Optimizing Linux kernel swap subsystem

Huang Ying, Intel OTC
Oct, 2016

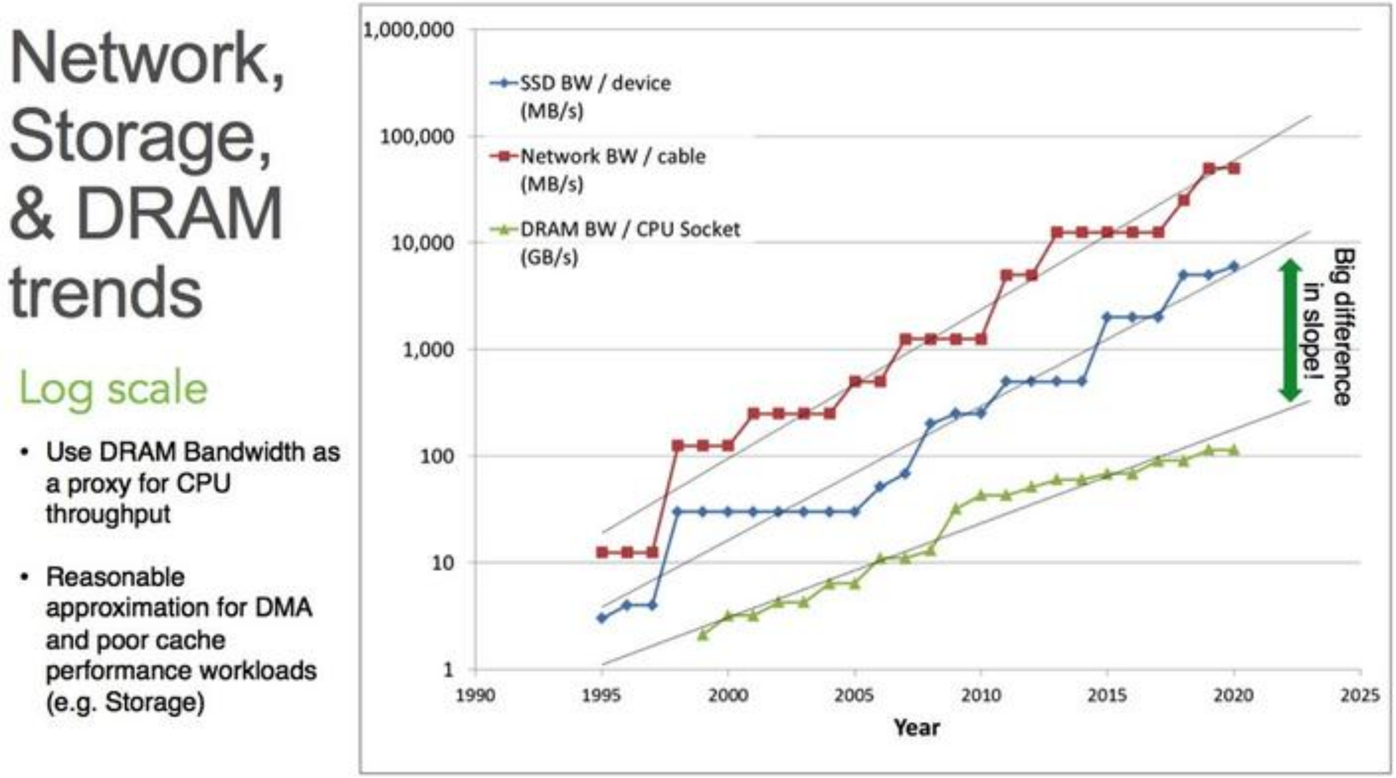
- Introduction to swap
- Swap implementation
- Swap IO performance optimization
- Other swap optimization
- Conclusion

What is swap?

- Run more/bigger applications
- Move RAM pages not needed in near future to disk, and move it back later when needed
- Performance
  - Acceptable for some use cases
    - For example, switch applications seldom
  - SSD is a good fit for swap
    - much lower latency than HDD

Opportunities

- Storage device performance keeps increasing



Network, Storage, & DRAM trends

Log scale

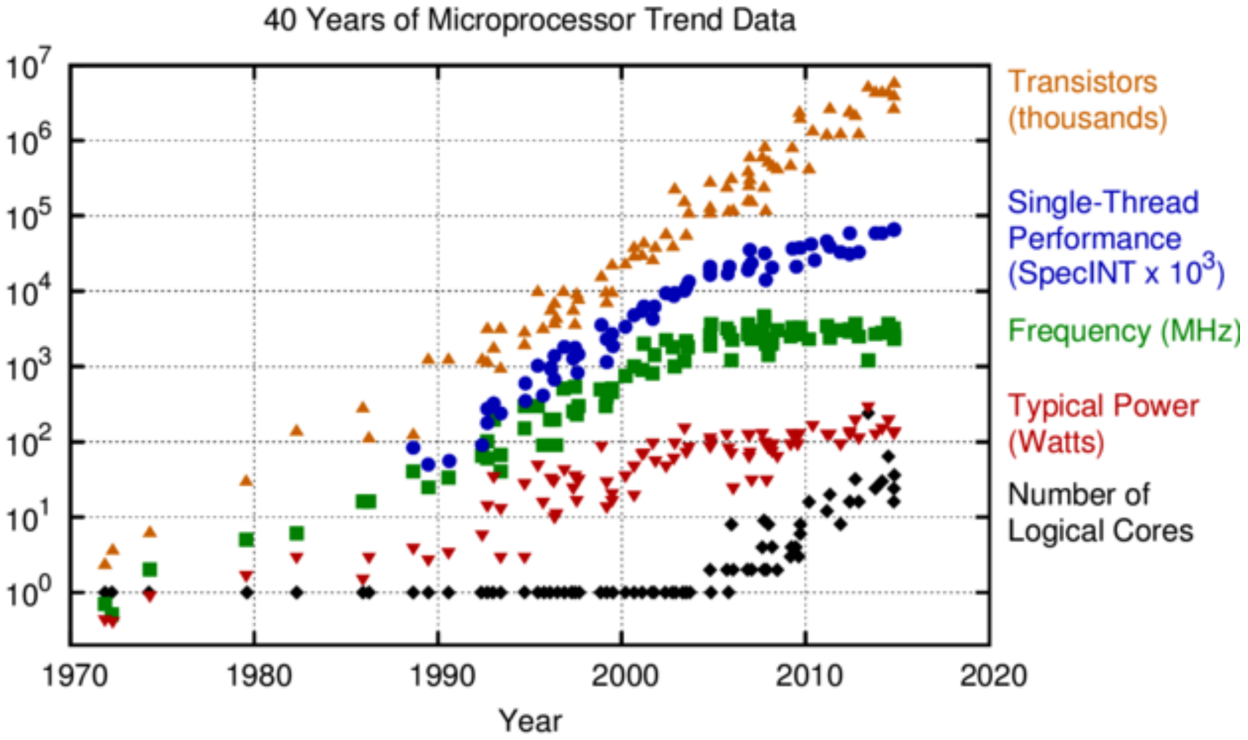
- Use DRAM Bandwidth as a proxy for CPU throughput
- Reasonable approximation for DMA and poor cache performance workloads (e.g. Storage)

https://itblog.sandisk.com/cpu-bandwidth-the-worrisome-2020-trend/

Challenges

- Single core/thread performance increases slowly
- CPU core/thread number increases rapidly

=> Scalability is key for swap performance



40 Years of Microprocessor Trend Data

Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

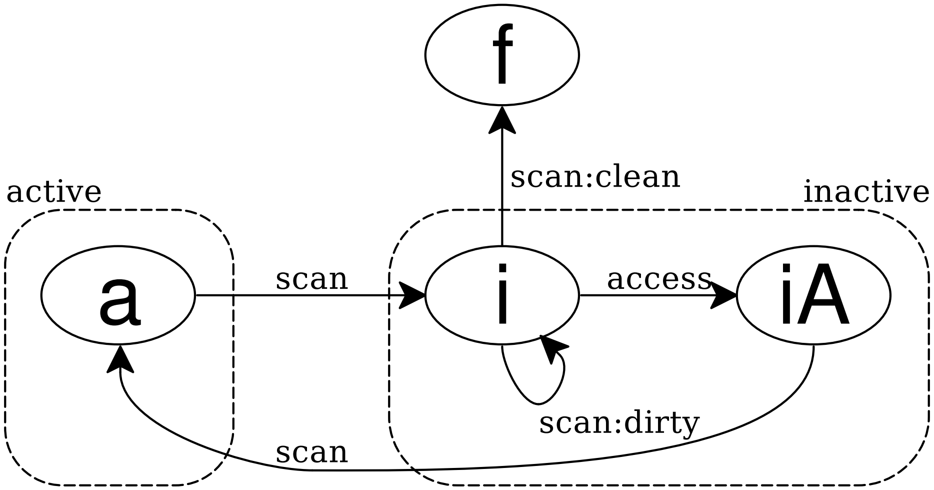
https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/

- Introduction to swap
- Swap implementation
- Swap IO performance optimization
- Other swap optimization
- Conclusion

Page reclaim

- Triggered when free memory becomes low
  - High watermark -> Low watermark
- File pages vs. Anonymous pages
  - Swap is used for anonymous pages
- data structure
  - Per-zone active/inactive LRU list
  - page->flags: Active
  - page table: Accessed bit

Page reclaim - State machine



```
a:  Active
i:  Inactive
iA: Inactive Accessed
f:  Free
```

Linux swap implementation - Data structure

- Page table
  - Map to page <-> map to swap entry
- **`swap_info_struct`**: swap space management
  - **`swap_map`**: array of usage count
    - One usage count for each swap entry
  - **`cluster_info`**: array of clusters
    - cluster: a set of swap entries
- Swap cache: "file cache" for anonymous pages
  - Radix tree: swap entry -> page

Linux swap implementation - Code flow

- swap out

```
select page to reclaim
allocate swap space
add page to swap cache
unmap page
write page to swap device
remove page from swap cache
free page
```

- swap in

```
page fault
allocate page
add page to swap cache
read in page
map page
remove page from swap cache
free swap space
```

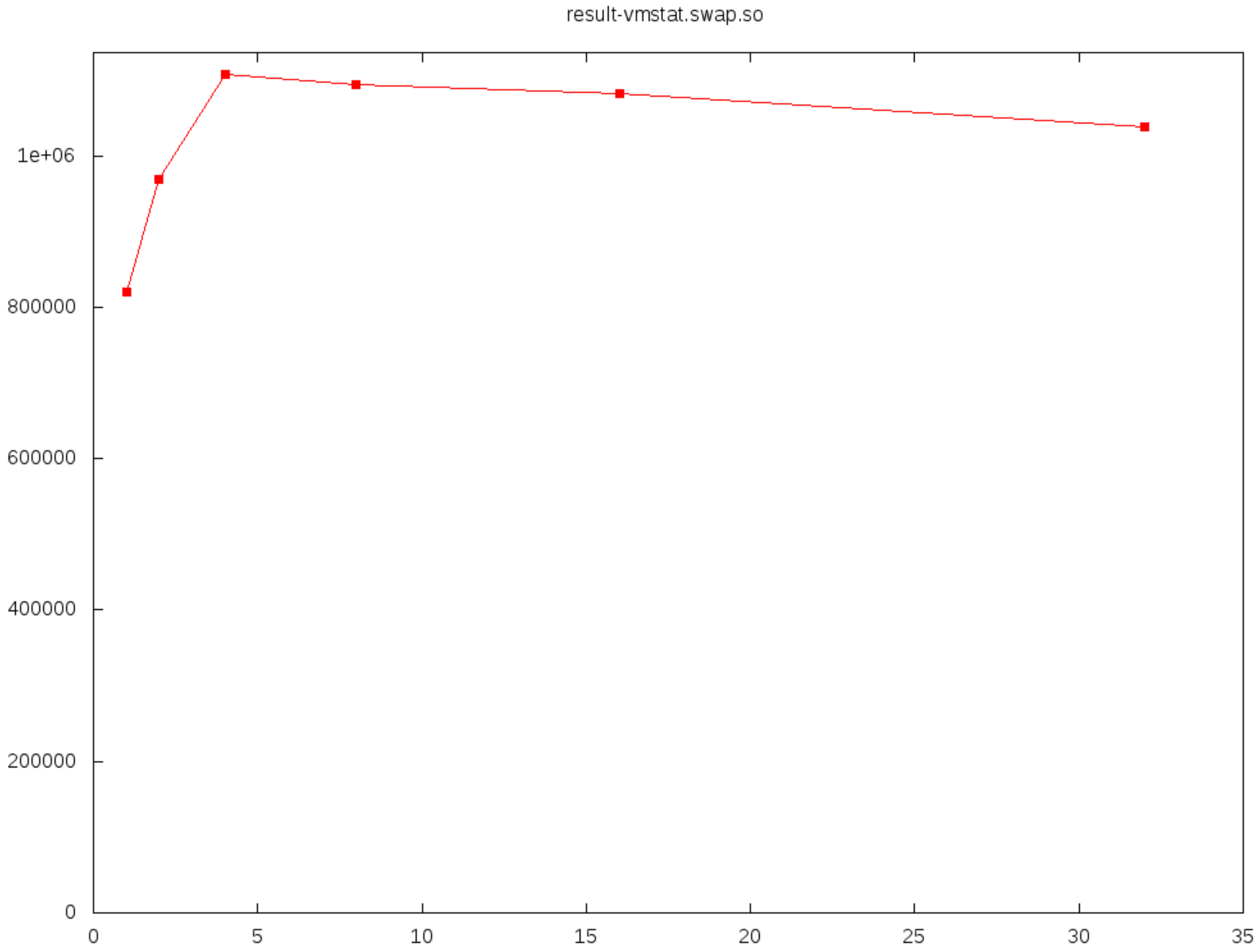
- Introduction to swap
- Swap implementation
- Swap IO performance optimization
- Other swap optimization
- Conclusion

Swap performance measure

- Swap device: RAM disk
  - Stress swap subsystem
- Test case: vm-scalability
  - Map large anonymous space, then write it sequentially or randomly
- Test machine
  - 2 sockets Xeon E5 v3 with 72 threads, 128G memory, 96G as RAM disk

Swap performance measure - Result 1

- Swap out throughput vs. process number



Swap performance measure - Result 2

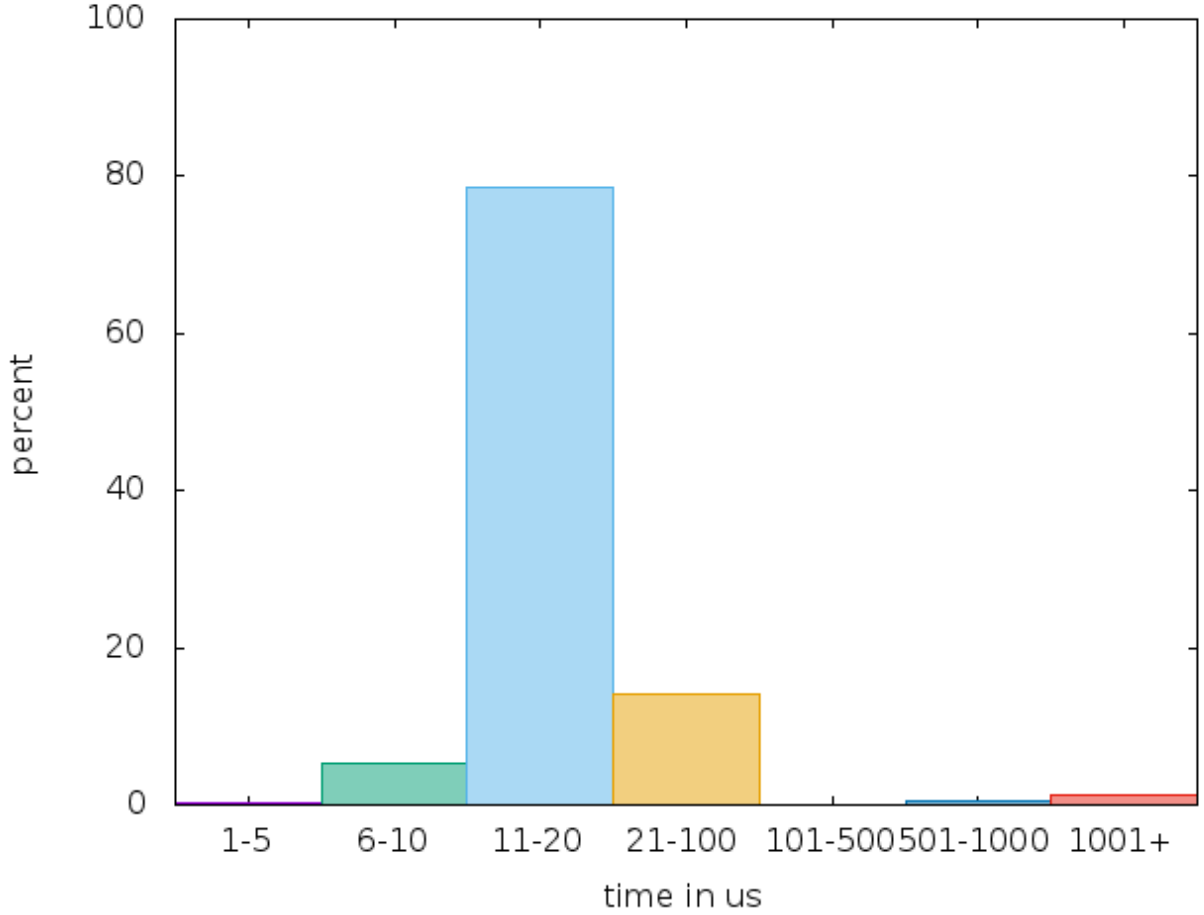
- perf profile result with call graph for 8 processes test

```
10.34% _raw_spin_lock_irq.__add_to_swap_cache.add_to_swap_cache.add_to_swap
10.24% _raw_spin_lock_irqsave.__remove_mapping.shrink_page_list
 9.91% _raw_spin_lock_irqsave.test_clear_page_writeback.end_page_writeback
 9.89% _raw_spin_lock_irqsave.__test_set_page_writeback.bdev_write_page
```

- > 40% CPU cyles spent for spinlock!

Swap performance measure - Result 3

Random swap in latency for 8 processes test



Swap performance issues

- Swap performance
  - swap out: cannot saturate storage device bandwidth
  - swap in: much higher than storage device latency
- Not scalable: heavy lock contention
  - swap cache lock
  - swap device lock

Optimize swap cache lock - Analysis

- swap out

```
select page to reclaim
allocate swap space
add page to swap cache
unmap page
*write page to swap device
remove page from swap cache
free page
```

- swap in

```
page fault
allocate page
add page to swap cache
read in page
map page
remove page from swap cache
free swap space
```

- *: update writeback tags!

Optimize swap cache lock - Solution

- Avoid touching writeback radix tree tags
    - Used by file writeback, not swap
    - Patch merged by v4.8-rc1
- Use fine grained lock
    - Split swap cache
        - One radix tree for every 64M swap space

Optimize swap device lock - Analysis

- swap out

```
select page to reclaim
allocate swap space
add page to swap cache
*unmap page
write page to swap device
*remove page from swap cache
free page
```

- swap in

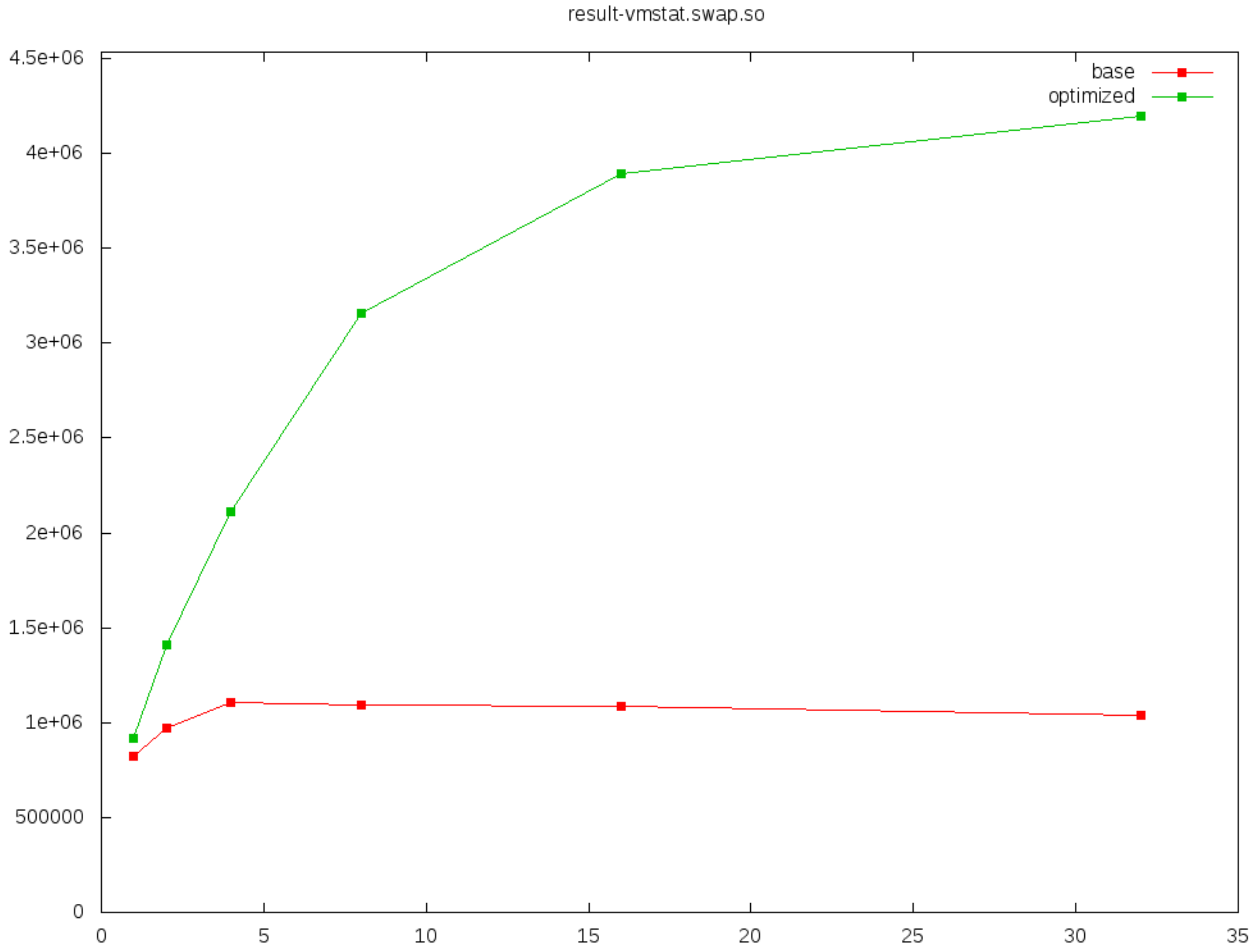
```
page fault
allocate page
*add page to swap cache
read in page
*map page
remove page from swap cache
free swap space
```

- *: update usage count for one swap entry at most times

Optimize swap device lock - Solution

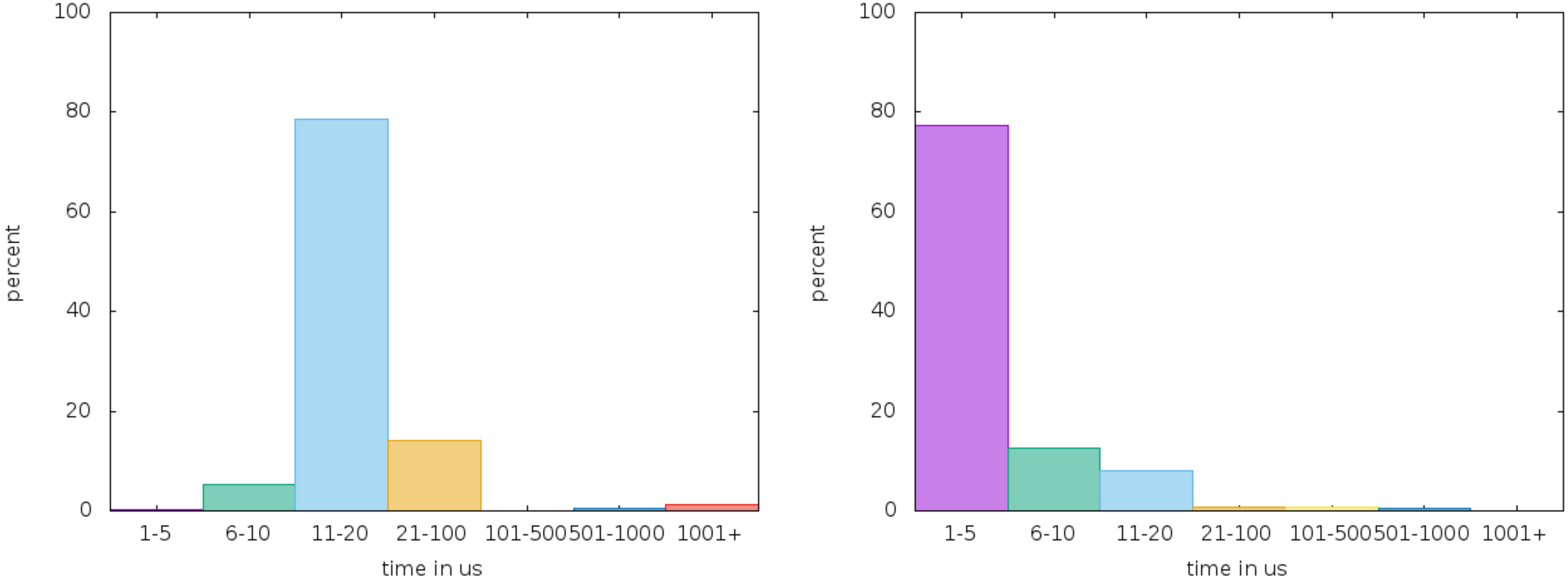
- Use fine grained lock
  - ○ Use lock on cluster to update usage count
- Batch swap space allocation and free
  - ○ Allocate 1 swap slot
    - □ Allocate <batch> swap slots
    - □ Put into per-CPU cache
  - ○ Free 1 swap slot
    - □ Put it into a per-CPU cache
    - □ Free <batch> swap slots

Effect of the optimization 1



- Swap out throughput vs. process number
- Base and Optimized

Effect of the optimization 2



base

optimized

Random swap in latency for 8 procqesses test

Optimize swap lock - Other solutions

- Batch locking
  - ○ From: Tim/Minchan/Mel

```
lock
remove 1st page from swap cache
unlock
...
lock
remove 2nd page from swap cache
unlock
...
```

=>

```
lock
remove <N> pages from swap cache
unlock
```

https://lkml.org/lkml/2016/5/3/798

- Improve radix tree updating parallelism
  - ○ Peter Z: fine grained lock on radix tree node

https://www.kernel.org/doc/ols/2007/ols2007v2-pages-311-318.pdf

- TSX spinlock

- Introduction to swap
- Swap implementation
- Swap IO performance optimization
- Other swap optimization
- Conclusion

Page reclaim: file or anonymous pages?

- From: Johannes Weiner
- Balance statically
  - swappiness: extend range to favor swap
- Balance intelligently
  - Refault and rotated

https://lwn.net/Articles/690079/

Optimize THP swap

- THP: Transparent Huge Page
  - Reduce TLB contention
  - Big memory size management
- Originally
  - Split THP before swapping out
  - Collapsed back to THP after swapping in
- Optimized
  - Swap out whole THP
  - Swap in whole THP

https://lwn.net/Articles/702159/

NUMA support

- Why
  - NVDIMM: per NUMA node
  - Reduce cross-node traffic
- Solution
  - Prefer to allocate swap space from NVDIMM in local node

https://patchwork.kernel.org/patch/9213531/

- Introduction to swap
- Swap implementation
- Swap IO performance optimization
- Other swap optimization
- Conclusion

Conclusion

- Storage hardware performance improvement makes swap more attractive
- Original swap subsystem implementation cannot take full advantage of latest storage hardware
- Resolving lock issue in swap subsystem improves scalability/performance of swap greatly

Thanks!