Satori:

Enlightened Page Sharing

Grzegorz Miłoś, Derek Murray, Steven Hand
University of Cambridge

Michael Fetterman
NVIDIA
Outline

• Motivation for page sharing
• Existing systems (a.k.a. state of the art)
• Satori overview
• Implementation
• Performance results
Motivation

- Virtualisation becomes ubiquitous
  “The number of virtualized PCs is expected to grow from less than 5 million in 2007 to 660 million by 2011”
  
  Source: Gartner, 2008

- Provisioning computer systems with memory
  ▸ is expensive (hardware cost)
  ▸ consumes power (running cost)
  ▸ is inflexible (limited # of slots, limited chip size)
Motivation

• Homogeneous VMs common

• Identical OSes use identical data:
  ▶ binaries (kernel + programs)
  ▶ libraries
  ▶ configuration files
  ▶ some data files

• Amount of sharable memory
  ▶ up to 70-80% for synthetic workloads
  ▶ ~21% for Linux kernel compilation
Motivation

- Memory sharing reduces VM footprint
- Memory overhead of subsequent homogenous VMs is smaller
- Extra memory can be used to
  - increase page cache size, and thus reduce paging I/O rate
  - increase # of VMs on the host
Sharing cycle

duplicates

page
Sharing cycle

duplicates

page

share

shared

page
Sharing cycle

duplicates

page

share

shared page

reclaimed duplicates
Sharing cycle

- duplicates
- page
- share
- shared page
- reclaimed duplicates
- credit
Sharing cycle

duplicates

page

share

write

shared page

reclaimed duplicates

credit
Sharing cycle

duplicates

page

share

write

shared page

reclaimed duplicates

debit

credit
Sharing cycle

- **Private page**
- **Shared page**
- **Reclaimed duplicates**

**Debit**

**Write**

**Share**

**Credit**
Sharing cycle

duplicates

page

share

copy

write

private page

debit

shared page

reclaimed duplicates

credit
Sharing cycle

duplicates
page

private page

copy

write

shared page

reclaimed duplicates

debit

credit
Sharing cycle

duplicates

page

reclaimed duplicates

private page

shared page

credit

debit

copy

write

share
Sharing cycle

copy
write
share
duplicates
reclaimed duplicates

debit
credit
Sharing cycle

private page

reclaimed duplicates

duplicates

page

write

shared page

credit

debit

copy
Sharing cycle

- Private page
- Shared page
- Reclaimed duplicates

- Copy
- Write
- Share
- Duplication

Debit → Credit
Satori key objectives
Satori key objectives

1. Detect sharing quickly and cheaply
Satori key objectives

1. Detect sharing quickly and cheaply

Hypervisor scans guest memory and compares fingerprints
Satori key objectives

1. Detect sharing quickly and cheaply

Satori monitors virtual I/O devices

→ no periodic scanning
Satori key objectives

1. Detect sharing quickly and cheaply
   Satori monitors virtual I/O devices ➔ *no periodic scanning*

2. Distribute memory savings fairly
Satori key objectives

1. Detect sharing quickly and cheaply
   Satori monitors virtual I/O devices
   ➔ no periodic scanning

2. Distribute memory savings fairly
   Hypervisor manages common pool of surplus memory
Satori key objectives

1. Detect sharing quickly and cheaply
   Satori monitors virtual I/O devices
   ➔ no periodic scanning

2. Distribute memory savings fairly
   VMs receive sharing entitlements
   in proportion to # pages shared
Satori key objectives

1. Detect sharing quickly and cheaply
   Satori monitors virtual I/O devices
   ➞ *no periodic scanning*

2. Distribute memory savings fairly
   VMs receive *sharing entitlements*
   in proportion to # pages shared

3. Reclaim memory efficiently
Satori key objectives

1. Detect sharing quickly and cheaply
   Satori monitors virtual I/O devices
   ⇒ no periodic scanning

2. Distribute memory savings fairly
   VMs receive sharing entitlements
   in proportion to # pages shared

3. Reclaim memory efficiently
   Hypervisor implements secondary memory paging algorithm
Satori key objectives

1. Detect sharing quickly and cheaply
   Satori monitors virtual I/O devices → *no periodic scanning*

2. Distribute memory savings fairly
   VMs receive *sharing entitlements*
   in proportion to # pages shared

3. Reclaim memory efficiently
   Memory managed *exclusively* by the VMs
   sharing exposed to the VMs
Sharing-aware block devs

- Intuition: most (non-zero) duplicates originate from VM page caches
- Sharing-aware block devices observe I/O reads to build up knowledge of page caches
Sharing entitlements

- Satori tracks the owners of shared pseudo-physical pages
- Entitlement proportional to the # of pages shared & # of pages reclaimed

<table>
<thead>
<tr>
<th>VM memory</th>
<th>VM1</th>
<th>VM2</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>entitlement</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Sharing entitlements

- Satori tracks the owners of shared pseudo-physical pages
- Entitlement proportional to the # of pages shared & # of pages reclaimed
Sharing entitlements

- Satori tracks the owners of shared pseudo-physical pages
- Entitlement proportional to the # of pages shared & # of pages reclaimed

<table>
<thead>
<tr>
<th>VM memory</th>
<th>VM1</th>
<th>VM2</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>entitlement</td>
<td>½</td>
<td>½</td>
</tr>
</tbody>
</table>
Satori tracks the owners of shared pseudo-physical pages.

Entitlement proportional to the # of pages shared & # of pages reclaimed.

VM memory

VM1

VM2

HW memory

entitlement

\[ \frac{2}{3} \]

\[ \frac{4}{3} \]
Memory transfer
Memory transfer
Memory transfer

credit

VM

balloon
Memory transfer

credit

VM

balloon
Memory transfer

credit

VM

balloon
Memory transfer

debit

credit

VM

balloon
Memory transfer

VM

repayment
FIFO

VM

balloon
Memory transfer

debit

credit

VM

VM

repayment

FIFO

balloon
Implementation in Xen

- Changes in the Xen hypervisor (5351 LoC)
  - low-level sharing support
  - sharing entitlement computation
  - fault handling
- Changes in Domain 0 (3894 LoC)
  - sharing-aware block devices
  - management tools
- Changes in Domain U (2306 LoC)
  - repayment FIFO (volatile pgs from IBM CMM)
Performance results

Overheads

• Sharing-aware block devices interpose on data read path

• Worst-case overhead for sequential reads
  
  hashing  0.2%
  hashing + IPC  34.8%

• Negligible for non-sequential reads

• Kernel compilation macro-benchmark:
  without Satori: 780s, with Satori 779s
Performance results
Detection effectiveness

Kernel Compilation (512MB)

Pages

Time (mins)

Potential
Satori
Performance results
Detection effectiveness

Kernel Compilation (512MB)

Pages

Time (mins)

Satori
VMware
Performance results

Performance impact – reads

Read progress in VM1

Read progress in VM2

0
2
4
6
8s

0.22s
Performance results

Performance impact - httpd

Httpd performance

Response rate (reqs/s) vs Time (s)

- Satori
- VMware without Tools
- VMware with Tools
Performance results

One slide summary

• Detection cheap and effective
  ▸ less than 1% overhead (except IPC)
  ▸ duplicates detected immediately
  ▸ more effective than scanning

• No physical I/O if data already present in any virtual machine memory

• Surplus memory improves overall system performance
Conclusions

• Satori implements enlightened page sharing
• Satori is efficient (low overheads)
• Satori is effective (high coverage)
• Satori is fair (proportional entitlements)
• Satori maintains isolation (security and perf)

Thanks!

gm281@cam.ac.uk
http://www.cl.cam.ac.uk/~gm281