Transcendent Memory on Linux

Speaker: Dan Magenheimer
Oracle Corporation

Rocky Mountain CMG
Fall 2009 Forum

originally presented at the 2009 Linux Symposium
Agenda

• Motivation and Challenge
• Overview of Physical Memory Management
• Transcendent Memory (“tmem”) Overview
• Transcendent Memory in Action
• Status, Futures, etc.
Motivation

• **Memory** is increasingly becoming a bottleneck in virtualized system
• Existing mechanisms have major holes
The Virtualized Physical Memory Resource Optimization Challenge

Optimize, across time, the distribution of machine memory among a maximal set of virtual machines by:

- measuring the current and future memory need of each running VM and
- reclaiming memory from those VMs that have an excess of memory and either:
  - providing it to VMs that need more memory or
  - using it to provision additional new VMs.

- **without** suffering a significant performance penalty
The Virtualized Physical Memory Resource Optimization Challenge

Optimize, across time, the distribution of machine memory among a maximal set of virtual machines by:

- measuring the current and future memory need of each running VM and
- reclaiming memory from those VMs that have an excess of memory and either:
  - providing it to VMs that need more memory or
  - using it to provision additional new VMs.
- *without* suffering a significant performance penalty

…..*Why is this a hard problem?*
Agenda

- Motivation and Challenge
- **Overview of Physical Memory Management**
  - *in an operating system*
    - in a virtual machine monitor (Xen)
- Transcendent Memory Overview
- Transcendent Memory In Action
- Status, Futures, etc.
OS Physical Memory Management

- Operating systems are memory hogs!

Memory constraint
OS Physical Memory Management

- Operating systems are memory hogs!

If you give an operating system more memory…..

New larger memory constraint
OS Physical Memory Management

- Operating systems are memory hogs!

My name is Linux and I am a memory hog

…it uses up any memory you give it!

Memory constraint
OS Physical Memory Management

• What does an OS do with all that memory?
  • Kernel code and data
  • User code and data
  • Page cache!
OS Physical Memory Management

What does an OS do with all that memory?

Page cache attempts to predict future needs of pages from the disk...

sometimes it gets it right → “good” pages
OS Physical Memory Management

- What does an OS do with all that memory?

Page cache attempts to predict future needs of pages from the disk...

sometimes it gets it wrong → “wasted” pages
OS Physical Memory Management

- What does an OS do with all that memory?
  ...much of the time mostly page cache
  ...some of which will be useful in the future
  ...and some of which is wasted
Agenda

• Motivation and Challenge
• Overview of Physical Memory Management
  • in an operating system
  • *in a virtual machine monitor (Xen)*
• Transcendent Memory Overview
• Transcendent Memory In Action
• Status, Futures, etc.
VMM Physical Memory Management

• Xen partitions memory
  • hypervisor memory
  • dom0 memory
  • guest memory
  • “leftover” (in case needed later)

Dom0 is special 😊

VMM Physical Memory Management

• Xen partitions memory
  • hypervisor memory
  • dom0 memory
  • guest memory
  • “leftover” (in case needed later)

Dom0 is special 😊
VMM Physical Memory Management

- Xen partitions memory
  - Xen memory
  - dom0 memory
  - guest 1 memory
  - guest 2 memory
  - whatever’s left over: “fallow” memory

fallow, adj., land left without a crop for one or more years
VMM Physical Memory Management

- Xen partitions memory
  - Xen memory
  - dom0 memory
  - guest 1 memory
  - guest 2 memory
  - whatever’s left over: “fallow” memory

*fallow*, adj., *land left without a crop for one or more years*
VMM Physical Memory Management

- Xen partitions memory among more guests
  - Xen memory
  - dom0 memory
  - guest 1 memory
  - guest 2 memory
  - guest 3…

- BUT still fallow memory leftover
VMM Physical Memory Management in the presence of migration

- migration
  - requires fallow memory in the target machine
  - leaves behind fallow memory in the originating machine
VMM Physical Memory Management in the presence of ballooning

- Use ballooning to allow guest memory size to grow?
  - Goal: fill fallow memory
VMM Physical Memory Management in the presence of ballooning

- Look! No more fallow memory!

But…..
VMM Physical Memory Management in the presence of ballooning

• Look! No more fallow memory!

*But*….
VMM Physical Memory Management

in the presence of ballooning

• Look! No more fallow memory!

But….

And but…
VMM Physical Memory Management in the presence of ballooning

- Ballooning down to handle incoming migrations
  - *NASTY issues*

To make room for a new guest, we have to starve existing guests!
VMM Physical Memory Management in the presence of ballooning

Using ballooning to take memory away:

• not instantaneous (memory inertia)
• guest can’t predict future needs
  • good pages are evicted along with the bad
• don’t know how much/fast to balloon
  • Too much or too fast
    → thrashing or the dreaded OOM killer

OOM
OOM
OOM

Transcendent Memory on Linux (RMCMG Fall Forum 2009) - Dan Magenheimer
The Virtualized Physical Memory Resource Optimization Challenge

Optimize, across time, the distribution of machine memory among a maximal set of virtual machines by:

• measuring the current and future memory need of each running VM and

• reclaiming memory from those VMs that have an excess of memory and either:
  • providing it to VMs that need more memory or
  • using it to provision additional new VMs.

• without suffering a significant performance penalty

…..This IS a hard problem!!!
Why this IS a hard problem!

Summary

• OS’s use as much memory as they are given
  • but cannot predict the future so often guess wrong
  • and often much memory owned by an OS is wasted

• Xen leaves large amounts of memory fallow
  • fixed partitioning results in fragmentation
  • migration requires fallow memory to succeed

• Ballooning helps but:
  • can’t predict future memory needs of guests
  • memory has inertia
  • the price of incorrect guesses can be dire

→ NEED A NEW APPROACH TO VIRTUALIZED PHYSICAL MEMORY MANAGEMENT!!
Agenda

• Motivation and Challenge
• Overview of Physical Memory Management
• Transcendent Memory Overview
• Transcendent Memory In Action
• Status, Futures, etc.
Transcendent memory
creating the transcendent memory pool

- Step 1a: reclaim fallow memory
- Step 1b: reclaim wasted guest memory (e.g. via ballooning)
- Step 1c: collect it all into a pool

Transcendent memory pool
Transcendent memory
creating the transcendent memory pool

• Step 2: provide *indirect* access, strictly controlled by the hypervisor and dom0
Transcendent memory

API characteristics

- OS changes (small)
- narrow
- well-specified
- operations are:
  - synchronous
  - page-oriented (one page per op)
  - copy-based
  - multi-faceted
  - extensible
Transcendent memory
Linux “tmem_ops” API

int (*new_pool)(struct tmem_pool_uuid uuid, u32 flags);

int (*put_page)(u32 pool_id, u64 object, u32 index, unsigned long pfn);

int (*get_page)(u32 pool_id, u64 object, u32 index, unsigned long pfn);

int (*flush_page)(u32 pool_id, u64 object, u32 index);

int (*flush_object)(u32 pool_id, u64 object);

int (*destroy_pool)(u32 pool_id);
Transcendent memory
Xen hypercall API

```c
int xen_tmem_op(
    u32 tmem_cmd, u32 tmem_pool, u64 object, u32 index, unsigned long gmfn, u32 tmem_offset, u32 pfn_offset, u32 len);

int xen_tmem_new_pool(
    struct tmem_pool_uuid uuid, u32 flags);
```
Transcendent memory
four different subpool types
➔ four different uses

<table>
<thead>
<tr>
<th>flags</th>
<th>ephemeral</th>
<th>persistent</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>“second-chance” clean-page cache!!</td>
<td>Fast swap “device”!!</td>
</tr>
<tr>
<td></td>
<td>➔ “precache”</td>
<td>➔ “preswap”</td>
</tr>
<tr>
<td>shared</td>
<td>server-side cluster filesystem cache</td>
<td>inter-guest shared memory?</td>
</tr>
<tr>
<td></td>
<td>➔ “shared precache”</td>
<td></td>
</tr>
</tbody>
</table>

Legend:
- **Implemented and working today (Linux + Xen)**
- **In development**
- **Under investigation**

**eph-em-er-al, adj., ... transitory, existing only briefly, short-lived (i.e. NOT persistent)**
Transcendent memory

caveats

• Requirements
  • guest OS kernel must be patched
  • 64-bit Xen hypervisor and CPU (32-bit guests OK)

• Workload:
  • should exert memory pressure in at least one guest
  • memory pressure in multiple guests should vary across time

• For best results:
  • dom0 should be configured with a fixed memory size
  • guest should have a (virtual) swap disk configured

• Complementary to:
  • feedback-directed ballooning
  • transparent content-based page sharing
Transcendent memory
Linux core tmem 2.6.30 patch diffstat

<table>
<thead>
<tr>
<th>File</th>
<th>Diffstat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation/transcendent-memory.txt</td>
<td>175 ++++++</td>
</tr>
<tr>
<td>include/linux/tmem.h</td>
<td>88 +++</td>
</tr>
<tr>
<td>mm/Kconfig</td>
<td>10</td>
</tr>
<tr>
<td>mm/Makefile</td>
<td>1</td>
</tr>
<tr>
<td>mm/tmem.c</td>
<td>62 ++</td>
</tr>
</tbody>
</table>

5 files changed, 336 insertions(+)

[PATCH 1/4] Declares the tmem_ops accessors and initializes them as no-ops (foundation layer for “precache” and “preswap”)

(see [http://lkml.org/lkml/2009/7/7/](http://lkml.org/lkml/2009/7/7/) for patchset)
Agenda

• Motivation and Challenge
• Overview of Physical Memory Management
• Transcendent Memory Overview
• Transcendent Memory In Action
  • private-ephemeral pool → “precache”
  • shared-ephemeral pool → “shared precache”
  • private-persistent pool → “preswap”
• performance analysis
• Status, Future, etc.
precache

- a *second-chance* clean page cache for a guest
  - “put” clean pages only
  - “get” only valuable pages
  - pages eventually are evicted
  - coherency managed by guest
  - exclusive cache semantics

Transcendent Memory Pool types

<table>
<thead>
<tr>
<th></th>
<th>ephemeral</th>
<th>persistent</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>private</strong></td>
<td>second-chance clean-page cache!!</td>
<td>Fast swap “device”!!</td>
</tr>
<tr>
<td></td>
<td>➔ “precache”</td>
<td>➔ “preswap”</td>
</tr>
<tr>
<td><strong>shared</strong></td>
<td>server-side cluster filesystem cache?</td>
<td>inter-guest shared memory?</td>
</tr>
<tr>
<td></td>
<td>➔ “shared precache”</td>
<td></td>
</tr>
</tbody>
</table>
precache (with compression)

- Compression
  - Optional (per-guest)
  - nominally doubles available memory
  - performance-space tradeoff
  - some fragmentation issues
precache (multiple guests)

- second-chance page cache for *multiple* guests

- Need “memory scheduler”:
  - global admission/eviction policy:
    - LRU queue
    - weight balanced
shared precache (for clustering)

- guests sharing a clustered filesystem
  - non-exclusive
  - LFU instead of LRU
  - compression optional

→ a server-side disk cache!
- security issues being worked
  - resolved

Transcendent Memory Pool types

<table>
<thead>
<tr>
<th></th>
<th>ephemeral</th>
<th>persistent</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>&quot;second-chance&quot; clean-page cache!</td>
<td>Fast swap &quot;device&quot;!</td>
</tr>
<tr>
<td></td>
<td>&quot;precache&quot;</td>
<td></td>
</tr>
<tr>
<td>shared</td>
<td>server-side cluster filesystem cache?</td>
<td>inter-guest shared memory?</td>
</tr>
<tr>
<td></td>
<td>&quot;shared precache&quot;</td>
<td></td>
</tr>
</tbody>
</table>
Transcendent memory
Linux precache API

extern void precache_init(struct super_block *sb);
extern int precache_get(struct address_space *mapping,
                         unsigned long index, struct page *empty_page);
extern int precache_put(struct address_space *mapping,
                         unsigned long index, struct page *page);
extern int precache_flush(struct address_space *mapping,
                           unsigned long index);
extern int precache_flush_inode(
                               struct address_space *mapping);
extern int precache_flush_filesystem(
                               struct super_block *sb);
Transcendent memory
Linux precache 2.6.30 patch diffstat

fs/buffer.c   |   5
fs/ext3/super.c |   2
fs/mpage.c   |   8 +
fs/super.c   |   5
include/linux/fs.h | 7 +
mm/truncate.c | 10 +
mm/filemap.c | 11 +
mm/Kconfig   | 8 +
mm/Makefile  | 1
include/linux/precache.h | 50 ++++++
mm/precache.c | 134 ++++++++++++++++

11 files changed, 241 insertions (+)

[PATCH 2/4] (see http://lkml.org/lkml/2009/7/7/ for patchset)
Preswap

- over-ballooned guests experiencing unexpected memory pressure have an emergency swap disk
  - much faster than swapping
  - persistent ("dirty") pages OK
  - prioritized higher than precache
  - limited by guest’s maxmem

Transcendent Memory Pool types

<table>
<thead>
<tr>
<th>Type</th>
<th>ephemeral</th>
<th>persistent</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>&quot;second-chance&quot; clean-page cache!!</td>
<td>Fast swap &quot;device&quot;!!</td>
</tr>
<tr>
<td></td>
<td>&quot;precache&quot;</td>
<td>&quot;preswap&quot;</td>
</tr>
<tr>
<td>shared</td>
<td>server-side cluster filesystem cache?</td>
<td>inter-guest shared memory?</td>
</tr>
<tr>
<td></td>
<td>&quot;shared precache&quot;</td>
<td></td>
</tr>
</tbody>
</table>
Transcendent memory
Linux preswap API

extern void preswap_init(unsigned type);
extern int preswap_put(struct page *page);
extern int preswap_get(struct page *page);
extern void preswap_flush(unsigned type,
    unsigned long offset);
extern void preswap_flush_area(unsigned type);
extern void preswap_shrink(unsigned long target_pages);
extern int preswap_test(struct swap_info_struct *sis,
    unsigned long offset);

(Note: “type” is Linux term for which swap disk, an index into
   swap_info_struct array)
Transcendent Memory on Linux (RMCMG Fall Forum 2009) - Dan Magenheimer

Transcendent memory
Linux preswap 2.6.30 patch diffstat

```plaintext
include/linux/swap.h |  57  ++++
m/m/swapfile.c       |  46  +++
include/linux/sysctl.h|    1           |
kernel/sysctl.c       |  12
m/m/Kconfig          |    8
m/m/Makefile          |    1
m/m/page_io.c         |  12
m/m/preswap.c         | 273  ++++++++++++++++++++

8 files changed, 404 insertions(+), 6 deletions(-)

[PATCH 3/4] Somewhat more invasive than precache due to new preswap_map array and “partial swapoff”

(see http://lkml.org/lkml/2009/7/7/ for patchset)
```
Transcendent memory
Linux tmem-xen glue 2.6.30 patch diffstat

arch/x86/include/asm/xen/hypercall.h | 8 +
drivers/xen/Makefile | 1
drivers/xen/tmem.c | 97 +++++++++++++++++
include/xen/interface/tmem.h | 43 ++++
include/xen/interface/xen.h | 22 ++

5 files changed, 171 insertions(+)

[PATCH 4/4] Changes in Xen-specific code to layer Linux tmem API on xen hypercall
(see http://lkml.org/lkml/2009/7/7/ for patchset)
Transcendent memory
Linux 2.6.30 tmem patch diffstat
showing changed existing files only

fs(buffer.c) | 5 +
fs/ext3/super.c | 2
fs/mpage.c | 8 ++
fs/super.c | 5 +
mm/filemap.c | 11 ++++
mm/page_io.c | 12 +++
mm/swapfile.c | 46 ++++++++++++++++++-
mm/truncate.c | 10 ++
kern/sctl.c | 12 +++
include/linux/fs.h | 7 ++
include/linux/swap.h | 57 ++++++++++++++++++
include/linux/sctl.h | 1
mm/Kconfig | 26 +++++
mm/Makefile | 3 +
14 files changed, 199 insertions(+), 6 deletions(-)

(see http://lkml.org/lkml/2009/7/7/ for patchset)
Agenda

- Motivation and Challenge
- Overview of Physical Memory Management
- Transcendent Memory Overview
- Transcendent Memory In Action
  - private-ephemeral pool \(\rightarrow\) “precache”
  - shared-ephemeral pool \(\rightarrow\) “shared precache”
  - private-persistent pool \(\rightarrow\) “preswap”
- performance analysis
- Status, Future, etc.
performance analysis: screenshot
performance analysis: workload

Eight VM’s, each running:
# service xenballoond start
# cd src/linux-2.6.30
# while true; do
  make clean; make –j4;
  done
### performance analysis: data collection

```
Every 2.0s: ./.xenballoon-monitor -w; echo; xm list

<table>
<thead>
<tr>
<th>id</th>
<th>mem-kb</th>
<th>tgt-kb</th>
<th>commit</th>
<th>swpin</th>
<th>swout</th>
<th>PRSWPin</th>
<th>PRSWout</th>
<th>pgin</th>
<th>pgout</th>
<th>PRCCHIn</th>
<th>TMEMsec</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>141220</td>
<td>141220</td>
<td>141220</td>
<td>0</td>
<td>0</td>
<td>79356</td>
<td>127833</td>
<td>1216486</td>
<td>3884505</td>
<td>699707</td>
<td>4.8</td>
</tr>
<tr>
<td>3</td>
<td>139264</td>
<td>139264</td>
<td>139264</td>
<td>0</td>
<td>0</td>
<td>53745</td>
<td>72097</td>
<td>1172038</td>
<td>3794961</td>
<td>706929</td>
<td>4.7</td>
</tr>
<tr>
<td>4</td>
<td>139264</td>
<td>139264</td>
<td>139264</td>
<td>0</td>
<td>0</td>
<td>59579</td>
<td>97217</td>
<td>1096154</td>
<td>3362377</td>
<td>583137</td>
<td>4.3</td>
</tr>
<tr>
<td>5</td>
<td>140016</td>
<td>140016</td>
<td>140016</td>
<td>0</td>
<td>0</td>
<td>45689</td>
<td>75186</td>
<td>979922</td>
<td>2489733</td>
<td>353064</td>
<td>3.1</td>
</tr>
<tr>
<td>6</td>
<td>142180</td>
<td>142180</td>
<td>142180</td>
<td>0</td>
<td>0</td>
<td>42324</td>
<td>67270</td>
<td>1090886</td>
<td>2426817</td>
<td>319616</td>
<td>3.4</td>
</tr>
<tr>
<td>7</td>
<td>139264</td>
<td>139264</td>
<td>139264</td>
<td>0</td>
<td>0</td>
<td>55509</td>
<td>89453</td>
<td>956202</td>
<td>1837041</td>
<td>271033</td>
<td>2.5</td>
</tr>
<tr>
<td>8</td>
<td>141852</td>
<td>141852</td>
<td>141852</td>
<td>0</td>
<td>0</td>
<td>6912</td>
<td>13553</td>
<td>324238</td>
<td>776253</td>
<td>68401</td>
<td>0.9</td>
</tr>
<tr>
<td>9</td>
<td>139264</td>
<td>139264</td>
<td>139264</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>186050</td>
<td>32289</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>

Free memory: 540 MiB  Idle tmem: 0 MiB  Ephemeral tmem: 1963 MiB
```

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Mem</th>
<th>VCPUs</th>
<th>State</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain-0</td>
<td>0</td>
<td>256</td>
<td>8</td>
<td>r--------</td>
<td>2409.2</td>
</tr>
<tr>
<td>vm0</td>
<td>2</td>
<td>384</td>
<td>2</td>
<td>-b-------</td>
<td>3332.9</td>
</tr>
<tr>
<td>vm1</td>
<td>3</td>
<td>384</td>
<td>2</td>
<td>-b-------</td>
<td>3220.5</td>
</tr>
<tr>
<td>vm2</td>
<td>4</td>
<td>384</td>
<td>2</td>
<td>-b-------</td>
<td>3167.7</td>
</tr>
<tr>
<td>vm3</td>
<td>5</td>
<td>384</td>
<td>2</td>
<td>r--------</td>
<td>2617.0</td>
</tr>
<tr>
<td>vm4</td>
<td>6</td>
<td>384</td>
<td>2</td>
<td>r--------</td>
<td>1798.4</td>
</tr>
<tr>
<td>vm5</td>
<td>7</td>
<td>384</td>
<td>2</td>
<td>-b-------</td>
<td>1183.0</td>
</tr>
<tr>
<td>vm6</td>
<td>8</td>
<td>384</td>
<td>2</td>
<td>-b-------</td>
<td>30.0</td>
</tr>
<tr>
<td>vm7</td>
<td>9</td>
<td>384</td>
<td>2</td>
<td>-b-------</td>
<td>30.0</td>
</tr>
</tbody>
</table>
performance analysis: data overview

Quadr-core physical CPU, single socket, dual-thread

8 guests ← with 384MB → and 2 vcpus each

Staggered starts, some running for nearly a virtual hour, some just started ← ←
performance analysis: memory pressure

- Aggressive self-ballooning
  - launched as an initd service
  - feedback-driven using CommittedAS from /proc/meminfo

See: Magenheimer, D., Memory Overcommit… without the Commitment, Xen Summit 2008
performance analysis: kernel data

```
root@dmgene-nsVPN-dhcp-141-144-22-8:~$ Every 2.0s: ./.xenballoon-monitor -w; echo: xm list

<table>
<thead>
<tr>
<th>id</th>
<th>mem-kb</th>
<th>tgt-kb</th>
<th>commit</th>
<th>swpin</th>
<th>swout</th>
<th>PRSVPin</th>
<th>PRSWout</th>
<th>pgin</th>
<th>pgout</th>
<th>PRSCChin</th>
<th>TIMMssec</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>141220</td>
<td>141222</td>
<td>141220</td>
<td>0</td>
<td>0</td>
<td>19356</td>
<td>127833</td>
<td>1216486</td>
<td>3884505</td>
<td>699707</td>
<td>4.8</td>
</tr>
<tr>
<td>3</td>
<td>139264</td>
<td>139264</td>
<td>139264</td>
<td>0</td>
<td>0</td>
<td>51745</td>
<td>72097</td>
<td>1172038</td>
<td>3794961</td>
<td>700929</td>
<td>4.7</td>
</tr>
<tr>
<td>4</td>
<td>139264</td>
<td>139264</td>
<td>139264</td>
<td>0</td>
<td>0</td>
<td>59579</td>
<td>97217</td>
<td>1096154</td>
<td>3362377</td>
<td>283137</td>
<td>4.3</td>
</tr>
<tr>
<td>5</td>
<td>140016</td>
<td>140016</td>
<td>140016</td>
<td>0</td>
<td>0</td>
<td>45689</td>
<td>75186</td>
<td>975922</td>
<td>2489733</td>
<td>253064</td>
<td>3.1</td>
</tr>
<tr>
<td>6</td>
<td>142183</td>
<td>142183</td>
<td>142183</td>
<td>0</td>
<td>0</td>
<td>42324</td>
<td>67278</td>
<td>1090886</td>
<td>2426817</td>
<td>319616</td>
<td>3.4</td>
</tr>
<tr>
<td>7</td>
<td>139264</td>
<td>139264</td>
<td>139264</td>
<td>0</td>
<td>0</td>
<td>55609</td>
<td>89453</td>
<td>956202</td>
<td>1837041</td>
<td>710333</td>
<td>2.5</td>
</tr>
<tr>
<td>8</td>
<td>141852</td>
<td>141852</td>
<td>141852</td>
<td>0</td>
<td>0</td>
<td>6912</td>
<td>13553</td>
<td>324238</td>
<td>776253</td>
<td>68401</td>
<td>0.9</td>
</tr>
<tr>
<td>9</td>
<td>139264</td>
<td>139264</td>
<td>139264</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>180650</td>
<td>32289</td>
<td>0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Free memory: 540 MiB  Idle tmem: 0 MiB  Ephemeral tmem: 1933 MiB

Name   | ID    | Mem | VCPUs | State   | Time(s)
-------|-------|-----|-------|---------|-------
Domain-0|       |     |       |         |       
vmo   | 6     | 384 | 2     | r------ | 2617.0 
vml   | 7     | 384 | 2     | -b------ | 1798.4 
vm2   | 8     | 384 | 2     | -b------ | 1183.0 
vms   | 9     | 384 | 2     | -b------ | 30.0   
```

normal /proc/vmstat
data from each VM
performance analysis: tmem data

Important per-VM tmem statistics
performance analysis: precache

- Precache benefit
  - reduced page-in’s from external storage:
    - by 30-40%
    - by 200-250/sec

(note, precache does not reduce page-out’s... pgout shown only for reference)
performance analysis: preswap

- Preswap benefit
  - eliminated all physical swapping
  - reduced I/Os to external storage:
    - by 128K writes
    - by 80K reads
    - read+write: ~70/sec
performance analysis: CPU cost

- Tmem total cost
  - instrumented in hypervisor
  - approximation only
  - CPU overhead:
    - 0.08%-0.15%
performance analysis: summary

• Bottom line:

~0.1% of one core
saves ~300 io/sec

(depending on workload, your mileage may vary, objects appear smaller in the mirror, etc., etc.)
Agenda

• Motivation and Challenge
• Overview of Physical Memory Management
• Transcendent Memory Overview
• Transcendent Memory In Action
• Status, Future, etc.
Current Status

• precache and preswap patches for 2.6.30 posted to LKML
  • shared precache later
• Tmem checked-in to Xen *(xen-unstable tree, target Xen 3.5)*
  • enabled with xen boot option (off by default)
  • Linux tmem patch checked-in in 2.6.18-xen tree
  • save/restore and live migration support not yet implemented
• Tmem planned for upcoming Oracle VM release
  • experimental support ("technology preview")
Future Work

- continue “shared precache” work (ocfs2)
- shared-persistent pool investigation
  - inter-guest communication?
- real world performance measurement/analysis
  - identify any tuning opportunities and repeat
- save/restore/live migration (now done)
- tmem for:
  - native Linux?
  - Linux containers?
  - KVM?
  - Hvm guests?
  - Embedded?
Acknowledgements

- Chris Mason (Oracle)
  - Linux vfs changes for precache
- Zhigang Wang (Oracle)
  - Xen tools (xm + libxc) code
- Kurt Hackel, Dave McCracken (Oracle)
  - design and debugging help
- Sunil Mushran, Joel Becker (Oracle)
  - ocfs2 help for shared precache (still under development)
- Jeremy Fitzhardinge, Ian Pratt, Keir Fraser, Jan Beulich
  - hard questions and good advice
- Excellent feedback on lkml and xen-devel
For more information

http://oss.oracle.com/projects/tmem
Transcendent Memory on Linux

http://oss.oracle.com/projects/tmem

Speaker: Dan Magenheimer
Oracle Corporation

dan.magenheimer@oracle.com