The Linux Virtual Memory System

Patrick Ladd
Technical Account Manager
pladd@redhat.com / pmladd@gmail.com

NY Red Hat Users Group
June 8, 2016

Slides are available at http://people.redhat.com/pladd/
Topics

• Evolution of the Linux Memory System
• Evolution of system memory architectures
• Latest innovations in Linux Memory System
  • NUMA
  • Hugepages
Virtual Memory
Virtual Memory System (simplified)

Virtual pages
- Cost "nothing"
- Basically unlimited on 64 bit architecture

Physical Pages
- RAM of system
- Cost money!

PageTables
- Virtual → Physical Mapping
PageTables

Data Structure
- Common code and x86 formats are the same
- All 4K in size
- $2^9$ (512) pointers per table
- `grep PageTables /proc/meminfo`

Accessible Memory
- Total:
  - $(2^9)^4 \times 4096 \rightarrow 48$ bits
  - 281.5 TB
Virtual Memory Fabric

Data structures for connecting

Hardware constrained structures
- pagetables

Software abstractions
- Tasks
- Processes
- Virtual memory areas (VMAs)
- `mmap (glibc malloc())`
Physical Page and `struct page`

Click to add subtitle

```
struct page
64 bytes
```

```
PAGE_SIZE (4K)
```

```
64 / 4096
(1.56% of RAM)
```

```
mem_map  Physical RAM ... 
```
Memory Management Algorithms
Memory System Heuristics

- Algorithms doing computations on memory fabric structures
- Need to solve hard problems with no guaranteed perfect solution
  - When is the right time to unmap pages? (swappiness)
  - Which page should I page out?
- Some design is unchanged
  - Measurement of how hard it is to free memory
  - Free memory used as cache
  - Overcommit by default
Page reclaim clock algorithm

Early kernels (~2.2) Early '90s
Small system RAM sizes

page[0]
page[0]
mem_map
page[n]
pgtable scan clock algorithm

Kernel 2.2 Mid 90's
Last Recently Used List

Kernel 2.2 Late 90's

![Diagram of Last Recently Used List for Kernel 2.2 Late 90's showing page references and page tables relationships.](image-url)
Active and Inactive List LRU

Kernel 2.4 - 2001

The active page LRU preserves the active memory working set:

- Only the inactive LRU loses information as fast as use-once I/O goes.
- Works well enough also with an arbitrary balance.
- Active/inactive list optimum balancing algorithm was solved in 2012-2014.
  - Shadow radix tree nodes that detect re-faults (more patches last month).
Active & Inactive LRU Lists

Kernel 2.4 Early 2000's

Use-once pages to trash

Use-many pages

pgtables
$ grep -i active /proc/meminfo
Active: 11192976 kB
Inactive: 2643936 kB
Active(anon): 10402692 kB
Inactive(anon): 2058248 kB
Active(file): 790284 kB
Inactive(file): 585688 kB
To free a candidate page, we must first drop all references to it (mark the page table entry non-present)
MM & VMA

- **mm_struct aka MM**
  - Memory of a process
  - Shared by all threads
- **vm_area_struct aka VMA**
  - Virtual memory area
  - Created and torn down by mmap & munmap
  - Defines the virtual address space of a MM
rmap Obsoletes pgtable Scan Clock Algorithm

Click to add subtitle

rmap – reverse mapping
- Allows direct connection to pagetables from any given physical page without scanning
If the user space program access the page, it will trigger a pagein/swapin
objrmap / anon-vma / ksm
Active & Inactive + rmap

Kernel 2.6 Late 2000's

Use-once pages to trash

Use-many pages
<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemTotal:</td>
<td>16054316 kB</td>
<td>SReclaimable:</td>
<td>644192 kB</td>
</tr>
<tr>
<td>MemFree:</td>
<td>1390476 kB</td>
<td>SUNreclaim:</td>
<td>218776 kB</td>
</tr>
<tr>
<td>MemAvailable:</td>
<td>2946740 kB</td>
<td>KernelStack:</td>
<td>23776 kB</td>
</tr>
<tr>
<td>Buffers:</td>
<td>112292 kB</td>
<td>PageTables:</td>
<td>147796 kB</td>
</tr>
<tr>
<td>Cached:</td>
<td>2652012 kB</td>
<td>NFS_Unstable:</td>
<td>0 kB</td>
</tr>
<tr>
<td>SwapCached:</td>
<td>39876 kB</td>
<td>Bounce:</td>
<td>0 kB</td>
</tr>
<tr>
<td>Active:</td>
<td>11228856 kB</td>
<td>WritebackTmp:</td>
<td>0 kB</td>
</tr>
<tr>
<td>Inactive:</td>
<td>2087104 kB</td>
<td>CommitLimit:</td>
<td>16096272 kB</td>
</tr>
<tr>
<td>Active(anon):</td>
<td>10440148 kB</td>
<td>Committed_AS:</td>
<td>32734512 kB</td>
</tr>
<tr>
<td>Inactive(anon):</td>
<td>1710336 kB</td>
<td>VmallocTotal:</td>
<td>34359738367 kB</td>
</tr>
<tr>
<td>Active(file):</td>
<td>788708 kB</td>
<td>VmallocUsed:</td>
<td>763212 kB</td>
</tr>
<tr>
<td>Inactive(file):</td>
<td>376768 kB</td>
<td>VmallocChunk:</td>
<td>34358783056 kB</td>
</tr>
<tr>
<td>Unevictable:</td>
<td>3188 kB</td>
<td>HardwareCorrupted:</td>
<td>0 kB</td>
</tr>
<tr>
<td>Mlocked:</td>
<td>3188 kB</td>
<td>AnonHugePages:</td>
<td>1280000 kB</td>
</tr>
<tr>
<td>SwapTotal:</td>
<td>8069116 kB</td>
<td>HugePages_Total:</td>
<td>0</td>
</tr>
<tr>
<td>SwapFree:</td>
<td>6731456 kB</td>
<td>HugePages_Free:</td>
<td>0</td>
</tr>
<tr>
<td>Dirty:</td>
<td>23076 kB</td>
<td>HugePages_Rsvd:</td>
<td>0</td>
</tr>
<tr>
<td>Writeback:</td>
<td>0 kB</td>
<td>HugePages_Surp:</td>
<td>0</td>
</tr>
<tr>
<td>AnonPages:</td>
<td>10531332 kB</td>
<td>Hugepagesize:</td>
<td>2048 kB</td>
</tr>
<tr>
<td>Mapped:</td>
<td>844892 kB</td>
<td>DirectMap4k:</td>
<td>305040 kB</td>
</tr>
<tr>
<td>Shmem:</td>
<td>1599248 kB</td>
<td>DirectMap2M:</td>
<td>15040512 kB</td>
</tr>
<tr>
<td>Slab:</td>
<td>862968 kB</td>
<td>DirectMap1G:</td>
<td>1048576 kB</td>
</tr>
</tbody>
</table>
More recent changes: Many More LRUs

- Separated LRU for anon and file backed mappings
- memcg (memory cgroups) introduced per-memcg LRUs
- Removal of un-freeable pages from LRUs
  - anonymous memory with no swap
  - mlocked memory
- Transparent Hugepages in the LRU increase scalability further
  (lru size decreased 512 times)
CPU Memory Architectures
Older Architectures

Direct memory bus
Older Architectures

Northbridge / Southbridge memory bus

- CPU
- FSB
- Northbridge
- Memory
- Southbridge
- I/O
- HS I/O (graphics / PCI-E)
NUMA Architecture

Multi-core Multi-Bus Architecture
NUMA – Non-Uniform Memory Architecture

- Multiple Nodes in a NUMA System
  - Each Node
    - CPU
    - Memory
    - PCI/Devices
  - All Nodes are interconnected
    - Interconnects are slow
  - Why NUMA?
Linux and NUMA
NUMA memory policies

{  
  MPOL_DEFAULT, /* no numactl */  
  MPOL_PREFERRED, /* --preferred=node */  
  MPOL_BIND, /* default numactl */  
  MPOL_INTERLEAVE, /* --interleave=nodes */  
  MPOL_LOCAL, /* --localalloc */  
}
Virt startup on CPU #0

MPOL_DEFAULT

Node #0

CPU #0

KVM

RAM #0

Node #1

CPU #1

RAM #1
Virt Mach allocates from RAM #0

MPOL_DEFAULT, no bindings
Scheduler CPU migration to #1

MPOL_DEFAULT, no bindings
Load goes away

Linux scheduler is blind – KVM will stay on CPU#1 with slow memory access
Hard NUMA bindings

- `man numactl`
- `man numastat`
- Kernel API
  - `sys_mempolicy`
  - `sys_mbind`
  - `sys_sched_setaffinity`
  - `sys_move_pages`
  - `/dev/cpuset`
- Full topology available in `/sys`

Numad can use the kernel API to monitor memory pressure and act accordingly.
# numastat -c qemu-kvm

Per-node process memory usage (in Mbs)

<table>
<thead>
<tr>
<th>PID</th>
<th>Node 0</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
<th>Node 4</th>
<th>Node 5</th>
<th>Node 6</th>
<th>Node 7</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>51722 (qemu-kvm)</td>
<td>68</td>
<td>16</td>
<td>357</td>
<td>6936</td>
<td>2</td>
<td>3</td>
<td>147</td>
<td>598</td>
<td>8128</td>
</tr>
<tr>
<td>51747 (qemu-kvm)</td>
<td>245</td>
<td>11</td>
<td>5</td>
<td>18</td>
<td>5172</td>
<td>2532</td>
<td>1</td>
<td>92</td>
<td>8076</td>
</tr>
<tr>
<td>53736 (qemu-kvm)</td>
<td>62</td>
<td>432</td>
<td>1661</td>
<td>506</td>
<td>4851</td>
<td>136</td>
<td>22</td>
<td>445</td>
<td>8116</td>
</tr>
<tr>
<td>53773 (qemu-kvm)</td>
<td>1393</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>6702</td>
<td>8114</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1769</td>
<td>463</td>
<td>2024</td>
<td>7462</td>
<td>10037</td>
<td>2672</td>
<td>169</td>
<td>7837</td>
<td>32434</td>
</tr>
</tbody>
</table>

No NUMA affinity
NUMA affinity

```
# numastat -c qemu-kvm
Per-node process memory usage (in Mbs)

<table>
<thead>
<tr>
<th>PID</th>
<th>Node 0</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
<th>Node 4</th>
<th>Node 5</th>
<th>Node 6</th>
<th>Node 7</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>51722 (qemu-kvm)</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>8072</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>8080</td>
</tr>
<tr>
<td>51747 (qemu-kvm)</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8113</td>
<td>0</td>
<td>8120</td>
</tr>
<tr>
<td>53736 (qemu-kvm)</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8110</td>
</tr>
<tr>
<td>53773 (qemu-kvm)</td>
<td>0</td>
<td>0</td>
<td>8050</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8051</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>0</strong></td>
<td><strong>0</strong></td>
<td><strong>8072</strong></td>
<td><strong>0</strong></td>
<td><strong>8072</strong></td>
<td><strong>0</strong></td>
<td><strong>8114</strong></td>
<td><strong>8110</strong></td>
<td><strong>32368</strong></td>
</tr>
</tbody>
</table>
```
Automatic NUMA balancing

Introduces “gravity” between CPU and memory

- Memory attracts CPU AGGRESSIVELY
  - If idle load balancing permits

- CPU attracts memory slowly
  - If anti-false sharing permits
NUMA example

Startup state
NUMA example

Converged state
NUMA threading example

Startup state

Thread 1
Thread 2
Thread 3
Thread 4

RAM #0
RAM #1
RAM
NUMA threading example

Converged state

Thread 1

Thread 2

RAM #0

RAM

RAM #1

Thread 3

Thread 4
Automatic NUMA balancing benchmark

- Intel SandyBridge (Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz)
  2 Sockets – 32 Cores with Hyperthreads
  256G Memory

- Software:
  - RHEV 3.6
  - Host bare metal – 3.10.0-327.el7 (RHEL7.2)
  - VM guest – 3.10.0-324.el7 (RHEL7.2)
  - VM – 32P, 160G (Optimized for Server)
  - Oracle – 12C, 128G SGA

- Storage – Violin 6616 – 16G Fibre Channel

- Test – Running Oracle OLTP workload with increasing user count and measuring Trans / min for each run as a metric for comparison
4VMs with different NUMA options
Automatic NUMA Balancing Configuration

- In RHEL7 Automatic NUMA balancing is enabled when:
  
  ```bash
  # numactl --hardware shows multiple nodes
  ```

- To disable automatic NUMA balancing:
  
  ```bash
  # echo 0 > /proc/sys/kernel/numa_balancing
  ```

- To enable automatic NUMA balancing:
  
  ```bash
  # echo 1 > /proc/sys/kernel/numa_balancing
  ```

- At boot:
  
  `numa_balancing=enable|disable`
Automatic NUMA balancing benchmark

- Intel SandyBridge (Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz)
  2 Sockets – 32 Cores with Hyperthreads
  256G Memory

- Software:
  - RHEV 3.6
  - Host bare metal – 3.10.0-327.el7 (RHEL7.2)
  - VM guest – 3.10.0-324.el7 (RHEL7.2)
  - VM – 32P , 160G (Optimized for Server)
  - Oracle – 12C , 128G SGA

- Storage – Violin 6616 – 16G Fibre Channel

- Test – Running Oracle OLTP workload with increasing user count and measuring Trans / min for each run as a metric for comparison
HugePages
HugePages

- Traditionally x86 hardware gave us 4KiB pages
- The more memory the bigger the overhead in managing 4KiB pages
- What if you had bigger pages?
  - 512 times bigger → 2MiB
Why HugePages?

- Improve CPU performance
  - Enlarge TLB size
  - Speed up TLB miss
    - Need 3 accesses to memory instead of 4 to refill the TLB
  - Faster to allocate memory initially (minor)
  - Page colouring inside the hugepage (minor)
  - Higher scalability of the page LRUs
- Cons
  - clear_page/copy_page less cache friendly
  - higher memory footprint sometime
  - Direct compaction takes time
TLB Miss cost: # of memory accesses

- novirt THP on
- novirt THP off
- host THP on guest THP on
- host THP off guest THP on
- host THP on guest THP off
- host THP off guest THP off
Conclusion & Questions
Recent Trends

- Large Memory Usage processes → HugePages (4KB → 2MB)
- Programs or Virtual Machines duplicating Memory → KSM
- Optimization of workloads for you, without manual tuning
  - Automatic NUMA balancing
  - Transparent HugePages
- Page pinning → MMU notifier
- Direct Managed private device memory (i.e. GPU) → UVM (unified virtual memory)
- 4th Layer of pagetables
- pagetables in high memory region
Slides are available at http://people.redhat.com/pladd/

THANK YOU

plus.google.com/+RedHat
linkedin.com/company/red-hat
youtube.com/user/RedHatVideos

facebook.com/redhatinc
twitter.com/RedHatNews