Kernel Samepage Merging (KSM) Usage at Meta and future Improvements

Stefan Roesch
November 2023

Agenda

- What is Kernel Samepage Merging (KSM)?
- How does it work?
- How can it be configured?
- How can it be monitored / traced?
- Using KSM for the instagram workload and required changes
- Evaluating if KSM is a fit for a new workload (metrics)
- Security concerns
- Current and future development
- Summary / Version breakdown of changes

What is Kernel Samepage Merging (KSM)?

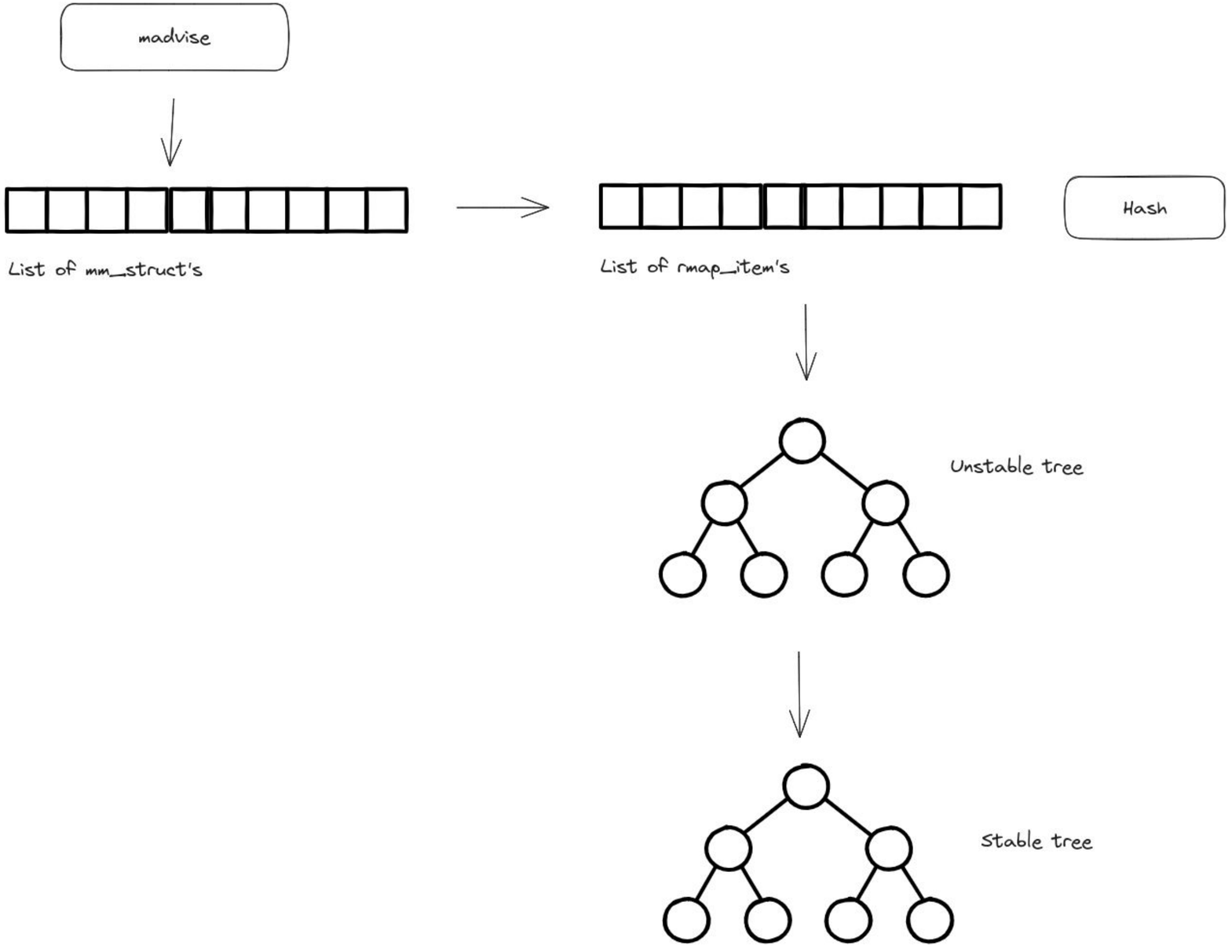
Kernel Samepage Merging

Is a memory de-duplicating scheme that finds duplicate anonymous memory pages and shares them in order to alleviate memory bottlenecks.

How does (KSM) work?

- ksmd kernel background thread scans all the anonymous pages whose VMA's have KSM enabled ("candidate pages")

- The scan operates in 3 major phases (simplified)
  - For each candidate page a so-called rmap_item is created
  - At Stage 1 it creates a hash for the page
  - At Stage 2 it adds the page (rmap_item) to the unstable tree
  - During stage 3: if duplicate pages are found,
    - It creates a KSM page and points the duplicate pages to this KSM page
    - The KSM pages are write-protected pages, which is automatically copied if a process wants to update the page (COW)
    - Further pages can later also point to this page


madvise
List of mm_struct's
List of rmap_item's
Hash
Unstable tree
Stable tree

Enabling applications / VMA’s to KSM

- There are two ways for an anonymous page to be included in the set of candidate pages:
  - madvise system call (this exists for a long time)
  - New option to the prctl system call
  - Not all memory regions can be used for KSM

Enabling KSM with madvise

- Madvise system call
  - To enable KSM for a memory region, the following system call can be used:
    - madvise(address, length, MADV_MERGEABLE)
  - The downside is that this assumes that you already know which memory region's will benefit the most from using KSM

  - Memory regions can also disable KSM with
    - madvise(address, length, MADV_UNMERGEABLE)

Enabling KSM with prctl (6.4)

- Prctl system call
  - The new option to the process control system call behaves differently: it enables KSM for all compatible VMA's in a process.
  - The setting is also inherited when a process is forked.

  - To enable KSM for a process, the following system call can be used:
    - prctl(PR_SET_MEMORY_MERGE, 1, 0, 0, 0)
  - To query if a process has KSM enabled (with the prctl system call), the following prctl system call can be used
    - prctl(PR_GET_MEMORY_MERGE, 0, 0, 0, 0)

How can KSM be configured

- Administration and monitoring is mostly done through the sysfs interface /sys/kernel/mm/ksm

- run
  - KSM is globally enabled with writing “1” to the run file
- pages_to_scan
  - determines how aggressively the ksmd background thread scans. The parameter determines how many pages are scanned per batch
- sleep_millisecs
  - determines the sleep time between batches
- Additional parameters for specific use cases like
  - use_zero_pages, max_page_sharing, merge_across_nodes, smart_scan

Monitoring KSM

- KSM monitoring information is exposed in
  - /sys/kernel/mm/ksm
  - /proc/<pid>/ksm_stat
  - /proc/<pid>/smaps and /proc/<pid>/smap_rollups (6.6)

Monitoring KSM

- /sys/kernel/mm/ksm
  - pages_scanned: Cumulative number of pages scanned (6.6)
  - pages_shared: Number of shared KSM pages
  - pages_sharing: Number of references to KSM pages
  - pages_skipped: Skipped by smart scan (6.7)
  - pages_unshared: Evaluated, but page is unique
  - pages_volatile: How many pages changed too fast

  - full_scans: Number of scans completed

Tracing KSM

- The trace events have been added for Kernel Version 6.4. Before version 6.4 no tracepoints existed.
- The tracepoints are defined in include/trace/events/ksm.h

List of tracepoints:

- ksm_start_scan
- ksm_stop_scan
- ksm_enter
- ksm_exit
- ksm_merge_one_page
- ksm_merge_with_ksm
- ksm_remove_ksm_page
- ksm_remove_rmap_item

KSM at Meta

- Instagram team was facing memory pressure and on older server machines also cpu pressure
- Workload is characterized by a controller and at least 32 worker processes. All of these worker processes initially load their interpreter in-memory, but also share a lot of other data structures that get loaded on demand

KSM seemed to be a good fit.

KSM at Meta

- At this point KSM only supported the madvise system call
- However we needed cgroup support: ideally enabling KSM through systemd when a cgroup gets started.
  - Basic idea
    - Enable KSM in the seed process and
    - Have the setting inherited when child processes are forked

- Added the new option to the prctl system call (6.4)
- Added a new configuration option to systemd that enables KSM for the cgroup.
  - The new option is called “MemoryKSM”
  - Systemd commit

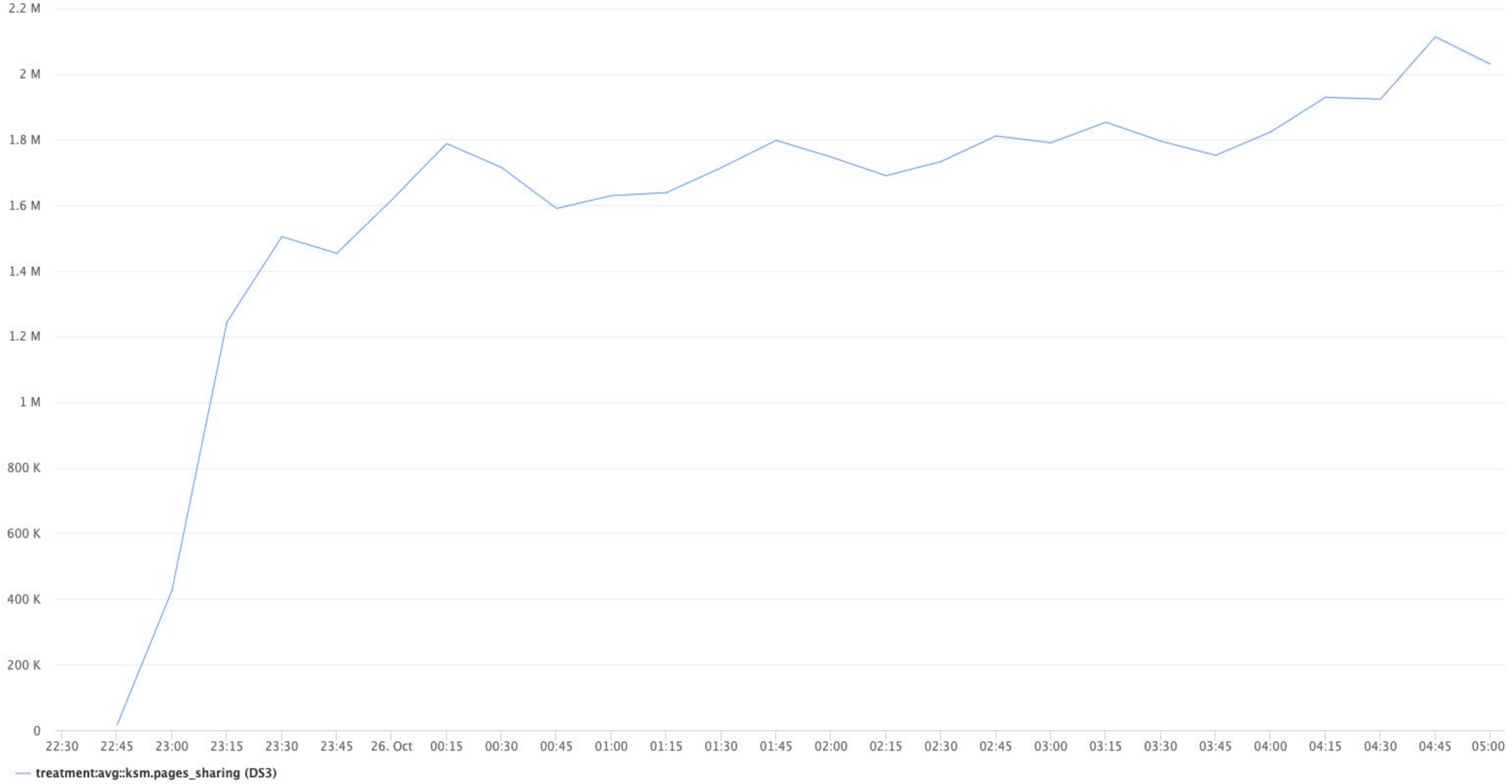
KSM at Meta

- Default setting of pages_to_scan (100) is insufficient for any real workloads.
- I run several experiments for the instagram workload a pages_to_scan value in the range of 4000 - 5000 seems to be a good compromise.

- Other workloads required an increase to 2000 - 3000.
  - Memory savings and
  - Scan time are good hints

Instagram workload metrics

Metric	Value
pages_shared	73,670
pages_sharing	2,110,000
pages_unshared	1,390,000
pages_volatile	7,070,000
Parameter	Value
pages_to_scan	var
sleep_millisecs	20


2.2 M
2 M
1.8 M
1.6 M
1.4 M
1.2 M
1 M
800 K
600 K
400 K
200 K
0
22:30
22:45
23:00
23:15
23:30
23:45
26. Oct
00:15
00:30
00:45
01:00
01:15
01:30
01:45
02:00
02:15
02:30
02:45
03:00
03:15
03:30
03:45
04:00
04:15
04:30
04:45
05:00
treatment:avg::ksm.pages_sharing (DS3)

Instagram workload metrics

Metric	Value
pages_shared	129,547
pages_sharing	5,119,146
pages_unshared	1,760,924
pages_volatile	10,761,341
Parameter	Value
pages_to_scan	4000
sleep_millisecs	20

Optimization - Smart Scan (6.7)

- A number of candidate pages are scanned over and over without being de-duplicated

Idea: keep some historic information and skip scanning these pages for the next scan or scans. The skip count is based on how often the pages have already been tried to be de-duplicated.

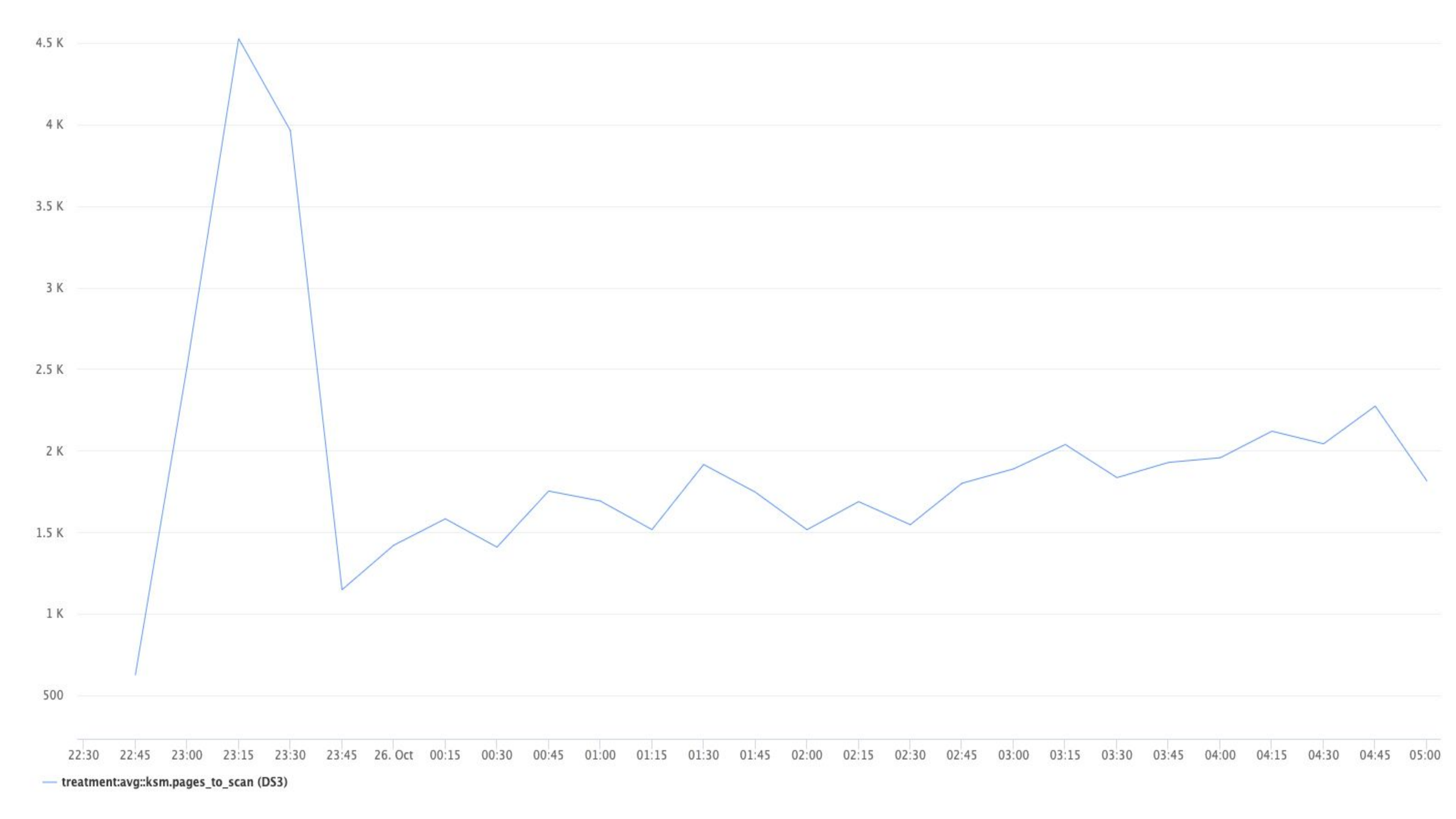
- The feature is enabled with the new parameter smart_scan.
- The default is on.
- With smart scan up to 10-20% less pages are scanned per scan.

Planned optimization: Auto-tune / Advisor

- It has been observed that during startup of an application more candidate pages need to be scanned than once “steady state” is reached.
  - For instagram during startup KSM needs to scan more than twice the number of candidate pages
  - Other workloads have shown similar behavior
- The proposal is to introduce an auto-tune component to automatically manage how aggressive ksmd scans the candidate pages:
  - Several parameters:
    - Target scan time
    - Min cpu usage (in percent)
    - Max cpu usage (in percent)

Planned optimization: Auto-tune / Advisor

- Results are very promising
  - During startup, it scans with a pages_to_scan value of 5000 to 6000
  - Once it reaches “steady state”, the scan rate is reduced to 2500 by the advisor, sometimes even less (this depends on the workload)
  - This achieves a CPU savings of 20-30% for the ksmd background thread
  - Configuration is more meaningful for the administrator: It is easier for him to configure KSM in terms of CPU usage and target scan time than with a per batch pages_to_scan parameter


4.5 K
4 K
3.5 K
3 K
2.5 K
2 K
1.5 K
1 K
500
22:30
22:45
23:00
23:15
23:30
23:45
26. Oct
00:15
00:30
00:45
01:00
01:15
01:30
01:45
02:00
02:15
02:30
02:45
03:00
03:15
03:30
03:45
04:00
04:15
04:30
04:45
05:00
treatment:avg::ksm.pages_to_scan (DS3)

How to evaluate new workloads

- Enable KSM for the process with the new prctl API
  - Change your application accordingly
  - Use the new systemd switch
  - Inject it into the executable with something like
    - LD_PRELOAD=/…/preload.so

```
#include <sys/prctl.h>
#include <sys/types.h>
#include <signal.h>

__attribute__((constructor)) static void on_load(void)
{
      prctl(PR_SET_MEMORY_MERGE,1, 0,0,0);
}
```

How to evaluate new workloads

- Run the application
- Overview how well KSM works
  - Query /sys/kernel/mm/ksm/general_profit
- Evaluate individual processes:
    - Query /proc/<pid>/smap_rollups
    - Query /proc/<pid>/ksm_stat
- Repeat for different values of the pages_to_scan parameter
  - Evaluation needs to run long enough
  - How aggressive to scan depends on the workload
  - Scan time gives some hints
  - Default value of the pages_to_scan parameter is not adequate

Warning: KSM is not suitable for all workloads

Optimize new workloads

- Evaluate /proc/<pid>/smaps
  - Search for “KSM:”
  - This shows which VMA’s benefit the most from KSM
- Remove prctl API call and replace it with madvise calls for the VMA’s that benefit the most

Hint: Smaller page sizes work better for KSM

Idea: Evaluate new workloads without enabling KSM

- Today KSM needs to be enabled to evaluate if they benefit from using KSM
- Having a test mode that reports potential sharing benefits might be beneficial without incurring the full costs of KSM

Potential approaches:

- Collecting / Exposing hashes of candidate pages
  - Cheaper solution, can be enabled a few times during different stages of the workload (startup, steady state)
- Maintaining unstable and stable tree, but don’t do any merging
  - More expensive (benefits of not merging pages needs to be measured)
  - More accurate

Limitations / Security

- KSM works best if the page size is smaller. The bigger the page size, the smaller the probability of sharing
- KSM can be abused to execute side channel attacks
  - If you control your workload this is less of a problem
  - There has been research on this subject. Two papers for reference
    - A memory deduplicate side channel attack to detect applications in co-resident virtual machines
    - Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector

Versions

- 6.1
  - Added /proc/<pid>/ksm_stat to report number of allocated rmap_items
- 6.4
  - Expose new general_profit metric /sys/kernel/mm/ksm/general_profit
  - Expose process_profit in /proc/<pid>/ksm_stat
  - Expose ksm_merging_pages in /proc/<pid>/ksm_stat
  - Add new prctl API to enable KSM per process
  - Introduced tracepoints for KSM operations

Versions

- 6.6
  - Count all zero pages
  - Add zero pages counter for each process
  - Include zeropages when calculating KSM profit
  - Add pages_scanned metric to /sys/kernel/mm/ksm
  - Add ksm stats to /proc/pid/smaps and /proc/pid/smaps_rollup

Credit

- David Hildenbrand
- Johannes Weiner
- Rik van Riel
- Tejun Hejo

Useful links

- KSM documentation
- LWN: /dev/ksm: dynamic memory sharing
- LWN: KSM tries again
- High-Level KSM overview, with a focus on administration
- Detailled break-down of KSM changes by Kernel Version