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
Operating systems are memory hogs!

If you give an operating system more memory.....
Operating systems are memory hogs!

...it uses up any memory you give it!
OS Physical Memory Management

• What does an OS do with all that memory?
OS Physical Memory Management

• What does an OS do with all that memory?
OS Physical Memory Management

• What does an OS do with all that memory?
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

page cache

Everything else
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

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
• 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….

And but…
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
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 all fallow memory
- Step 1b: reclaim wasted guest memory (e.g. via ballooning)
- Step 1c: collect it all into a pool
Transcendent memory
creating the transcendent memory pool

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

Transcendent memory API
• paravirtualized (lightly)
• narrow
• well-specified
• operations are:
  • synchronous
  • page-oriented (one page per op)
  • copy-based
• multi-faceted
• extensible
Transcendent memory
four different subpool types → four different uses

<table>
<thead>
<tr>
<th></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>→ “hcache”</td>
<td>→ “hswap”</td>
</tr>
<tr>
<td>shared</td>
<td>server-side cluster filesystem cache?</td>
<td>inter-domain shared memory?</td>
</tr>
<tr>
<td></td>
<td>→ “shared hcache”</td>
<td></td>
</tr>
</tbody>
</table>

ephemeral, adj., ... transitory, existing only briefly, short-lived (i.e. NOT persistent)
Transcendent memory
caveats

• Requirements
  • guest OS must be paravirtualized
  • 64-bit hypervisor and CPU

• 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
Agenda

• Motivation and Challenge
• Overview of Physical Memory Management
• Transcendent Memory Overview
• Transcendent Memory In Action
  • private-ephemeral pool → “hcache”*
  • shared-ephemeral pool → “shared hcache”
  • private-persistent pool → “hswap”*
• Status, Future, etc.

* called “precache” and “preswap” for Linux
**hcach**

- 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>“hcach”</td>
<td>“hswap”</td>
</tr>
<tr>
<td><strong>shared</strong></td>
<td>server-side cluster filesystem cache?</td>
<td>inter-domain shared memory?</td>
</tr>
<tr>
<td></td>
<td>“shared hcach”</td>
<td></td>
</tr>
</tbody>
</table>
hcaché (with compression)

- Compression
  - Option (per-domain)
  - nominally doubles available memory
  - performance-space tradeoff
hcache (multiple guests)

- second-chance page cache for *multiple* guests

- Need “memory scheduler”:
  - global admission/eviction policy:
    - LRU queue, or
    - weight balanced *(future)*
**shared hcache** (for clustering)

- **guests** sharing a **clustered filesystem**
  - non-exclusive
  - LFU instead of LRU
  - compression optional
  - a server-side disk cache!

**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;</td>
<td>Fast swap</td>
</tr>
<tr>
<td></td>
<td>clean-page cache!</td>
<td>&quot;device&quot;!!</td>
</tr>
<tr>
<td></td>
<td>&quot;hcage&quot;</td>
<td>&quot;hswap&quot;</td>
</tr>
<tr>
<td>shared</td>
<td>server-side cluster</td>
<td>inter-domain</td>
</tr>
<tr>
<td>filesystem cache?</td>
<td></td>
<td>shared memory?</td>
</tr>
<tr>
<td></td>
<td>&quot;shared hcache&quot;</td>
<td></td>
</tr>
</tbody>
</table>
hswap

- over-ballooned guests experiencing unexpected memory pressure have an **emergency swap disk**
  - much faster than swapping
  - persistent (“dirty”) pages OK
  - prioritized higher than hcache
  - limited by domain’s maxmem

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></td>
<td>Fast swap “device”!!</td>
</tr>
<tr>
<td></td>
<td>“second-chance” clean-page cache!!</td>
<td>→ “hcache”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>→ “hswap”</td>
</tr>
<tr>
<td><strong>shared</strong></td>
<td>server-side cluster filesystem cache?</td>
<td>inter-domain shared memory?</td>
</tr>
<tr>
<td></td>
<td>→ “shared hcache”</td>
<td></td>
</tr>
</tbody>
</table>
Agenda

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

• hcache and hswap fully working
  • shared hcache soon
• xen-side patch ready for inclusion in xen-unstable
  • ~3K line patch, but low impact on existing code
  • enabled with xen boot option (off by default)
    • “technology preview”
  • goal: broader community usage (3.4?)
• linux-side patch ready
  • low impact on existing code
  • 2.6.18-xen version ready for inclusion in Xen-linux tree
  • 2.6.28 version working
Future Work

- finish “shared hcache” work (ocfs2)
- shared-persistent pool investigation
  - inter-domain communication?
- real world performance measurement/analysis
  - identify tuning opportunities (e.g. scaleability) and repeat
- finish “memory scheduler”
- tmem for:
  - native Linux?
  - Linux containers?
  - KVM?
  - Hvm domains?
Acknowledgements

- Chris Mason (Oracle)
  - Linux vfs changes for hcache
- Zhigang Wang (Oracle)
  - Xen tools (xm + libxc) code
- Kurt Hackel (Oracle), various HP friends, Ian, Keir, Jeremy
  - design feedback along the way
For more information

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

Speaker: Dan Magenheimer
Oracle Corporation
Transcendent Memory API

overview (API v0.0.1)
Transcendent memory API

op overview (API v0.0.1)

Two classes of operations:

• Create a pool
  Syntax: `pool_id = tmem_new_pool(uuid, flags)`

• Operate on a created pool
  Generic syntax:
  `retval = tmem_op(handle, pfn[, ofs1, ofs2, len])`
Transcendent memory API
pool creation (API v0.0.1)

Syntax: \texttt{pool\_id = tmem\_new\_pool(uuid, flags)}

<table>
<thead>
<tr>
<th></th>
<th>ephemeral</th>
<th>persistent</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>“second-chance” clean-page cache!! (\rightarrow \text{“hcache”})</td>
<td>Fast swap “device”!! (\rightarrow \text{“hswap”})</td>
</tr>
<tr>
<td>shared</td>
<td>server-side cluster filesystem cache?</td>
<td>inter-domain shared memory?</td>
</tr>
</tbody>
</table>

\*Implemented and working today (Linux + Xen)\*

Under investigation

flags: private vs. shared, ephemeral vs. persistent, page size, API version, … ???

\texttt{uuid: 128-bit “share name”} (for shared pools, ignored for private pools)
Transcendent Memory API
what is a “handle”?? (API v0.0.1)

```
retval = tmem_op(handle, pfn) → (is actually)
retval = tmem_op(pool_id, object_id, page_id, pfn)
```

- The “handle” used in previous slides is actually a three-element “handle-tuple” consisting of:
  - a 32-bit pool-id (obtained from `tmem_new_pool()`)  
  - a 64-bit object-id  
  - a 32-bit page-id  
- In filesystem-like usage:
  - pool-id → one per filesystem  
  - object-id → inode  
  - page-id → page index into a file
Transcendent Memory API

**API operations** (API v0.0.1)

- `tmem_new_pool(uuid, flags)`
- `tmem_destroy_pool(pool_id)`
- `tmem_put_page(pool_id, object_id, page_id, pfn)`
- `tmem_get_page(pool_id, object_id, page_id, empty_pfn)`
- `tmem_flush_page(pool_id, object_id, page_id)`
- `tmem_flush_object(pool_id, object_id)`
- `tmem_read(pool_id, object_id, page_id, pfn, offset1, offset2, len)`
- `tmem_write(pool_id, object_id, page_id, pfn, offset1, offset2, len)`
- `tmem_xchg(pool_id, object_id, page_id, pfn, offset1, offset2, len)`
- `tmem_control(TBD...)`
Transcendent Memory API

**important semantic details** (v0.0.1)

- **get_page** on a *private+ephemeral* pool is destructive (*auto-flush*)
  - implements *exclusive cache* semantics
- no serialization guarantees are provided for SMP VMs
- clients must ensure coherency with their own caches/data stores but implementation provides following guarantees:
  - **put/put/get** (aka “dup put”) coherency
    ```
    tmem_put_page(ABC, D1);
    tmem_put_page(ABC, D2);
    tmem_get_page(ABC, E);
    ```
    E may never contain the data from D1.
    (implies that on persistent pools, dup put must never fail)
  - **get/get coherency**
    ```
    tmem_get_page(ABC, E);
    tmem_get_page(ABC, E);
    ```
    If the first get fails, the second must also fail
- all **flush** operations must always succeed
- return values: >=0 means success, < 0 failure (errno)
- see spec for more information
Transcendent memory

hcacheperformance

(smaller is better)

Benchmark: Linux compile, cold page cache, pre-caching enabled (ccache)
Transcendent memory

**hcach**e compensates for underprovisioned memory

**Benchmark**: Linux compile, *warm* page cache, pre-caching *disabled*


**hcache** *(multiple domains + compressed)*

- *shared compressed* extended page cache for more than one guest