CS695 Assignment 4 Report

Virendra Kabra and Advaid Deepak
200050157 - 20d070006

Spring 2024

Contents

1 Overview

1.1 KSM

KSM (Kernel Samepage Merging) [1] is a memory-deduplication feature in Linux kernels. A common use-case for this is when we want to fit multiple VMs onto a single physical machine. For example, these VMs might be running the same OS or the same applications, and such data can be shared across them. KSM enables this by merging copies of the same page.

1.2 Kernel Implementation of KSM

We work with Linux Kernel v6.5 available with Ubuntu 22.04 LTS. A brief description of the KSM implementation [2] follows:

1. Data-structures
   - Two red-black trees are used for storing pages: a stable tree and an unstable tree. Tree nodes are ordered by page contents.
   - The stable tree holds pointers to all the merged pages (ksm pages). Each such page is write-protected. Two types of nodes are used: regular nodes that keep reverse mapping structures in a list, and chain nodes that hold lists of duplicate pages.
   - The unstable tree holds pointers to pages which have been found to be unchanged for a certain period of time. These pages are not write-protected, hence the term "unstable". This is similar to the hint entries from `[memmgmt]` paper [3].
2. Workflow
   - `ksm_do_scan`: The KSM daemon `ksmd` scans through all mergeable pages (e.g. `madvise`'d `MADV_MERGEABLE` and anonymous) in each iteration.
   - `cmp_and_merge_page`: The hash value (checksum) of this page is computed. If checksum is same as seen in the previous iteration, it is inserted into the unstable tree or merged with another page from unstable tree into the stable one.
   - This is different from the approach in `[memmgmt]` [3] where hash value of a page is compared with hashes of other pages. Here, if the page hash is constant across full-scans, its entire content is compared against pages from tree pages using `memcmp` [4].

2 Parameters and Metrics

2.1 Parameters

Our main script `orchestrate.sh` allows several command-line arguments, including the below parameters.

- Number of VMs. Each VM runs Debian12 with 512MB memory and 4GB disk. We setup the first VM manually and clone the rest using `virsh`.
- Scan-rate: Scanning rate of mergeable pages by the KSM daemon. This can be varied with `pages_to_scan` - the number of pages to scan before `ksmd` sleeps, and with `sleep_millisecs` - the daemon's sleep-time. Since both inversely affect the scan-rate, we list experiments with `pages_to_scan`.
- VM Workload: This is the workload that each VM runs. We experiment with no workload, and with variations of `stress-ng` [5]. For `stress-ng`, we disable swapping and start 2 workers per VM continuously calling mmap/-munmap and writing to the allocated memory, varying advise `mergeable` and `unmergeable`. Other options are also used, and are discussed with VM experiments 5.
- For experiments without VMs, a different set of parameters is varied. Details below in 4.

2.2 Metrics

All monitoring happens via the `monitor_ksm.sh` script. This is invoked from the main orchestrator script. We note that the monitoring is not CPU-intensive, as can be observed in the experiments. Average CPU usage remains almost the same even as we increase the monitoring frequency by 60 times (per-minute to per-second granularity). Some metrics of interest are listed below.

- CPU usage: Average CPU usage on the machine in the interval.
- `pages_shared`: Number of shared pages being used, i.e., count of distinct KSM pages. Same as number of regular nodes in the stable tree.
- `pages_sharing`: Number of duplicate pages reclaimed. This is disjoint from pages counted in `pages_shared`.
- `pages_unshared`: Number of nodes in the unstable tree, i.e., pages that haven't changed since last full-scan but are yet to be shared.
- General Profit: Overall measure of how effective KSM is. Computed as `pages_sharing * PAGE_SIZE - rmap_items * sizeof(rmap_item)`

These are plotted both against intervals and number of VMs.

When plotting against number of VMs, we take values from the last interval. This captures stabilized metrics.

In some cases, we also refer to per-process KSM stats. These are obtained from `/proc/<pid>/ksm_stats`.

3 File-Structure and Usage Instructions

Please refer to the project README for a description of files, along with detailed setup and usage instructions.

4 Experiments without VMs

These experiments are performed to understand the internal working of KSM.

In these experiments, we allocate two KSM-mergeable arrays and observe effects on the metrics. Steps: allocate `array1` and `array2` in 10-second intervals. `array[i]=i` for both arrays. 10-seconds after both arrays have been allocated, some part of `array1` is modified to see KSM working and effect on metrics.

Below, for all three experiments, we observe that `general_profit` is negative for initial intervals. This happens because no sharing has been done yet but rmap items are being created and unstable tree is being populated.

1. After sharing stabilizes, exactly one page is modified and hence unmerged. This is clearly seen in the logs.

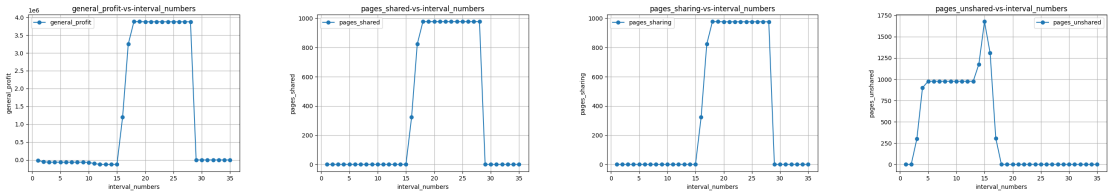

Figure 1: Changing `array1[0]` to -1

2. A dip in the middle happens due to unmerging of first half of arrays. But after that, half the pages of `array1` themselves can be merged as all values are -1, increasing the profit and shared pages.

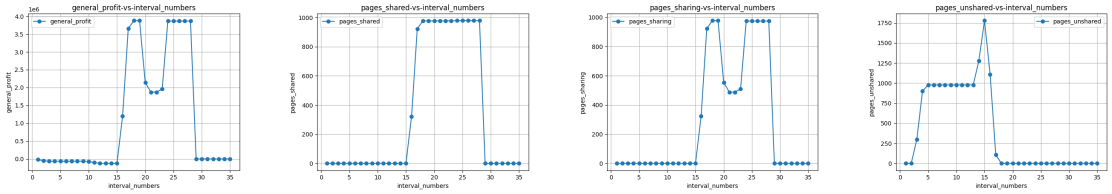

Figure 2: Changing half of first array to all -1s

3. Similar to previous experiment, a dip is seen in the middle. But now since modified values are distinct, they cannot be merged. Since the values are not changing after that, a smaller peak is visible at the end in `pages_unshared`.

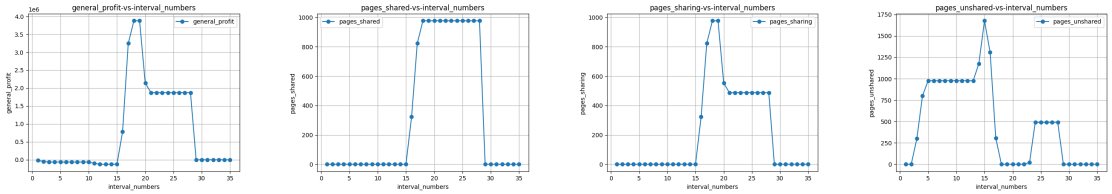

Figure 3: Changing half of first array by `array1[i]=-i`

5 Experiments with VMs

We performed experiments by varying the parameters discussed in 2.1. For each experiment, we monitored the metrics described in 2.2.

An important thing to note is that recorded metrics are for **system-wide** deduplication, as `ksmd` is running on the host machine. Hence, some noise is expected from merging and unsharing of other processes' pages.

5.1 Experiment 1

In this experiment, the scan rate is 1000 pages per 20 milliseconds. No workload was put on the VMs. We monitor for 30 intervals each lasting 60 sec.

5.1.1 Metric vs Interval

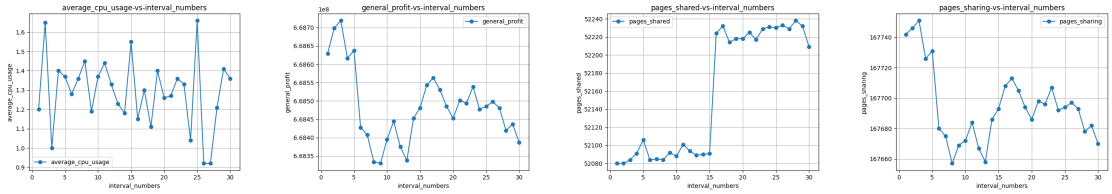

Figure 4

Three VMs were ran for the above plots, without workloads. We observe random variation in CPU usage. General profit and `pages_sharing` *slightly* decrease after an initial large value. This may be attributed to VMs writing to common pages after an initial phase. This may be due to noise as well - we do not observe this trend when experiments in the same setup with a larger number of VMs (5.8). The slight increment in `pages_shared` around 15 minutes may either be because of noise or because of sharing in 2 of the 3 VMs.

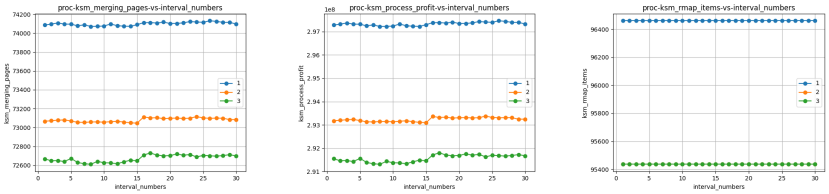

Figure 5

Each line in the above plots is for a VM process. `ksm_merging_pages` is the sum of `pages_sharing` and `pages_shared` [6]. It is higher for the first VM as it is more likely that de-duplication of later VMs happens *due to* memory regions of the first one. Further, we observe an increase in sharing for VMs 2 and 3 in the middle. Again, note that the differences are very small. We do not observe these in a later experiment 5.8.

5.1.2 Metric vs VM

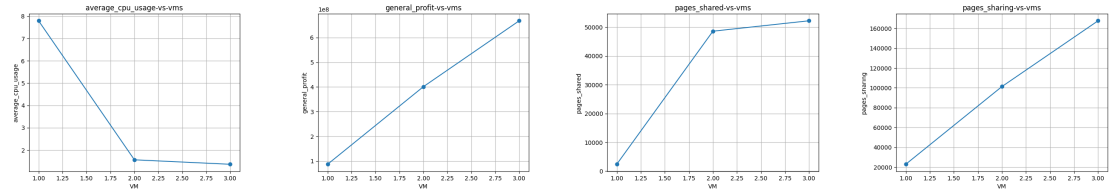

Figure 6

We observe that all KSM metrics increase with increase in number of VMs. This is expected because as number of VMs increase, there is that much more opportunity for page-sharing. This observation is common for all experiments.

5.2 Experiment 2

This experiment is similar to 5.1, with the only variation being in the scan rate: we increased it to 10000 pages per 20 milliseconds.

5.2.1 Metric vs Interval

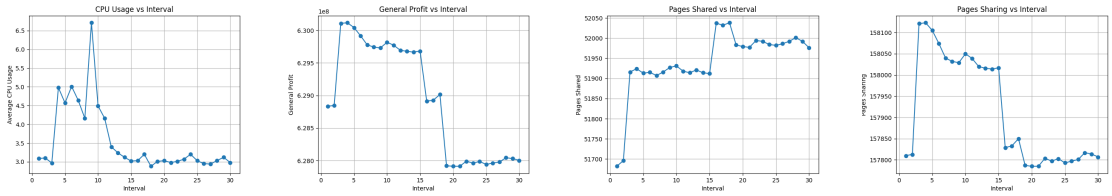

Figure 7

We observe that, owing to the increased scan rate, sharing happens quicker compared to the previous experiment. Also, an increased CPU usage is observed. This may be attributed to the KSM daemon. However, the profit and sharing values are almost same as before. This is because the earlier scan-rate is sufficiently large to scan through mergeable pages.

5.2.2 Metric vs VM

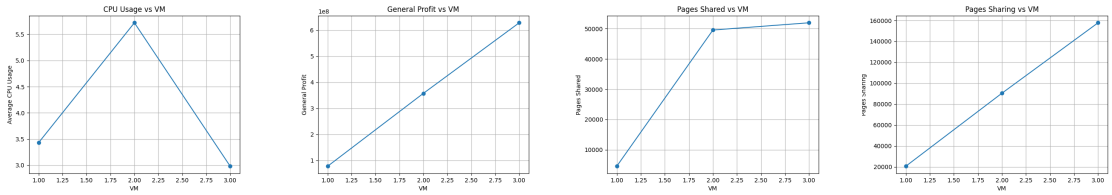

Figure 8

Similar observations as in the previous experiment follow.

5.3 Experiment 3

This experiment is similar to 5.1 (scan-rate: 1000 pages per 20 ms) except there is a workload in the VMs. This workload is created using stress-ng. Details of this workload were discussed above in 2.1. Swapping is disabled in the VMs. We run stress-ng for only the first half of the experiment to contrast metrics with/without workloads. Further, the advise for stress-ng is `unmergeable`.

5.3.1 Metric vs Interval

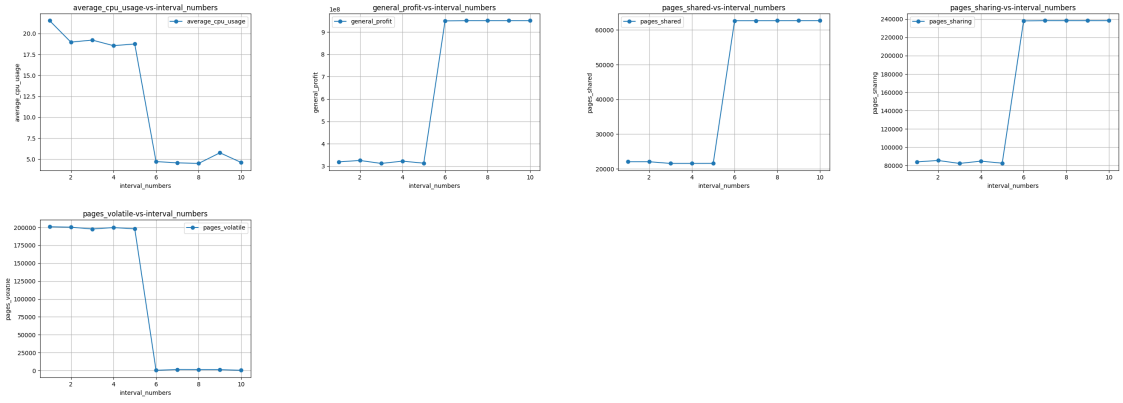

Figure 9

Since we run stress-ng for only first half of the experiment, the CPU usage is higher in the first half. Sharing happens more in the second half when stress-ng is turned off as it repeatedly mmaps/unmaps and writes to certain pages reducing the number of pages which can be shared. Trend in volatile pages is characteristic of the workload. This pattern is common to all subsequent experiments where this workload is used.

5.3.2 Metric vs VM

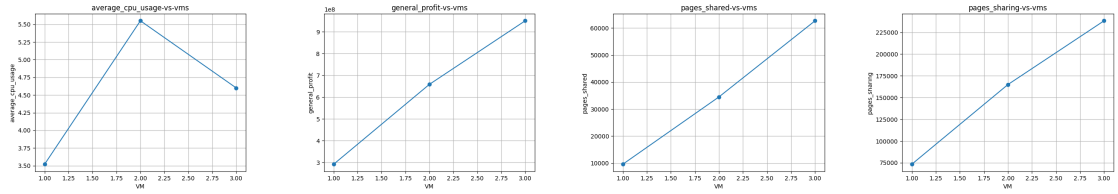

Figure 10

5.4 Experiment 4

This experiment is similar to 5.3 except we scan 10000 pages in each 20 ms interval.

5.4.1 Metric vs Interval

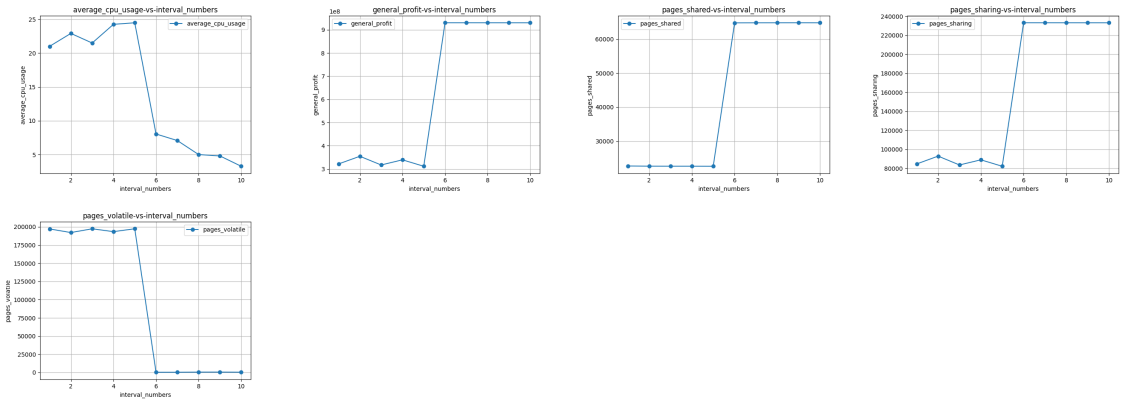

Figure 11

5.4.2 Metric vs VM

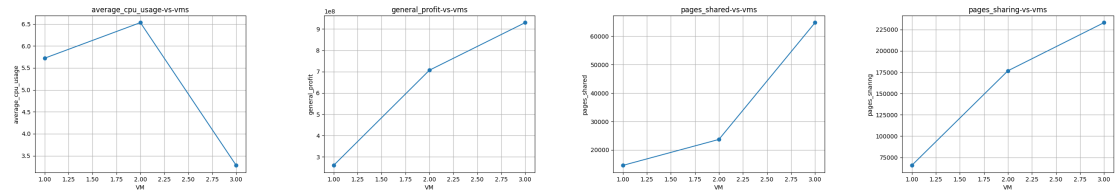

Figure 12

Similar observations follow as in 5.3. Higher CPU usage may be attributed to the increased scan rate.

5.5 Experiment 5

This experiment is similar to 5.3 (scan-rate: 1000 pages per 20 ms) except that the advise for stress-ng is `mergeable`. This was `unmergeable` in 5.3.

5.5.1 Metric vs Interval

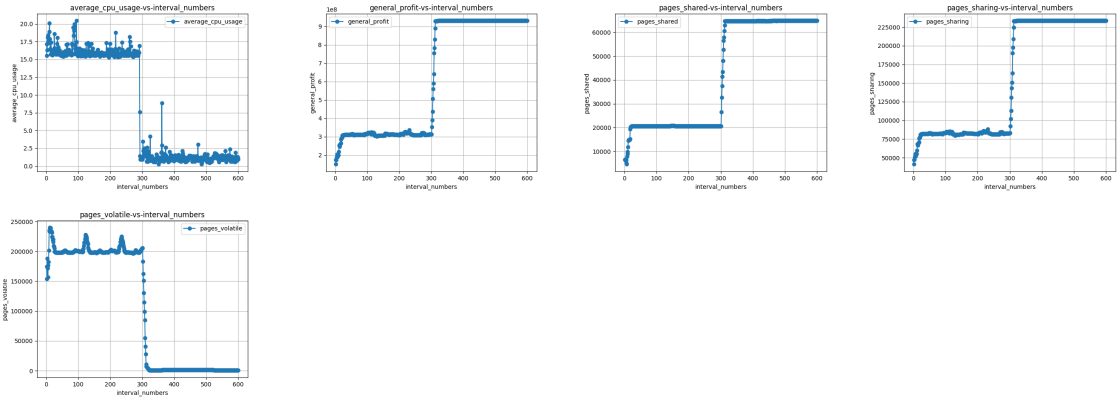

Figure 13

We observe that the sharing does not improve even though the workload advise was `mergeable`. The number of volatile pages too is same as seen in the last two experiments. This is due to the workload nature: it repeatedly mmaps/unmaps pages with 0 wait. So we expect no significant deviations from the experiments where advise was `unmergeable`.

5.5.2 Metric vs VM

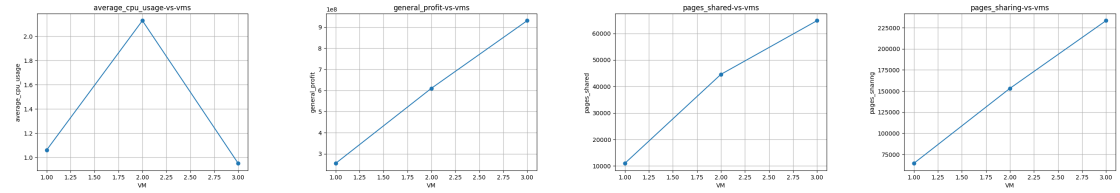

Figure 14

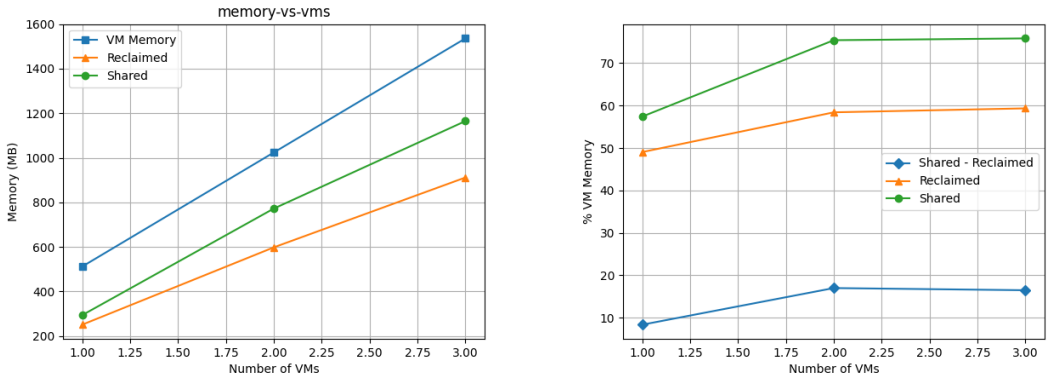

Figure 15

The pattern observed is similar to that in `memmgmt` [3], where VM memory, shared memory, and reclaimed memory increase as the number of VMs increase. The sharing percentage tends to flatten after 2 VMs, as seen clearly in 5.8.

5.6 Experiment 6

This experiment is similar to 5.5. Since we could not get the benefits of advising `mergeable`, we now also waited for 10 seconds before unmapping the mmap'ed memory regions. This was by adding `--vm-hang 10` in the VM workload.

5.6.1 Metric vs Interval

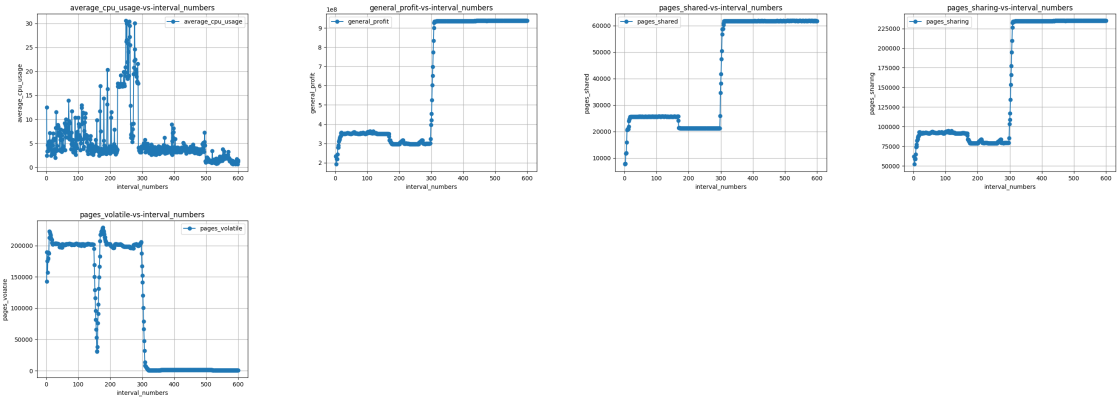

Figure 16

We observe an increase in the profit and shared pages in the first half (that is, when the workload runs). However, it is not huge, which might be attributed to repeated writing by the workload. We did not find options for limiting the writing-rate in stress-ng.

5.6.2 Metric vs VM

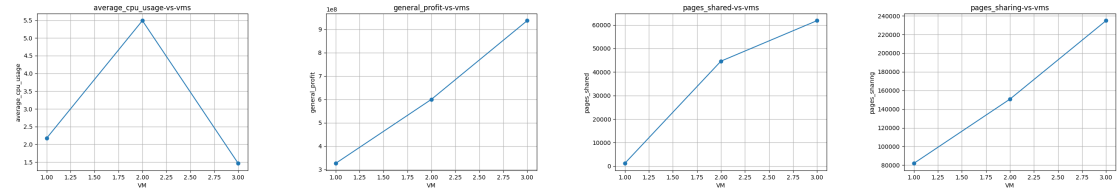

Figure 17

5.7 Experiment 7

This experiment is similar to 5.5, except we scan 10000 pages per 20 ms.

5.7.1 Metric vs Interval

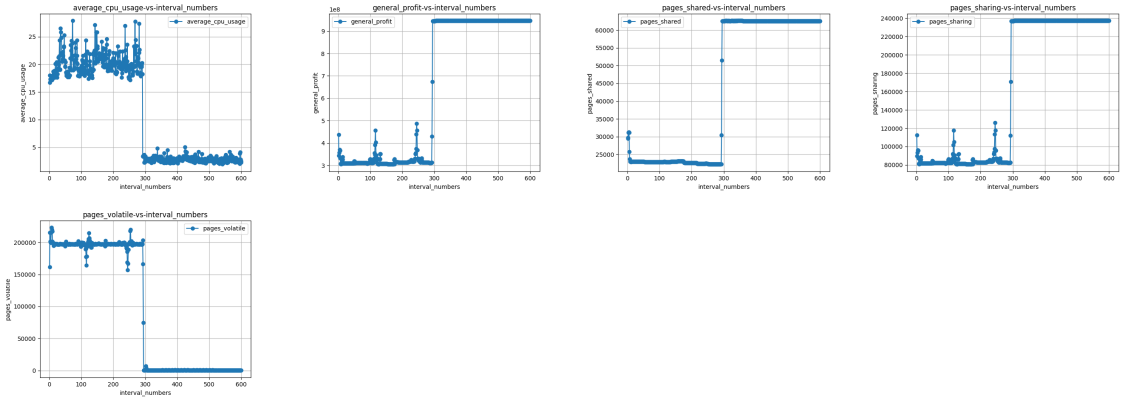

Figure 18

5.7.2 Metric vs VM

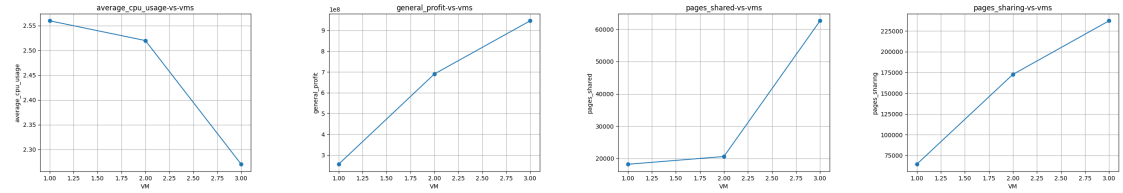

Figure 19

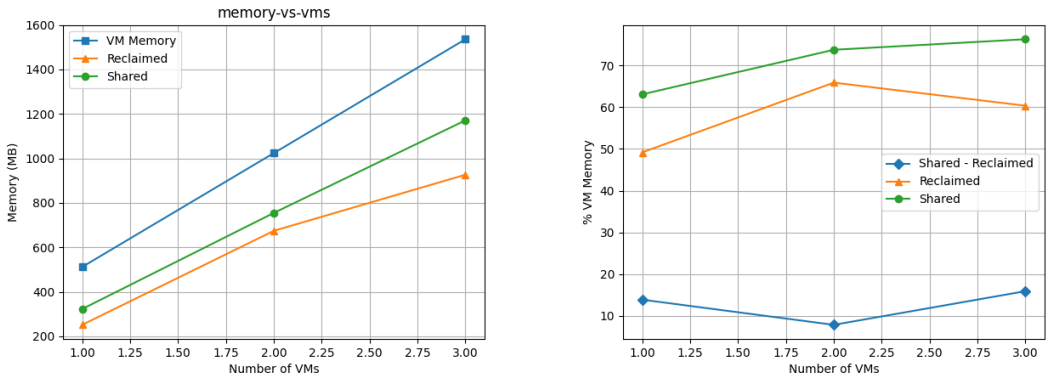

Figure 20

Similar observations follow as in 5.5.

5.8 Experiment 8

This experiment is similar to 5.1, except we run 10 VMs. This is done in an attempt to reproduce plots from the `memmgmt` paper [3].

5.8.1 Metric vs Interval

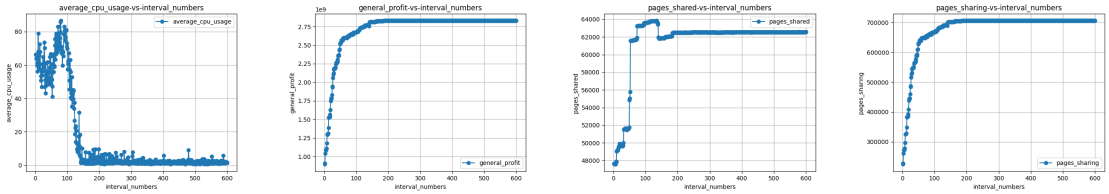

Figure 21

5.8.2 Metric vs VM

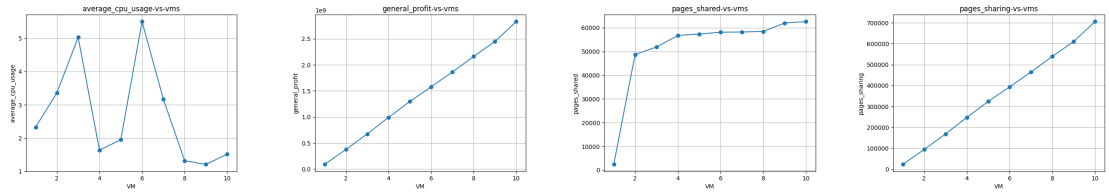

Figure 22

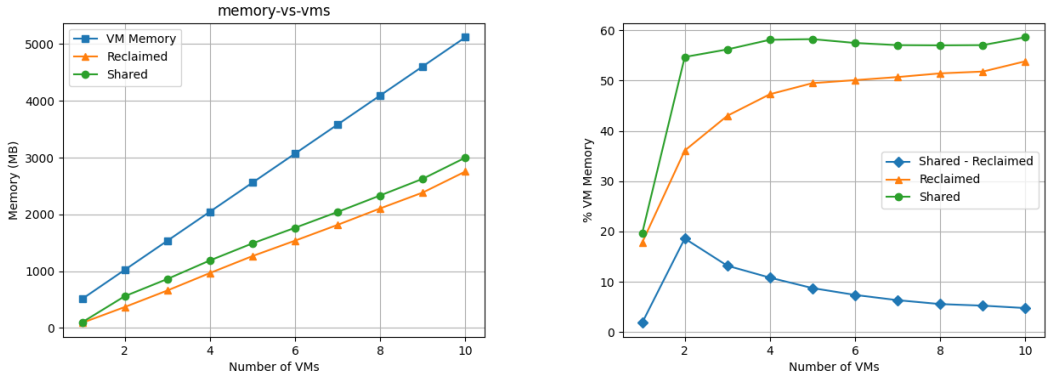

Figure 23

We observe that the percentage of shared and reclaimed pages saturate to around 58% and 53% respectively.

6 Conclusions

We created a framework and performed experiments using it to study the working of KSM. We experimented with parameters such as scan-rate and workloads. For all experiments, we recorded various metrics including CPU usage and several metrics for sharing. The observations of relevant sharing metrics are in accordance with the results obtained in `[memmgmt]` paper [3] even though some implementation differences exist.

Some suggestions based on our experiments and understanding of the implementation:

- A policy may be used for the KSM daemon. For example, large scan-rates like 10000 pages per 20 ms have a visible effect on the CPU load, and such rates may only be needed in short bursts. Prediction or detection of such periods would be beneficial both for memory de-duplication and for CPU usage.
- Only potentially mergeable pages should be madvise'd as such. We saw in one of the experiments that the workload marked pages mergeable but the workload design was such that intra/inter-VM sharing would be not possible due to continuous writing (5.5 and 5.6). This leads to a waste of CPU resources.
- Instead of comparing the entire page, storing and comparing hashes may be less expensive with certain workloads. We note that this does conflict with the concept of unstable tree.

References

[1] `https://docs.kernel.org/admin-guide/mm/ksm.html`

[2] `https://elixir.bootlin.com/linux/v6.5/source/mm/ksm.c`

[3] `https://www.cse.iitb.ac.in/~puru/courses/spring2023-24/downloads/memmgmt-annotated.pdf`

[4] `https://elixir.bootlin.com/linux/v6.5/source/mm/ksm.c#L1698`

[5] `https://github.com/ColinIanKing/stress-ng/`

[6] `https://elixir.bootlin.com/linux/v6.5/source/mm/ksm.c#L2090`