Reducing Memory Access Latency

Satoru Moriya <satoru.moriya.br@hitachi.com>

Linux Technology Center
Yokohama Research Lab.
Hitachi, Ltd

HITACHI
Inspire the Next

Contents

1. Introduction

HITACHI
Inspire the Next

1-1 Background

- Hitachi focuses its business on
  - Enterprise system
    - Stock exchange system, banking system, etc.
  - IT systems backing social infrastructure
    - Train control system, plant control system, etc.
  - Highly reliable cloud

- Some of them require very low latency
  - Latency order
    - Depend on each system (e.g. 1 msec/transaction)
  - Determinism
    - In those systems, there are time limits and we should not run past it at any time (target: soft realtime)
- Minimizing worst latency
  - Every process keeps to the time limit

1-2 Background – cont.

- Standard system
  - Optimized for best average (throughput)
- We have to tune/change systems to get required latency
  - Entire system
    - Hardware, firmware, OS, middleware & applications
  - OS
    - Cpu, memory, network, IRQ, etc.

This presentation focuses on latency in memory management area

1-3 Our Goal of mm Improvement

- Make the worst memory access latency less than 1 msec

#	Kind of pages		Sensitive for delay	detail
1	filebacked	unmapped	N	Pages are pagecache and so users think access delay due to I/O is acceptable because applications issue I/O explicitly when they access the data which is included in these pages.
2		mapped	Y	Pages are mapped to process's memory space – e.g. library page etc. Users don't accept access delay because application doesn't issue I/O explicitly when they access the data which is included in these pages.-(*)
3	anonymous		Y	Pages are allocated by applications. Users don't accept access delay because (*).

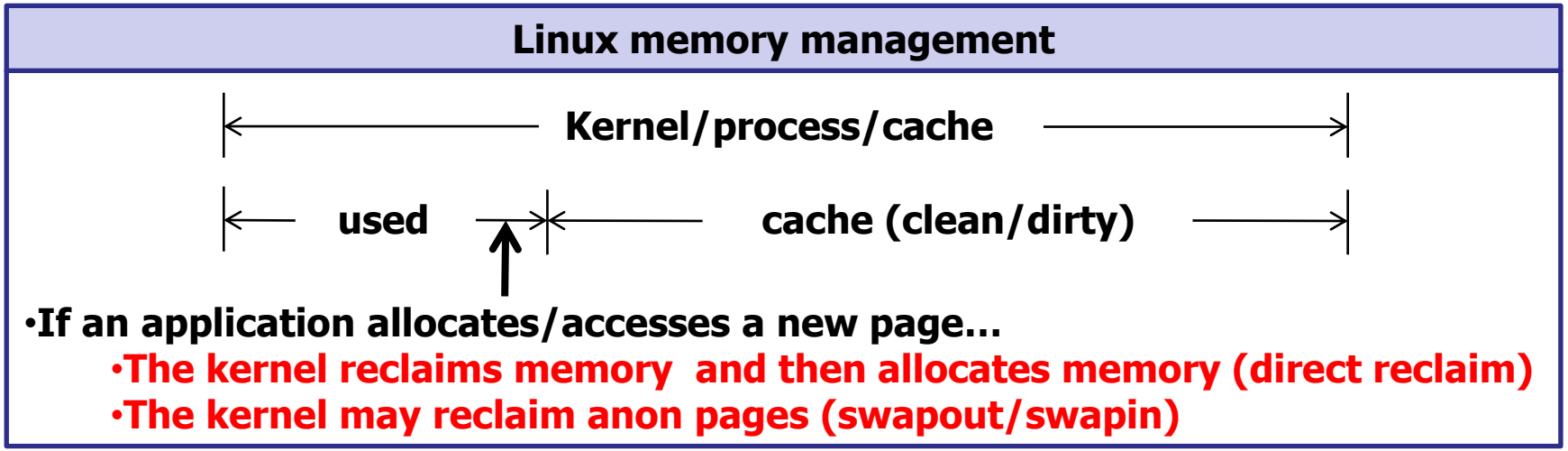
This talk focuses on access latency to "anonymous pages"

2. Issues

HITACHI
Inspire the Next

2-1 Issues

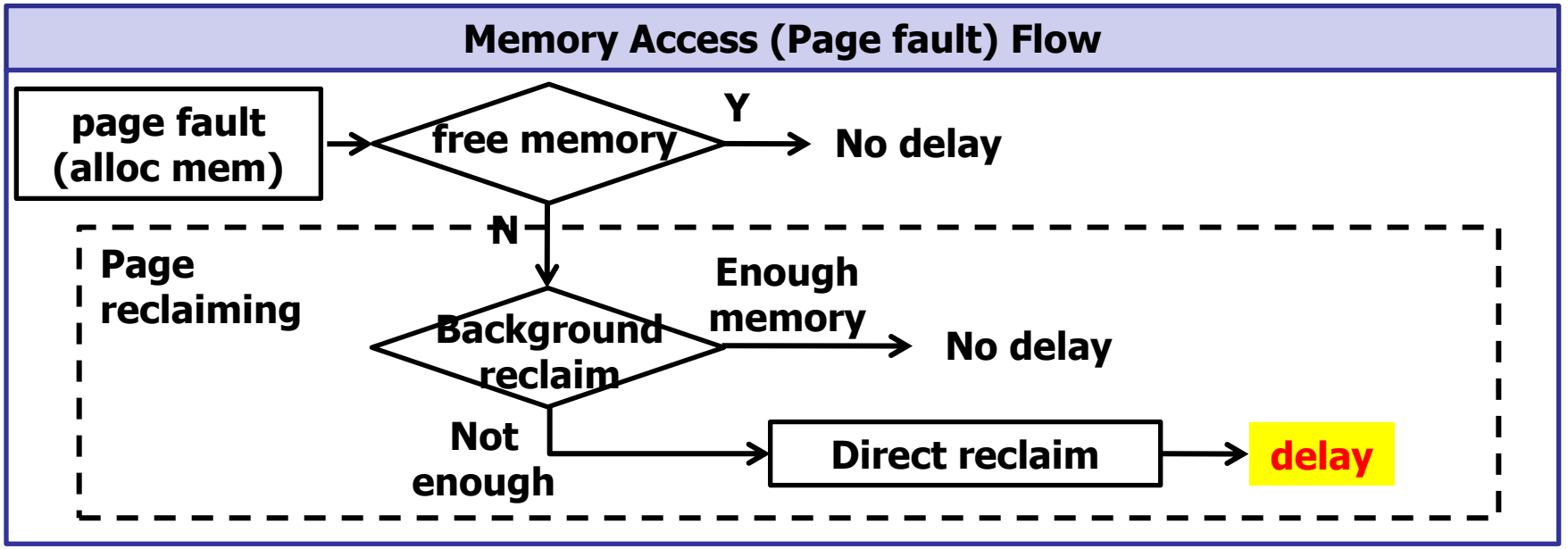
- Linux uses free memory for pagecache as much as possible



- 2 issues for memory access latency
  - Reclaim in page alloc path (direct reclaim)
    - It takes some time
    - May need I/O
  - swapout/swapin
    - Put out anon pages to disk
    - Need I/O to read data from disk at next access

2-2 Direct reclaim

- If Linux runs short of memory, it reclaims used pages and then allocate new pages
- There are 2 type of reclaim
  - Background reclaim (kswapd)
  - Foreground reclaim (direct reclaim)
    - Reclaim pages in process's context



2-3 swapout/swapin

- Anon pages may be reclaimed even if there are enough pagecache pages
- /proc/sys/vm/swappiness
  - We can configure how aggressive the kernel will swap anon pages
  - Anon pages can't avoid to be swapped out even if swappiness = 0
- When applications access the memory region which was swapped out, the kernel has to swap in necessary pages

How to get anon/filebacked scan ratio

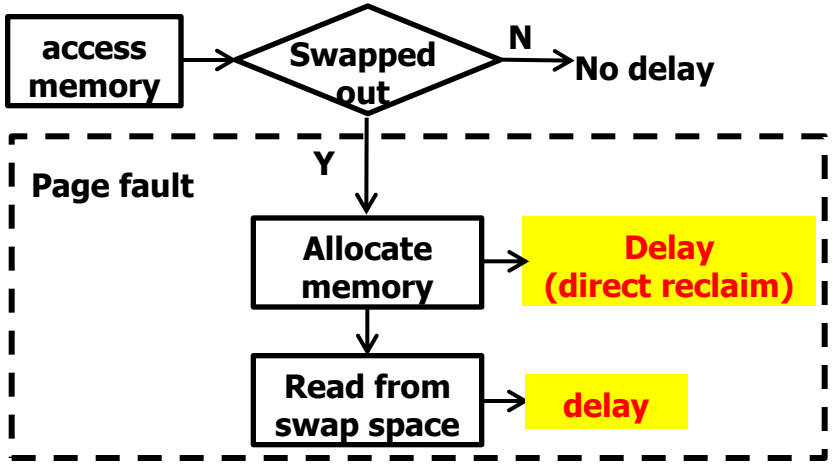
1. ap = (swappiness+1) * (scanned+1)/(rotated+1)
   fp = (200 – swappiness) * (scanned+1)/(rotated+1)

2. anon = ap / ap + fp + 1
   file = fp / ap + fp + 1

3. nr_scan_anon = (anon_lru_length >> priority) * anon
   nr_scan_file = (file_lru_length >> priority) * file

In some situation, nr_scan_anon is not zero and so anonymous pages may be reclaimed

Memory Access Flow



2-4 Current Solution (1)

- Preallocation + mlock(2)/mlockall(2)
  - Preallocate and call mlock/mlockall before starting critical sections
  - Ensure that necessary pages are resident in RAM
  - Avoid page allocation and reclaim during critical sections
    = Avoid delay

It needs to change application
Sometimes, we can't do it…..

2-5 Current Solution (2)

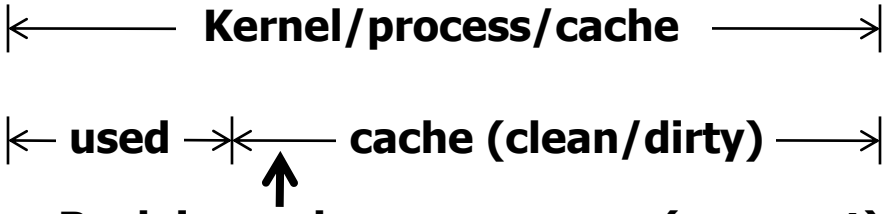
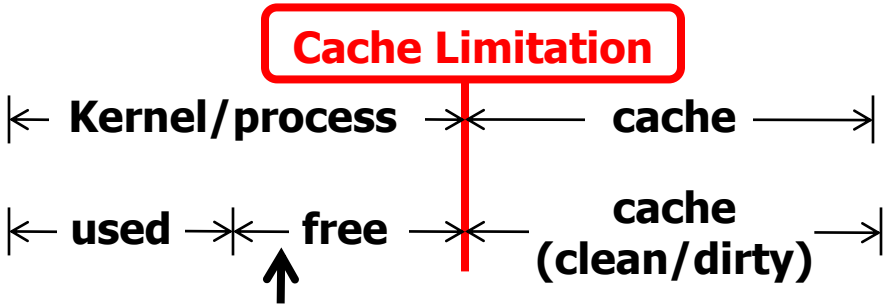
- cgroup (memcg)
  - Create a memcg
  - Set a max limit (memory.limit_in_bytes)
  - Put processes which consume pagecache (e.g. back up process etc.) into it

If the process which has latency critical sections increases pagecache, this solution may not work…. because current memcg doesn't have background reclaim

This solution doesn't help swapout/swapin issue

2-4 Solution in UNIX

- Commercial UNIX has pagecache limitation feature
  - It saves free memory by limiting the amount of cache

Without cache limitation	With cache limitation
	
•Reclaim cache or anonpage (swapout) and then allocate memory •Reclaiming memory may cause delay •Reclaiming anonpage may cause delay	•Reclaim cache only •Keep free memory and just allocate memory from it •Avoid delay

- Some enterprise users really want the feature because they use it in their current system
- This feature was proposed to the kernel community several times. But it has not been accepted yet.
  - 2007: Limit the size of the pagecache
    - http://lwn.net/Articles/218890/
  - 2011: Unmapped page cache control
    - https://lkml.org/lkml/2011/3/30/10

2-4 Solution in UNIX

- Commercial UNIX has pagecache limitation feature
  - It saves free memory by limiting the amount of cache

Without cache limitation	With cache limitation
•Reclaiming anonpage may cause delay	•Avoid delay

We need to take other approach…

- This feature was proposed to the kernel community several times. But it has not been accepted yet.
  - 2007: Needs/reasons were not discussed well.
    - http://kerneltrap.org/mailarchive/linux-kernel/2007/1/24/47350
  - 2010: Implement issue (Too big negative impact on fast path)
    - https://lkml.org/lkml/2011/3/30/10

3. Solution

HITACHI
Inspire the Next

3-1 Approach in Linux

- Issues
  - Direct reclaim
    - Reclaim pages in page alloc path
    - May need I/O
  - swapout/swapin
    - Put out anon pages to disk
    - Need I/O to read data from disk at next access

- Issuing I/O in memory access path causes huge latency

We need to avoid I/O in memory access path

3-2 Avoid I/O in direct reclaim

- Issue
  - Huge latency is caused when the kernel writebacks in direct reclaim
- Solution
  - Avoid writeback in direct reclalim
    = reclaim only clean pages in direct reclaim
  - The patch was proposed by Mel Gorman and merged into 3.2
    - mm: vmscan: do not writeback filesystem pages in direct reclaim
    - https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=ee72886d8ed5d9de3fa0ed3b99a7ca7702576a96

We can avoid I/O in direct reclaim now!!

3-3 Avoid swapout/swapin

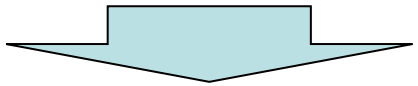
- Issue
  - Huge latency is caused when an application accesses the swapped out page
  - We can't avoid swapout even if swappiness == 0

- Solution
  - Change the behavior with swappiness == 0
    - With this value the kernel doesn't swapout any anon pages while it has enough filebacked pages
    - If we set cgroup swappiness to 0, we can avoid swap out completely for the processes in the cgroup
  - I proposed the patch and it was merged into 3.5
    - mm: avoid swapping out with swappiness == 0
    - https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=fe35004fbf9eaf67482b074a2e032abb9c89b1dd

We can avoid swapout with swappiness== 0!!

3-4 Status

- Issues
  - Direct reclaim
    - Reclaim pages in page alloc path
    - ~~May need I/O~~ **Fixed**
  - ~~swapout/swapin~~ **Fixed**
    - ~~Put out anon pages to disk~~
    - ~~Need I/O to read data from disk at next access~~



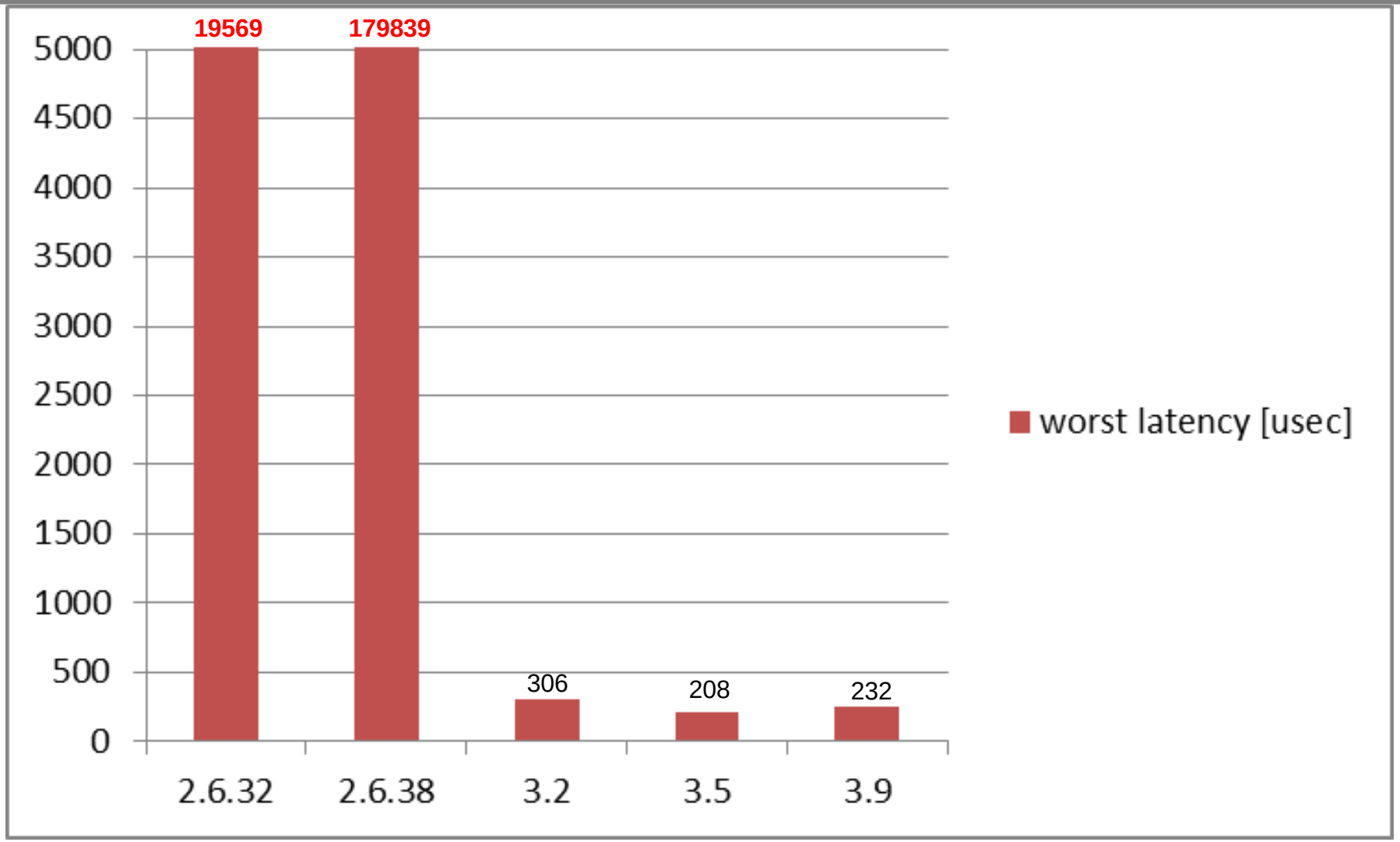
- In most cases, we can avoid latency issues
- But we can't avoid direct reclaim itself
- Do we really need to avoid direct reclaim??
  - How big does it impact ??

So…let's measure the latency after improvement

3-5 Measure memory access latency

- Hardware
  - CPU: 4
  - Mem: 8GB
- Software
  - RHEL6.2 + upstream kernel
  - Filesystem: system(ext4), data(ext3)
- Test
  - Measure memory access latency with heavy I/O
    - Foreground task
      - mapped_file_stream (modified from mmtest)
    - Background task
      - dd (Heavy I/O)

3-6 Result



Now…latency is improved significantly!

3-8 Still need page cache limitation?

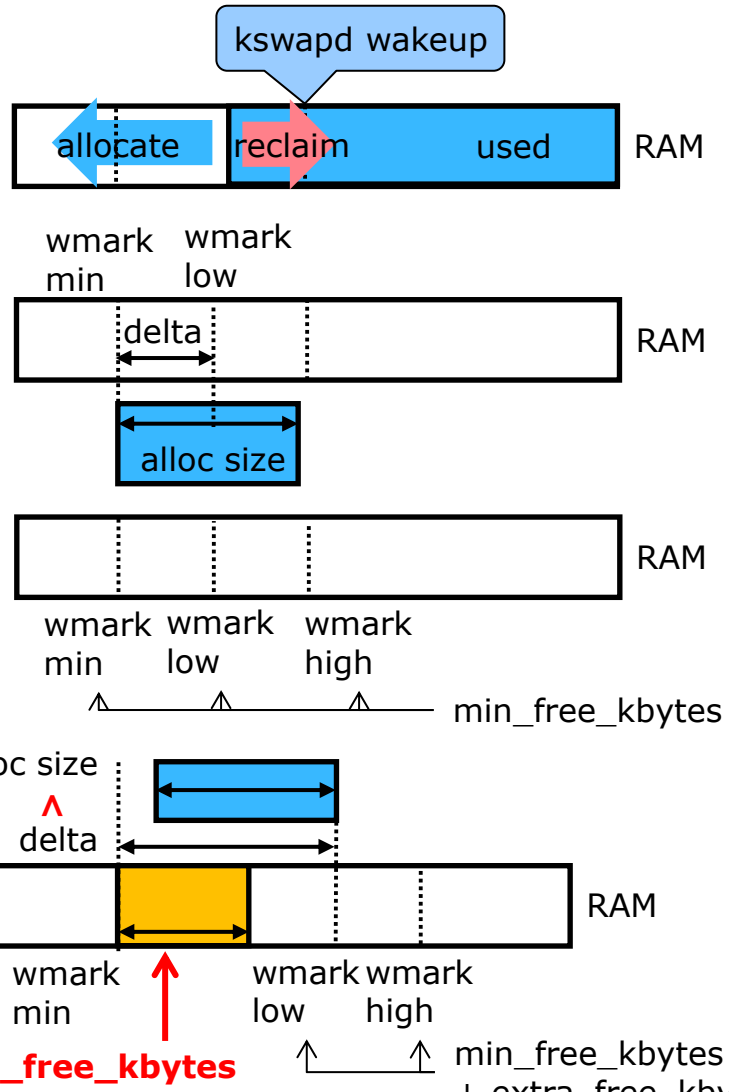
- Some times we hit a bug….
  - https://lkml.org/lkml/2013/2/11/570
- Basically, I agree with the approach community take that fix the root source of issue
- In enterprise area, we'd like to avoid bugs as much as possible in advance
- So…we'd like to have tunables like pagecache limitation
- Introducing pagecache limit is difficult…

Thinking about other approach

3-9 extra_free_kbytes

Linux Technology Center

- Why pagecache limitation? ⇒ Avoid direct reclaim!
- Issues
  - Direct reclaim will occur when…
    - **Allocation is faster than background reclaim**
    - **The amount of burst allocation is bigger than the delta between wmark_low and wmark_min**
  - All watermarks are set based on min_free_kbytes



- Solution

  "Add extra bytes between wmark_min and wmark_low"
  - Introduce new sysctl
    - /proc/sys/vm/extra_free_kbytes
  - Users can make the delta between wmark_min and wmark_low bigger than burst alloc size

⇒ We can avoid direct reclaim !!

3-10 Summary

- Issues
  - Direct reclaim
    - Reclaim in page allocation path
    - I/O
  - Swapout/swapin
    - I/O
- Solution
  - Avoid writeback in direct reclaim
  - Avoid swapout with swappiness == 0

⇨ Latency issues have gone away in most cases

- For users who really need pagecache limitation
  - extra_free_kbytes
  - You should evaluate current kernel with your workload
    - Linux may handle issues in your UNIX systems ;)

Linux Technology Center

4. Question and Discussion

HITACHI
Inspire the Next

Thank you

HITACHI
Inspire the Next

Legal Statements

- Linux is a registered trademark of Linus Torvalds.
- UNIX is a registered trademark of The Open Group.
- All other trademarks and copyrights are the property of their respective owners.