Coordinated and Efficient Huge Page Management with Ingens

Youngjin Kwon, Hangchen Yu, Simon Peter, Christopher J. Rossbach, and Emmett Witchel
High address translation cost

- Modern applications: large memory footprint, low memory access locality
- TLB coverage using base pages is insufficient

% of cpu cycles spent by page walk

<table>
<thead>
<tr>
<th>Cpu cycles</th>
<th>429.mcf</th>
<th>Graph analytics</th>
<th>SVM</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>10%</td>
<td>10%</td>
<td>10%</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>20%</td>
<td>20%</td>
<td>20%</td>
<td>20%</td>
<td>20%</td>
</tr>
<tr>
<td>30%</td>
<td>30%</td>
<td>30%</td>
<td>30%</td>
<td>30%</td>
</tr>
<tr>
<td>40%</td>
<td>40%</td>
<td>40%</td>
<td>40%</td>
<td>40%</td>
</tr>
<tr>
<td>50%</td>
<td>50%</td>
<td>50%</td>
<td>50%</td>
<td>50%</td>
</tr>
<tr>
<td>60%</td>
<td>60%</td>
<td>60%</td>
<td>60%</td>
<td>60%</td>
</tr>
<tr>
<td>70%</td>
<td>70%</td>
<td>70%</td>
<td>70%</td>
<td>70%</td>
</tr>
</tbody>
</table>
High address translation cost

- Virtualization requires additional address translation

Virtual address
  ↓
Guest page table

Guest physical address
  ↓
Host page table

Host physical address

% of cpu cycles spent by page walk

<table>
<thead>
<tr>
<th>Cpu cycles</th>
<th>%</th>
<th>429.mcf</th>
<th>Graph analytics</th>
<th>SVM</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host page table walk</td>
<td>70%</td>
<td>0%</td>
<td>10%</td>
<td>20%</td>
<td>30%</td>
</tr>
<tr>
<td>Guest page table walk</td>
<td>30%</td>
<td>90%</td>
<td>80%</td>
<td>70%</td>
<td>60%</td>
</tr>
</tbody>
</table>
Huge pages improve TLB coverage

- Architecture supports larger page size (e.g., 2MB page)
  - Intel: 0 to 1,536 entries in 2 years (2013 ~ 2015)
- Operating system has the burden of better huge page support

<table>
<thead>
<tr>
<th>Year</th>
<th>Sandy Bridge 2011</th>
<th>Ivy Bridge 2013</th>
<th>Haswell 2014</th>
<th>Skylake 2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLB coverage</td>
<td>0.01% 0.1%</td>
<td>0.01% 0.1%</td>
<td>0.05% 3.2%</td>
<td>0.11% 4.6%</td>
</tr>
</tbody>
</table>
Operating system support for huge pages

- OS transparently allocates/deallocates huge pages
- Huge pages in both guest and host

FreeBSD

Practical, transparent operating system support for superpages
Juan Navarro† Sitaram Iyer† Peter Druschel† Alan Cox†

OSDI '02

Linux

Transparent huge pages in 2.6.38
LWN.net, 2011
Huge pages improve performance

- Application speed up over using base pages only

**Bar Chart**

- Better Speed up
- 0% to 60% increments

Applications:
- 429 mcf (Spec CPU)
- Canneal (PARSEC)
- SVM (Liblinear)
- Graph analytics (PowerGraph)
- Machine learning (Spark MLlib)
- Web server (Cloudstone)
- Redis
- MongoDB
- Average
Are huge pages a free lunch?
WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.

WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled.
Disable Transparent Huge Pages (THP)

On this page

- Init Script
- Using tuned and ktune
- Test Your Changes

Transparent Huge Pages (THP) is a Linux memory management system that reduces the overhead of Translation Lookaside Buffer (TLB) lookups on machines with large amounts of memory by using larger memory pages.

However, database workloads often perform poorly with THP, because they tend to have sparse rather than contiguous memory access patterns. You should disable THP on Linux machines to ensure best performance with MongoDB.

# WARNING you have Transparent Huge Pages (THP)

Redis must be restarted after THP is disabled.
Are huge pages a free lunch?

Disable Transparent Huge Pages (THP)

Percona Database Performance Blog

Why TokuDB hates Transparent HugePages

Peter Zaitsev | July 23, 2014 | Posted in: MySQL, TokuDB

If you try to install the TokuDB storage engine on a modern Linux distribution it might fail with following error message:

2014-07-17 19:02:55 13865 [ERROR] Please disable them to continue.
2014-07-17 19:02:55 13865 [ERROR] (echo never > /sys/kernel/mm/transparent_hugepage/enable)

Redis must be restarted after THP is disabled.
Transparent Huge Pages: Thanks for your help...please don’t help

By the next morning CPU contention was worse.

The alarmingly high system CPU usage that we’d seen in the previous 3 months was always due to MySQL using kernel mutex. But since this problem, what the heck was this?

We discussed turning off TCMalloc, but that would’ve been a mistake. Implementing TCMalloc was a critical link in the chain of problems, ultimately strengthening our platform.

We discovered very quickly that the culprit this time was a *hugepaged* enabled by a Linux kernel flag called Transparent Huge Pages (default in most Linux distributions). Huge pages are designed to improve performance by helping the operating system manage large allocations of memory. They effectively increase the page size from the standard 4kb to 2MB or 1Gb (depending on how it is configured).

THP makes huge pages easier to use by, among other things, arranging your memory into larger chunks. It works great for app servers and memory-intensive operations.

**Disabling Transparent Hugepage Compaction**

Most Linux platforms supported by CDH 5 include a feature called transparent hugepage compaction which interacts poorly with Hadoop workloads and can seriously degrade performance.
Huge page pathologies in Linux

• High page fault latency
• Memory bloating
• Unfair huge page allocation
• Uncoordinated memory management
Huge page pathologies in Linux

• High page fault latency
• Memory bloating
• Unfair huge page allocation
• Uncoordinated memory management
Ingens
Efficient huge page management system

How to allocate huge pages?

<table>
<thead>
<tr>
<th>Problems</th>
<th>Linux</th>
<th>Ingens</th>
</tr>
</thead>
<tbody>
<tr>
<td>High page fault latency</td>
<td>Synchronous allocation</td>
<td>Asynchronous allocation</td>
</tr>
<tr>
<td>Memory bloating</td>
<td>Greedy allocation</td>
<td>Spatial utilization based allocation</td>
</tr>
</tbody>
</table>
High page fault latency
Huge page allocation increases page fault latency

- Page allocation path of both base and huge page

**Page fault latency**
- 4KB page: 3.6 us
- 2MB page: 378.0 us (mostly from page zeroing)
- Increases tail latency
Huge page allocation might require extra memory copying

- Page allocation path of huge page

**Diagram:**
- **Page fault handler**
  - Application pause
  - Allocate page(s)
  - Map the page(s) to page table
- **Physical memory manager**
  - Get page(s) from free page list
  - Zero the page(s)
- Application resume
Huge page allocation might require extra memory copying

- Page allocation path of huge page

Application pause → Allocate page(s) → Map the page(s) to page table

Physical memory manager

- Get page(s) from free page list
- Not enough contiguous memory

Zero the page(s)

Application resume
External fragmentation

Not enough contiguous memory
External fragmentation

- As system ages, physical memory is fragmented
  - 2 minutes to fragment 24 GB
  - All memory sizes eventually fragment
- Linux compacts physical memory to create contiguous pages

Not enough contiguous memory

Huge page boundary

Virtual address

Physical address

Allocated Base page
External fragmentation

- As system ages, physical memory is fragmented
  - 2 minutes to fragment 24 GB
  - All memory sizes eventually fragment

- Linux compacts physical memory to create contiguous pages

Huge page boundary

Virtual address

Physical address

Not enough contiguous memory

Allocated Base page
External fragmentation

- As system ages, physical memory is fragmented
  - 2 minutes to fragment 24 GB
  - All memory sizes eventually fragment
- Linux compacts physical memory to create contiguous pages

Huge page boundary

Virtual address

Physical address

Allocated Base page
External fragmentation

Not enough contiguous memory
Huge page allocation might require extra memory copying

- Page allocation path of huge page includes memory compaction

**Diagram:**

- **Page fault handler**
  - Application pause
  - Allocate page(s)
  - Map the page(s) to page table
  - Application resume

- **Physical memory manager**
  - Get page(s) from free page list
  - Zero the page(s)
  - Not enough contiguous memory
Huge page allocation might require extra memory copying

- Page allocation path of huge page includes memory compaction

---

**Page fault handler**

- Application pause
- Allocate page(s)
- Map the page(s) to page table

**Physical memory manager**

- Get page(s) from free page list
- Not enough contiguous memory
- Compact physical memory
- Zero the page(s)

- Application resume
Huge page allocation might require extra memory copying

- Page allocation path of huge page includes memory compaction

![Diagram]

- Application pause
  - Allocate page(s)
  - Map the page(s) to page table
  - Application resume

- Physical memory manager
  - Get page(s) from free page list
  - Not enough contiguous memory
  - Compact physical memory
  - Compaction may or may not succeed
  - Zero the page(s)
In gens: asynchronous allocation

- Page fault handler only allocates base pages
- Huge page allocation in background
- Memory compaction in background
- No extra page fault latency
  - No huge page zeroing
  - No compaction

Fast page fault handling
Page fault latency experiment

• Machine specification
  • Two Intel Xeon E5-2640 2.60GHz CPUs
  • 64GB memory and two 250 MB SSDs

• Cloudstone workload (latency sensitive)
  • Web service for social event planning
  • nginx/PHP/MySQL running in virtual machines
  • 85% read, 10% login, 5% write workloads
  • 2 of 7 web pages modified to use modern web page sizes
  • The average web page is 2.1 MB
    https://www.soasta.com/blog/page-bloat-average-web-page-2-mb/
Cloudstone result

Throughput (requests/s)

<table>
<thead>
<tr>
<th></th>
<th>Linux</th>
<th>Ingens</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>922.3</td>
<td>1091.9 (+18%)</td>
</tr>
</tbody>
</table>

Latency (millisecond)

- Memory is highly fragmented
- Ingens reduces
  - average latency up to 29.2%
  - tail latency up to 41.4%
- Linux page fault handler performs 461,383 memory compactions
Cloudstone result

- Memory is highly fragmented
- Ingens reduces
  - average latency up to 29.2%
  - tail latency up to 41.4%
- Linux page fault handler performs 461,383 memory compactions
Memory bloating

Application occupies more memory than it uses
Internal fragmentation

- Greedy allocation in Linux
  - Allocate a huge page on first fault to huge page region
  - The huge page region may not be fully used
- Greedy allocation causes severe internal fragmentation
  - Memory use often sparse

![Diagram showing virtual and physical addresses with huge page regions and boundaries.](image-url)
Memory bloating experiment

- Redis
  - Delete 70% objects after populating 8KB objects

- MongoDB
  - 15 million get requests for 1KB object with YCSB

<table>
<thead>
<tr>
<th></th>
<th>Using huge page</th>
<th>Using only base page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redis</td>
<td>20.7GB (+69%)</td>
<td>12.2GB</td>
</tr>
<tr>
<td>MongoDB</td>
<td>12.4GB (+23%)</td>
<td>10.1GB</td>
</tr>
</tbody>
</table>

Physical memory consumption

Bloating makes memory consumption unpredictable
Memory-intensive applications can’t provision to avoid swap
Ingens: Spatial utilization based allocation

- Ingens monitors spatial utilization of each huge page region

- Utilization-based allocation
  - Page fault handler requests promotion when the utilization is beyond a threshold (e.g., 90%)
  - Bounds the size of internal fragmentation
Redis memory bloating experiment

Physical memory consumption

- Linux (base only): 12.2 GB
- Ingens: 12.3 GB
- Linux (huge): 20.7 GB

Better

GET throughput

- Linux (base only): 19.0K
- Ingens: 20.9K
- Linux (huge): 21.7K

Better

+ 10%

- 4%

Huge: 2MB page
Base: 4KB page
Ingens overhead

- Overhead for memory intensive application

<table>
<thead>
<tr>
<th>Application</th>
<th>429.mcf</th>
<th>Graph</th>
<th>Spark</th>
<th>Canneal</th>
<th>SVM</th>
<th>Redis</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.9%</td>
<td>0.9%</td>
<td>0.6%</td>
<td>1.9%</td>
<td>1.3%</td>
<td>0.2%</td>
<td>0.6%</td>
</tr>
</tbody>
</table>

- Overhead for non-memory intensive application

<table>
<thead>
<tr>
<th>Application</th>
<th>Kernel build</th>
<th>Grep</th>
<th>Parsec 3.0 Benchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.2%</td>
<td>0.4%</td>
<td>0.8%</td>
</tr>
</tbody>
</table>

Ingens overhead is negligible
Ingens
Make huge pages widely used in practice

<table>
<thead>
<tr>
<th>Linux</th>
<th>Ingens</th>
<th>Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous allocation</td>
<td>Asynchronous allocation</td>
<td>No extra page fault latency</td>
</tr>
<tr>
<td>Greedy allocation</td>
<td>Spatial utilization based allocation</td>
<td>Bound memory bloating</td>
</tr>
</tbody>
</table>

Source code is available at https://github.com/ut-osu/ingens
Backup slides
Other operating systems

- Window, MacOS
  - Does not support transparent huge page

- FreeBSD
  - Very conservative approach
  - No memory compaction functionality
  - Performance speedup in Linux and FreeBSD

<table>
<thead>
<tr>
<th></th>
<th>SVM</th>
<th>Canneal</th>
<th>Redis</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeBSD</td>
<td>1.28</td>
<td>1.13</td>
<td>1.02</td>
</tr>
<tr>
<td>Linux</td>
<td>1.30</td>
<td>1.21</td>
<td>1.15</td>
</tr>
<tr>
<td>Ingens</td>
<td>1.29</td>
<td>1.19</td>
<td>1.15</td>
</tr>
</tbody>
</table>
Operating system support for huge pages

- User-controlled huge page management
  - Admin reserves huge page in advance
  - New APIs for memory allocation/deallocation
  - It could fail to reserve huge pages when memory is fragmented

- Transparent huge page management
  - Developers do not know about huge page
  - OS Transparently allocates/deallocates huge pages
  - OS manages memory fragmentation