A comprehensive analysis of superpage management mechanisms and policies

Weixi Zhu, Alan L. Cox, Scott Rixner
{wxzhu, alc, rixner}@rice.edu
Department of Computer Science, Rice University
Superpages benefit large-memory Applications’ performance

• Large memory applications have high address translation overhead
• Using superpages can reduce address translation overhead
• Many challenges in implementing transparent superpage support in the operating system – can cause performance regression
Contributions of this paper

1. Developed a comprehensive scheme for describing the design space
2. Presented novel insights from existing systems – Linux, FreeBSD, Ingens and HawkEye
3. Proposed Quicksilver based on FreeBSD, driven by our novel insights

https://github.com/rice-systems/quicksilver
X86-64 4KB-page address translation

Load/store (virtual address)

Miss (VA)

TLB

Hit (PA)

Cache

CPU

MMU (CR3)

L1 page table (512 entries)

L2

...

...

L3

...

...

L4

...

...

Physical address
(4 memory accesses)
Translation Look-aside Buffers (TLBs)

• Caches 4KB/2MB page mappings
• Typical capacity: 1536 entries in Intel Skylake STLB
• Fewer TLB misses -> fewer page walks -> better performance
Benefits of Superpages (2MB)

Address translation benefits
• Cheaper page walk cost: 4 -> 3 memory accesses
• Significantly increased TLB coverage: 6MB -> 3GB
  • Intel Skylake STLB: $1536 \times (4\text{KB} \times 2\text{MB}) = 6\text{MB} \times 3\text{GB}$
  • Reduced # TLB misses (page walks) -> better performance

OS-level benefits
• Reduced number and average cost of page faults
Drawbacks of Superpages (2MB)

• Underutilization
  • Waste free memory, causing memory bloat
  • Waste CPU time preparing unused memory

• Allocation is easier to fail under fragmentation
  • Require 2MB-aligned free contiguous physical memory

• Latency spikes
  • Preparing a 2MB page (e.g. zeroing or disk-reading) is much more costly
Contributions of this paper

1. Developed a comprehensive scheme for describing the design space
2. Presented novel insights from existing systems – Linux, FreeBSD, Ingens and HawkEye
3. Proposed Quicksilver based on FreeBSD, driven by our novel insights

https://github.com/rice-systems/quicksilver
Five decoupled events of superpage lifetime
-- To help understand OS superpage management

<table>
<thead>
<tr>
<th>Event</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical allocation</td>
<td>Acquisition of a free physical superpage</td>
</tr>
<tr>
<td>Physical preparation</td>
<td>Incremental or full preparation of the initial data for an allocated physical superpage</td>
</tr>
<tr>
<td>Mapping creation</td>
<td>Creation of a virtual superpage in a process’s address space and mapping it to a fully prepared physical superpage</td>
</tr>
<tr>
<td>Mapping destruction</td>
<td>Destruction of a virtual superpage mapping</td>
</tr>
<tr>
<td>Physical deallocation</td>
<td>Partial or full deallocation of an allocated physical superpage</td>
</tr>
</tbody>
</table>
Implementation choices

• Sync vs. Async allocation
  • During page fault time
  • When scanning page tables

• Incremental vs. full preparation
  • 4KB at a time
  • 2MB all at once

• In-place vs. out-of-place mapping (4KB->2MB promotion)
  • In-place promotion requires tracking allocated physical superpage
  • Out-of-place promotion involves migrating used pages to a different allocated physical superpage
Contributions of this paper

1. Developed a comprehensive scheme for describing the design space
2. Presented novel insights from existing systems – Linux, FreeBSD, Ingens and HawkEye
3. Proposed Quicksilver based on FreeBSD, driven by our novel insights

https://github.com/rice-systems/quicksilver
## Existing designs in 5-event design space

<table>
<thead>
<tr>
<th>Events</th>
<th>Linux</th>
<th>Ingens (Linux-based)</th>
<th>HawkEye (Linux-based)</th>
<th>FreeBSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation</td>
<td>Sync upon first page fault, or async for regions with utilization &gt; 0. Defragmenting if necessary</td>
<td>Only async for regions with utilization &gt; 90%, round-robin among processes</td>
<td>Only async for regions with utilization &gt; 0, with fine-grained order</td>
<td>Upon first page fault (tracked by reservation system)</td>
</tr>
<tr>
<td>Preparation</td>
<td>Coupled with allocation, sync or async, full</td>
<td>Coupled with allocation, only async, full</td>
<td>Same as left</td>
<td>Incrementally prepares in-place 4KB pages on page faults</td>
</tr>
<tr>
<td>Mapping</td>
<td>Coupled with preparation, sync or async</td>
<td>Coupled with preparation. Async and out-of-place</td>
<td>Same as left</td>
<td>After the last page preparation. Sync and in-place</td>
</tr>
<tr>
<td>Unmapping</td>
<td>Upon freeing, partial or full mapping change</td>
<td>Same as left</td>
<td>Same as left</td>
<td>Same as left</td>
</tr>
<tr>
<td>Deallocation</td>
<td>Upon superpage unmapping</td>
<td>Same as left</td>
<td>Same as left</td>
<td>Deferred as long as possible</td>
</tr>
</tbody>
</table>
Observation #1: coupling physical allocation, preparation and mapping creation brings more drawbacks

System: Linux

Benefit: Immediate address translation benefits and fewer page faults
--- Best performance on freshly booted machine

Multiple Drawbacks:
• Easy to create underutilized superpages and bloat memory
• Fail to create superpages for growing heap, e.g. 602.gcc_s in SpecCPU-2017
  • Allocations will fail when the 2MB virtual region is not covered.
• Cannot easily choose between 2 superpage sizes, e.g. 64KB and 2MB in ARM
• Cannot extend to 1GB superpages or file-backed superpages (higher full preparation cost)
Observation #2: asynchronous out-of-place promotion delays superpage mapping creation

Systems: Ingens (Linux-based), HawkEye (Linux-based)

Benefit: Alleviate latency spikes from costly page faults

Drawbacks:

• Preparation involves costly page migrations (the asynchronously allocated superpage is out-of-place)

• Superpage mapping creation is delayed – much slower than in-place promotion (FreeBSD)

<table>
<thead>
<tr>
<th>Speedups</th>
<th>Linux</th>
<th>Ingens</th>
<th>HawkEye</th>
<th>FreeBSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphChi: PageRank</td>
<td>1</td>
<td>0.58</td>
<td>0.53</td>
<td>0.77</td>
</tr>
<tr>
<td>BlockSVM: classification</td>
<td>1</td>
<td>0.81</td>
<td>0.73</td>
<td>0.96</td>
</tr>
</tbody>
</table>
Observation #3: Reservation-based policies enables speculative physical allocation, multiple page sizes and in-place promotion

System: FreeBSD

Requirement: A reservation system that tracks allocated physical superpages

Benefits:

• Decoupled allocation and preparation – enables speculative allocation for growing heaps (602.gcc_s), incremental preparation and in-place promotion
• Obviating need of async out-of-place promotion – can allocate physical superpages for growing heaps
• Supporting multiple page sizes
Observation #4: Reservations and delaying partial deallocation fight fragmentation

System: FreeBSD

Benefit:

- Less memory fragmentation from delayed partial deallocation – individual 4KB pages are less likely reallocated for other purpose
- No latency spikes – Linux’s memory compaction during page faults result in latency spikes in server workloads.
Observation #5: Bulk zeroing is consistently more efficient on modern processors

Typical zeroing: 512 calls of zeroing assembly code with size of 4KB
Bulk zeroing: Fewer calls of zeroing assembly code with bulk size > 4KB

Latency (us) of 2MB zeroing: drops consistently with larger bulk sizes

<table>
<thead>
<tr>
<th>CPU (GHz)</th>
<th>temporal</th>
<th></th>
<th></th>
<th>Non-temporal</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4KB</td>
<td>32KB</td>
<td>2MB</td>
<td>4KB</td>
<td>32KB</td>
<td>2MB</td>
</tr>
<tr>
<td>Bulk Size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E3-1231v3 (3.4)</td>
<td>92</td>
<td>88</td>
<td>87</td>
<td>114</td>
<td>99</td>
<td>97</td>
</tr>
<tr>
<td>E3-1245v6 (3.7)</td>
<td>84</td>
<td>67</td>
<td>65</td>
<td>92</td>
<td>74</td>
<td>71</td>
</tr>
<tr>
<td>E5-2640v3 (2.6)</td>
<td>355</td>
<td>287</td>
<td>280</td>
<td>154</td>
<td>112</td>
<td>106</td>
</tr>
<tr>
<td>E5-2640v4 (2.4)</td>
<td>409</td>
<td>334</td>
<td>325</td>
<td>163</td>
<td>113</td>
<td>106</td>
</tr>
<tr>
<td>R7-2700X (4.3)</td>
<td>185</td>
<td>183</td>
<td>159</td>
<td>99</td>
<td>60</td>
<td>53</td>
</tr>
</tbody>
</table>
Contributions of this paper

1. Developed a comprehensive scheme for describing the design space
2. Presented novel insights from existing systems – Linux, FreeBSD, Ingens and HawkEye
3. Proposed Quicksilver based on FreeBSD, driven by our novel insights

https://github.com/rice-systems/quicksilver
Quicksilver – guided by novel observations

• Allocation: allocates a reservation speculatively upon first page fault
• Preparation: incrementally prepares 4KB on demand, performs a synchronous full preparation upon a utilization threshold (Sync-1, Sync-64) – match or beat Linux’s performance
• Mapping: Relaxed for more file-backed mappings
• Unmapping: same as FreeBSD
• Deallocation: delayed until the superpage is inactive, then asynchronously evicts 4KB pages to perform a whole deallocation
Evaluation of Quicksilver

- Performance of a wide variety of workloads
  - on a freshly booted machine
  - on a heavily fragmented machine
- Throughput and tail latency of server workloads
- A parallel compilation task with many small jobs
Quicksilver Beats Linux on a freshly-booted machine

<table>
<thead>
<tr>
<th>Frag-0</th>
<th>GUPS</th>
<th>Graphchi-PR</th>
<th>BlockSV M</th>
<th>XSBench</th>
<th>ANN</th>
<th>Canneal</th>
<th>Freqmine</th>
<th>Gcc</th>
<th>mcf</th>
<th>Dsjeng</th>
<th>XZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Ingens</td>
<td>0.87</td>
<td>0.58</td>
<td>0.81</td>
<td>0.98</td>
<td>1</td>
<td>0.95</td>
<td>0.99</td>
<td>1</td>
<td>0.99</td>
<td>0.99</td>
<td>0.96</td>
</tr>
<tr>
<td>HawkEye</td>
<td>0.28</td>
<td>0.53</td>
<td>0.73</td>
<td>0.88</td>
<td>1</td>
<td>0.95</td>
<td>0.99</td>
<td>0.99</td>
<td>0.94</td>
<td>0.86</td>
<td>0.9</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>0.96</td>
<td>0.77</td>
<td>0.96</td>
<td>0.99</td>
<td>0.98</td>
<td>1.14</td>
<td>1</td>
<td>1.05</td>
<td>0.99</td>
<td>1</td>
<td>0.99</td>
</tr>
<tr>
<td>Sync-1</td>
<td>0.99</td>
<td>1.07</td>
<td>1</td>
<td>1</td>
<td>1.07</td>
<td>1.14</td>
<td>0.99</td>
<td>1.05</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sync-64</td>
<td>0.98</td>
<td>1.05</td>
<td>1</td>
<td>1</td>
<td>1.08</td>
<td>1.14</td>
<td>0.99</td>
<td>1.05</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Linux is no longer the best on a freshly-booted machine!
Quicksilver outperforms every other systems under severe memory fragmentation

2.18x speedup on PageRank task!
Quicksilver obtains high throughput without latency spikes

Throughput (GBps) and 95th tail latency (ms) of Redis workloads

<table>
<thead>
<tr>
<th>Cold-start</th>
<th>Linux-4KB</th>
<th>Linux</th>
<th>Ingens</th>
<th>HawkEye</th>
<th>FreeBSD</th>
<th>Sync-1</th>
<th>Sync-64</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frag-0</td>
<td>1.04 (5.6)</td>
<td>1.34 (4.1)</td>
<td>1.00 (5.9)</td>
<td>1.00 (5.9)</td>
<td>1.11 (6.1)</td>
<td>1.26 (4.5)</td>
<td>1.20 (4.8)</td>
</tr>
<tr>
<td>Frag-50</td>
<td>1.04 (5.7)</td>
<td>0.92 (10.2)</td>
<td>0.95 (5.9)</td>
<td>1.02 (5.9)</td>
<td>1.04 (6.2)</td>
<td>1.25 (4.5)</td>
<td>1.27 (4.7)</td>
</tr>
<tr>
<td>Frag-100</td>
<td>1.07 (5.6)</td>
<td>0.81 (9.9)</td>
<td>0.94 (6.1)</td>
<td>1.00 (5.8)</td>
<td>0.98 (6.5)</td>
<td>1.31 (4.5)</td>
<td>1.26 (4.6)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Warm-start</th>
<th>Linux-4KB</th>
<th>Linux</th>
<th>Ingens</th>
<th>HawkEye</th>
<th>FreeBSD</th>
<th>Sync-1</th>
<th>Sync-64</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frag-0</td>
<td>1.06 (6.5)</td>
<td>1.32 (5.2)</td>
<td>1.23 (5.7)</td>
<td>1.03 (6.7)</td>
<td>1.30 (5.6)</td>
<td>1.32 (5.5)</td>
<td>1.31 (5.5)</td>
</tr>
<tr>
<td>Frag-50</td>
<td>1.07 (6.5)</td>
<td>1.17 (5.9)</td>
<td>1.09 (6.4)</td>
<td>1.03 (6.7)</td>
<td>1.18 (6.1)</td>
<td>1.32 (5.5)</td>
<td>1.32 (5.5)</td>
</tr>
<tr>
<td>Frag-100</td>
<td>1.07 (6.5)</td>
<td>1.16 (5.9)</td>
<td>1.01 (6.9)</td>
<td>1.05 (6.6)</td>
<td>1.10 (6.6)</td>
<td>1.33 (5.4)</td>
<td>1.34 (5.5)</td>
</tr>
</tbody>
</table>

Also observe low memory bloat on Quicksilver
Quicksilver (Sync-64) avoids creating underutilized superpages

FreeBSD Kernel compilation task (make buildkernel –j9):

<table>
<thead>
<tr>
<th>Buildkernel</th>
<th>real</th>
<th>user</th>
<th>sys</th>
<th># superpages</th>
<th># page faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync-1</td>
<td>197.7</td>
<td>1409.4</td>
<td>89.4</td>
<td>200.5 K</td>
<td>5.3 M</td>
</tr>
<tr>
<td>Sync-64</td>
<td>196.9</td>
<td>1408.8</td>
<td>78.5</td>
<td>99.6 K</td>
<td>10.3 M</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>203.7</td>
<td>1436.7</td>
<td>98</td>
<td>36.9 K</td>
<td>30.2 M</td>
</tr>
</tbody>
</table>

Sync-1 creates 100.9 K underutilized superpages with average utilization < 50 4KB pages
-- Sync-64 is as competitive as Sync-1, but also avoids underutilized superpages
Takeaways from this paper

- Our comprehensive scheme allow comparing and contrasting superpage management policies
- Our novel insights motivated Quicksilver’s innovative design
- Quicksilver obtains benefits of aggressive superpage allocation, with mitigated memory bloat and fragmentation issues that arise from underutilized superpages
- Sync-64 and Sync-1 can both match or beat existing systems in both lightly and heavily fragmented scenarios, in terms of application performance, tail latency and memory bloat
- Sync-64 avoids creating underutilized superpages and is preferable for long-running servers
Thank you

For more details, please check our ATC-2020 paper.
Quicksilver source code: https://github.com/rice-systems/quicksilver