX server:**
  - Content-based page sharing
  - Reduction of memory footprint by 10-40% (homogenous systems)
Proposed solution: Difference Engine
Proposed solution: Difference Engine

- Built on the base of Xen

Main ideas:

- Page sharing (as VMware does)
- Sub-page level sharing (patching)
- In-memory compression

Source: http://www.sensationallycreative.com/
Page Sharing

- How to locate identical pages quickly?
  - Use hashing and byte-by-byte comparison

- Then, we just update virtual memory to point at the shared copy

- But what if one virtual machine changes shared page?
  - Mark shared copy as read-only
  - Writing to the page causes page fault trapped by VMM
  - VMM creates a private copy and updates virtual memory
Handling similar pages

- Store a reference page and a patch
- How to detect similar pages?
  - Hash 64-byte blocks at random locations
  - Compare computed hashes
- Generate patch (or patches and choose best one)
- Don’t use too large patches
- When use patching?
In-memory compression

- Basic idea: compress pages that are not similar to anything
- Compress when it is worth

- Use compression and patching only for pages accessed infrequently

- How to locate those infrequently accessed pages?
  - Not-recently-used policy and global scanning every some time
  - Scans checks and clears referenced and modified bits
  - VM have time to reset those bits
  - Only a part of memory is scanned (for each VM)
And a Bonus: Paging Machine Memory

- What if all VMs require all their allocated memory?
- And this allocated memory exceeds physical memory?

- Employ paging mechanism
  - Writing pages out to disk

- Use mechanism of locating infrequently used pages

- It is slow! So this operation must be infrequent!
Step-by-step example

- Step 1: Page sharing
Step-by-step example

- Step 2: Patching
Step-by-step example

- Step 3: Compression
A few implementation notes

- Implementation on the top of Xen 3.0.4
- Roughly 14,500 lines of code
- Additional 20,000 lines from ports of existing algorithms
- And I’m not going to bore you with more details :)
Evaluation

Evaluation setups:

- Homogeneous setup
- Mixed-1
  - Windows XP SP1 hosting RUBiS
  - Debian 3.1 compiling Linux kernel
  - Slackware 10.2 compiling Vim 7.0 followed by run of the lmbench
- Mixed-2
  - Windows XP SP1 running Apache 2.2.8 hosting a lot of static content (httpperf running on a separate machine requesting these pages)
  - Debian 3.1 running SysBench database benchmark
  - Slackware 10.2 running dbench followed by IOZone
Memory savings for homogeneous setup

- DE Shared
- DE Patched
- DE Compressed
- DE Total

4 VMs running dbench

Time (s)

Savings (%)
Memory savings for Mixed-1
Memory savings for Mixed-2

![Graph showing memory savings over time for different configurations: DE Shared, DE Patched, DE Compressed, and ESX aggressive. The graph plots savings (%) against time (s).]
Performance evaluation

- **Mixed-1 setup**

<table>
<thead>
<tr>
<th></th>
<th>Kernel Compile (sec)</th>
<th>Vim compile, Imbench (sec)</th>
<th>RUBiS requests</th>
<th>RUBiS response time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td>670</td>
<td>620</td>
<td>3149</td>
<td>1280</td>
</tr>
<tr>
<td><strong>DE</strong></td>
<td>710</td>
<td>702</td>
<td>3130</td>
<td>1268</td>
</tr>
</tbody>
</table>

- Observed performance is within 7% of the baseline

- VMware ESX Server is within 5% of the baseline
How use reclaimed memory?

- We can spawn more virtual machines
Conclusions

- Harvesting identical pages across virtual machines
  - Works well on homogeneous systems
- Patching and in-memory page compression
  - Some improvement on heterogeneous systems
- Small performance overhead

- Article addresses many technical challenges
  - Algorithms
  - Xen limitations
  - Paging support
  - Clock mechanism for infrequently used pages location