All page sharing is equal, but some sharing is more equal than others

Benchmarking of Kernel Samepage Merging and an Adaptive Page Sharing Technique

Shashank Rachamalla, Debadatta Mishra and Purushottam Kulkarni
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
e-mail: \{shashankr, deba, puru\}@cse.iitb.ac.in

Abstract
Content based memory sharing in virtualized environments has been proven to be a useful memory management technique. As part of this work, we are interested in studying the trade-off between extent of sharing opportunities (compared to actual sharing potential) that can be identified and the overheads for doing the same. Our work is based on Kernel Virtual Machine (KVM) in Linux which in turn uses Kernel Samepage Merging (KSM) to identify and exploit sharing opportunities. We instrument the Linux kernel and KSM to log sharing and copy-on-write related parameters for accurate estimation of useful and effective KSM sharing extent. For several combinations of workloads, exhibiting different memory usage characteristics, we benchmark KSM performance (achieved savings vs. total possible savings) and CPU overheads for different KSM configurations. The benchmarking results are used to develop an adaptive scheme to dynamically reconfigure KSM configuration parameters (scan rate and pages per scan), to maximize sharing opportunities at minimal overheads. We evaluate the adaptive technique and compare it with default settings of KSM to show its efficacy. Based on our experiments we show that the adaptive sharing technique correctly adapts scan rate of KSM based on sharing potential. With our setup, the adaptive sharing technique yields a factor of 10 improvement in memory savings at negligible increase in CPU utilization.

1 Introduction
Virtualization-based provisioning is a growing trend for hosting services and applications within data-centers. The major factor towards this increasing interest has been the on-demand and elastic nature of resource provisioning and a pay-on-use model for customers [1]. On the other hand, data-center providers benefit from multiplexing virtual machines by sharing resources and increasing utilization levels. Intelligent resource management techniques [2, 3, 4] employing dynamic resource allocation and migration techniques can efficiently multiplex resources across several consolidated virtual machines. The focus of this work is related to multiplexing memory across consolidated virtual machines (on a physical machine).

A commonly used technique by hypervisors to support memory over-commit is demand paging [8]—pages are swapped back and forth to alleviate memory requirements of virtual machines. While this technique enables over-commit, it may lead to frequent disk accesses during high aggregate memory pressure. An improvement over vanilla demand paging is to use the memory ballooning technique [8]. A balloon driver within each virtual machine can be inflated to force the machine to relinquish pages according to it's local memory management policy. A memory manager (in the host machine) coordinates overall memory requirements by inflating and deflating the balloon and minimizes the use and impact of demand paging on applications within the virtual machines.

Another technique to efficiently enable memory over-commit is to exploit redundancy of memory content, i.e., share pages with the same content [15, 14]. If several pages with the same content are shared—all pages with the same content refer to the same physical page, the effective memory footprint of a virtual machine can be reduced. A combination of memory sharing, ballooning and demand paging techniques are candidates for providing an efficient memory over-commit solution.

Many hypervisors today, both open source and commercial, employ content based page sharing (CBPS) techniques [15, 8]. The sharing techniques used by hypervisors can broadly be classified into two categories—in-band sharing and out-of-band sharing. Same page detection and merging in the I/O path (mostly disk access path) is an in-band technique as proposed in [12]. As opposed to this, a technique that periodically scans memory
to identify and merge shareable pages is an out-of-band technique [5, 8]. In the out-of-band approach, memory pages are scanned and compared with other pages to check for content equality. If such a page is found, a single copy of the page is shared and the other page is freed, thereby reducing the total memory footprint of resident VMs. The single copy is marked copy-on-write (CoW) and all writes to the shared page are handled by either returning exclusive copies to all virtual machines or rebuilding the shared state for that page. In this paper, we focus on Kernel Samepage Merging (KSM), an out-of-band page sharing technique employed by the KVM hypervisor.

Out-of-band sharing techniques can potentially take advantage of the complete system memory to identify identical memory pages, as opposed to in-band techniques which are limited to finding duplicates from I/O buffers and page caches. However, maximizing this benefit depends on parameters of the scanning process itself. Normally, out-of-band techniques oscillate between a sleep period and a scan period, e.g., with KSM, number of pages to scan for a time interval can be specified. As a result, the memory scanning rate directly affects the efficiency of exploiting sharing opportunities. With a “lower” scan rate, the likelihood of missing sharing opportunities that do not last for long, e.g., that do not last the time required to scan all pages, will be higher. On the other hand, an aggressive scan rate will capture most sharing opportunities at the cost of additional CPU utilization. Another side effect is the increased number of CoW breaks (and associated overheads) due to sharing opportunities of small durations.

With regards to the out-of-band sharing technique, and more specifically with KSM, we are interested in studying the following issues.

- Memory usage patterns of virtual machines (applications within virtual machines) are dynamic and exhibit different temporal characteristics as regards sharing potential. As levels of consolidation (number of virtual machines on a host) vary, so does the sharing potential and total number of potentially shareable pages. Thus, no single setting of the KSM parameters will be able to capture shareable pages across different memory usage scenarios. Further, for a given memory usage pattern and a given KSM configuration how much of the sharing potential is identified by KSM needs thorough benchmarking. For example, for a setting of KSM parameters, is the scan rate so low that KSM can only capture 50% of shareable pages in the best case? Our aim is benchmark KSM—it’s benefits (ability to capture sharing opportunities) and overheads, by exploring its configuration parameters and with different memory usage patterns.

- We explore the notion of usable sharing—sharing that can be positively exploited by sharing-based memory management techniques. For example, consider a ballooning based memory management scenario and KSM reporting 90% pages across virtual machines as shared. Further, assume that more than half these pages are shared for less than a minute. Should the balloon-based memory manager rely on the 90% sharing statistic or try to identify useful sharing opportunities (those that last longer)? A side effect of several pages being shared and modified very soon implies that the pages reused due to sharing need to reclaimed quickly as well. Unless this behavior is explicitly accounted and planned for by the memory manager, the sharing statistics will lead to incorrect memory over-commit levels and high demand paging activity. As part of this work, we use the notion of usable sharing, dependent on the sharing duration, and inferences from the benchmarking exercise to develop an adaptive technique to dynamically configure KSM settings. The adaptive techniques aims to maximize usable sharing at the least resource utilization cost.

2 Background and Tools

2.1 Kernel Samepage Merging

Kernel Samepage Merging (KSM) is a kernel thread which performs content based page sharing. Any user process can register itself to KSM by specifying a range of memory pages that KSM can scan and merge. The KVM hypervisor registers all memory pages of its virtual machines with KSM. This allows KSM to exploit intra-VM and inter-VM page sharing opportunities across all virtual machines on a KVM host.

KSM uses two red-black trees, stable & unstable, to perform page sharing. The nodes of these trees are indexed based on the contents of memory pages they point to. Stable tree nodes point to shared pages while unstable tree nodes point to pages which are good candidates for sharing but not yet shared. Both trees are initially empty. Figure 1 explains the working of KSM through a flow chart. As long as the stable tree is empty, scanned pages are searched for a match in unstable tree. If there is no match in the unstable tree then a page is inserted into the unstable tree. As the number of pages in unstable tree increase, the probability of finding a matching page increases. Upon a match, a copy of the matching page is created, marked copy-on-write and inserted into the stable tree. The page in the unstable tree and the page scanned are then merged with the stable tree page and both are freed. The merge operation involves modifying page table entries of the page in unstable tree and page
scanned to point to the page in the stable tree. KSM computes and persists a checksum for each scanned page. It does not search the unstable tree for a matching page if the newly computed checksum of scanned page differs from its previous checksum. This heuristic is used by KSM to avoid sharing of volatile pages—pages that are modified often. However, this check is not applicable while searching the stable tree for a matching page. A matching page found in the stable tree is already shared and hence not volatile. Therefore, scanned page is directly merged with stable tree page and freed.

The rate at which KSM scans pages can be controlled by setting sleep time of the KSM thread between scans and number of pages considered for sharing per scan.

2.2 Page Sharing Overhead

Page sharing reduces the memory footprint of individual virtual machines and hence also the aggregate memory requirement. This increases the probability of higher consolidation levels—number of virtual machines per physical host, by allowing higher levels of overcommitment. Memory overcommitment has to carefully deal with increasing demand for memory from virtual machines as the extent of sharing can vary dynamically. Page sharing presents an additional problem to consider when page content changes frequently across shared pages resulting in significant overhead. In the context of KSM, page sharing overheads can be broadly divided into: (i) compare & merge overhead, (ii) MMU notifications overhead and (iii) CoW break overhead.

Page comparison overhead constitutes hashing and comparison of page content. Merge overhead constitutes overhead from page write protection and page table entry (PTE) modifications. The overhead for comparisons is proportional to the number of pages to be scanned while merge overhead is proportional to the sharing opportunities that are identified by KSM.

Each PTE modification during the merge operation triggers an MMU notification. Interested kernel routines can register themselves to receive such notifications. KVM is one such kernel module which registers to receive these notifications. KVM maintains one shadow page table for each guest VM to translate guest virtual addresses to host physical addresses. A PTE modification during merge updates the host physical address for a particular host virtual address causing an existing shadow PTE to become invalid. KVM keeps its shadow page tables in-sync through MMU notifications.

Whenever a virtual machine writes to a KSM shared page, the kernel automatically performs a copy-on-write break on that page. The kernel allocates a separate copy of the page, writes to it and marks it as dirty. High frequency of CoW breaks contributes to additional page sharing overhead.

2.3 Tools & KSM modifications

This section presents an overview of tools and modifications to KSM required for deeper analysis of page sharing opportunities, KSM sharing potential and measuring overheads of the sharing process.

- mtrace
  In order to accurately compute the absolute page sharing opportunities and total zero page sharing opportunities, we have developed a tool called mtrace. mtrace periodically scans all the memory allocated to KVM virtual machines processes to compute potential sharing opportunities. mtrace employs hashing to determine pages with same content. The mtrace technique is similar to KSM but with the highest scan rate (scan all pages after each sleep duration) and the volatile page check turned off. mtrace executed over the allocated memory for a set of KVM virtual machines outputs the number of identical pages and the number of zero pages. We use mtrace as an oracle to estimate the instantaneous maximum sharing extent.

- Modifications to KSM
  (i) Estimation of effective sharing
  KSM internally keep tracks of several metrics. One such metric is the current count of pages shared by KSM. Whenever a page is scanned and merged (shared) this count is incremented. However, a
KSM shared page can be subjected to a CoW break much before KSM rescans the page. Such a CoW break is handled by the kernel independent of KSM, causing the KSM pages shared metric to be temporarily inconsistent. The degree of inconsistency depends on the number of CoW breaks and the full scan duration of KSM. We have instrumented an additional metric, effective sharing which is visible to both KSM and the kernel CoW break handler. The value of this new metric is incremented when KSM shares a page and decremented when the kernel handles a CoW break on a KSM shared page. As a result, the effective sharing metric captures the correct count of pages shared by KSM at any point of time.

(ii) Calculating page sharing durations
To understand the sharing behavior of different memory usage patterns, we instrument KSM to capture the duration for which a page remain shared after KSM identifies it to be shareable. The sharing duration is difference between the time instance when the kernel handles a CoW break on a KSM-shared page and when the page was identified and merged by KSM. As KSM maintains the list of pages merged for each stable node, the only overhead lies in finding the page subjected to CoW break from this list.

(iii) Counting number of zero pages
To understand the sources of sharing for different workloads, we instrument KSM to capture the number of zero pages shared at any point of time. We do this by searching for the zero page node in stable tree and finding the count of pages shared with it.

3 Empirical Analysis of KSM

Virtual machines host applications which exhibit time-varying memory usage patterns. Further, based on instantiation of new virtual machines or migration or shut down of existing virtual machines, the sharing potential will change. Such variations in memory usage and availability of potential memory for share will impact the effectiveness of sharing due to KSM. Further, the rate at which KSM operates and the rate at which memory is utilized (implying changes in available sharing opportunities) impact effectiveness of KSM as well. Also, the cost of sharing is related to the effectiveness of the sharing procedure. A potential goal is to extract maximum sharing benefits at the lowest resource utilization cost. Towards exploring these points, we empirically evaluate KSM behavior by studying the following aspects for workloads with different memory usage patterns:

- We use mtrace to capture instantaneous potential sharing opportunities presented by a set of workloads (applications) executing within virtual machines. Further, for these workloads, we compare the extent of sharing captured by KSM and the potential sharing opportunities.
- Characterizing the behavior of KSM—levels of sharing identified and CPU utilization overheads, as it’s configuration parameters are changed. Specifically, we are interested in tracking the improvement in sharing and the associated cost as the scanning rate is increased and compare it with the total available sharing opportunities.
- Study the impact of KSM scan rate on duration of page sharing. Duration of page sharing is used to determine the quotient of usable sharing, which further determines a good KSM scan rate—one which captures most of the usable sharing opportunities and minimizes CPU utilization overheads of the scanning process.

3.1 Experimentation Setup

Hardware: All experiments were conducted on IBM blade servers equipped with the Intel Xeon E5507 processor (8 cores) and 8 GB of physical memory. Ubuntu 12.04 Linux server with KVM hypervisor was used to create virtual machines. We pin virtual machines to CPU cores during all experiments to ensure deterministic behavior.

Workloads: We use the following workloads (applications running in virtual machines) for experimentation.

(i) The Kernel compilation workload [19] compiles the Linux 3.2 kernel with minimum modules (to reduce the compilation time).

(ii) The RUBiS workload [18] is an auction site prototype modeled after eBay.com. This workload consists of a web server and database server. Clients which run on different physical hosts send requests to web server based on a transition mix which determines the percentage of read and write operations. As part of our evaluation, we use browse-only requests with 1400 clients.

(iii) The Twitter workload [17] is representative of a large class of popular applications (i.e., social networking websites). Such applications are characterized as containing simple operations that access a large and complex graph-based data set. This workload allows us to configure the mix of operations that can be performed on database. For example, operations like get user tweets is a read only operation while operations like insert user tweet is a write operation. Our workload used 60% get-user-tweets and 30% insert-user-tweets with 10 clients each generating 600 requests/sec.

Each experiment is conducted using one or more of these workloads exhibiting different memory usage patterns.
as shown in Figure 2. As can be seen, the percentage of Resident Set Size (RSS) pages dirtied with RUBiS is much smaller than the Kernel compile and Twitter workloads. Further, the memory dirtying pattern with Kernel compile shows large fluctuations, while the dirty rate with Twitter is quite stable. We use these individual workloads and associated memory usage patterns, and combinations of workloads for characterizing the behavior of KSM.

3.2 Homogeneous workloads

We begin with an analysis of homogeneous workloads to ensure maximum sharing opportunities for our analysis.

(i) Idle virtual machines
The first workload consists of four idle VMs configured with 1GB RAM each. Two of them are Linux-based machines while the other two are Windows XP-based. All the four VMs are booted up and kept idle for the duration of experiment.

The maximum possible savings with the idle workload (Figure 3) is around 75% of 3GB (only 3GB of allocated memory is actually used by all idle VMs). Zero pages contribute to more than 50% of potential savings because the Windows-based VMs consume all allocated pages and initialize them to zero. With default scan rate, KSM could save 50% of 3GB in 1000 seconds. For higher scan rates, KSM can achieve maximum possible savings as shown in Figure 4. However, the savings do not increase proportionally beyond a scan rate of 800 pages per 200ms. This emphasizes the fact that choosing higher scan rates does not always result in increased savings. Also, the overhead of scanning and CoW breaks need to be considered while determining an ideal scan rate. Further, Table 1 lists the overheads of KSM—the CPU utilization, average number of CoW breaks and average memory savings, with different scan rates.

The CoW break count is low (compared to other workloads) at all scan rates as there is no significant page usage activity with idle VMs. However, CPU utilization increases significantly with increase in scan rate. A factor of 10x increase in CPU utilization, from 6% to 63%, is observed for scan rates of 800 pages and 100,000 pages per 200ms, respectively. The highest scan rate setting yielded additional sharing benefits of only 6% (increase in sharing with regards the memory resident set size).

(ii) Kernel compile workload
This workload consists of four kernel compile VMs configured with 512 MB RAM each. Each VM completely uses up the allocated memory within five minutes of
Table 1: CPU utilization, CoW breaks and memory savings at different KSM scan rates for idle VM workload.

<table>
<thead>
<tr>
<th>Scan rate (pages per 200ms)</th>
<th>KSM CPU utilization (avg %)</th>
<th>CoW breaks per minute (avg)</th>
<th>Average savings (% of RSS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>3.14</td>
<td>669</td>
<td>35.83</td>
</tr>
<tr>
<td>200</td>
<td>3.79</td>
<td>342</td>
<td>46.29</td>
</tr>
<tr>
<td>400</td>
<td>5.12</td>
<td>442</td>
<td>55.79</td>
</tr>
<tr>
<td>800</td>
<td>6.12</td>
<td>731</td>
<td>62.41</td>
</tr>
<tr>
<td>1000</td>
<td>6.45</td>
<td>893</td>
<td>63.73</td>
</tr>
<tr>
<td>2000</td>
<td>8</td>
<td>1576</td>
<td>65.96</td>
</tr>
<tr>
<td>4000</td>
<td>11.07</td>
<td>2663</td>
<td>66.58</td>
</tr>
<tr>
<td>8000</td>
<td>16.46</td>
<td>3899</td>
<td>66.88</td>
</tr>
<tr>
<td>10000</td>
<td>18.75</td>
<td>4051</td>
<td>67.09</td>
</tr>
<tr>
<td>20000</td>
<td>29.28</td>
<td>4287</td>
<td>67.66</td>
</tr>
<tr>
<td>40000</td>
<td>43.36</td>
<td>4463</td>
<td>68.02</td>
</tr>
<tr>
<td>80000</td>
<td>58.48</td>
<td>4608</td>
<td>68.27</td>
</tr>
<tr>
<td>100000</td>
<td>62.98</td>
<td>4600</td>
<td>68.27</td>
</tr>
</tbody>
</table>

Table 1: CPU utilization, CoW breaks and memory savings at different KSM scan rates for idle VM workload.

starting the compilation process. The maximum possible savings with kernel compile workload is almost 60% of 2GB, as shown in Figure 5. However, with the default scan rate, KSM could save only 5-10% of all pages which is much less than the maximum possible savings. The reason for reduced savings is that the scan rate is very low to capture all the sharing opportunities that exist for a duration less than the KSM full scan time. Also, kernel compilation being I/O intensive, causes the disk cache in Linux to use free pages to speed up disk access. This results in a very low value of potential zero page sharing opportunities.

As shown in Figure 5, KSM achieves maximum savings at a scan rate of 10,000 pages per 200ms. Also, beyond this scan rate, increase in savings is not proportional to overhead as shown in Table 2. Number of CoW breaks are much higher for the kernel compilation workload, about 10x higher than the RUBiS workload, due to its I/O intensive nature.

Majority of the memory savings achieved at lower scan rates is from long term sharing opportunities or partially exploited sharing opportunities. Figure 6 shows that only 35% of sharing lasts below 200 seconds for scan rates 100, 400 and 800 pages per 200ms. For higher scan rates, about 60% of sharing lasts below 200 seconds indicating that majority of sharing achieved at higher scan rates is from short term sharing opportunities. Thus, if usability of sharing is related to its duration, with increase in scan rate, memory savings may increase but usable memory savings may not.

(iii) RUBiS workloads

This workload consists of two RUBiS setups started in parallel. RUBiS web VM is configured with 776MB of RAM while RUBiS database VM is configured with 384MB of RAM for both the setups. We consider a browse only transition mix for both the setups to study sharing behavior in case of read-only memory usage.

With default scan rate, KSM can achieve most of the possible savings (Figure 7). This increase in memory savings for default KSM scan rate can be divided into two different segments. The first segment between 0 to 1000 seconds indicates a slow increase while the second segment between 1000 to 1500 seconds indicates a fast increase. Apart from the scan time required to cover all memory pages, another reason could be the time taken for disk cache stabilization in both the VMs. The CoW break count for this kind of a read intensive workload is less compared to a write intensive kernel compile workload as shown in Table 2.

The sharing duration CDF (Figure 8) shows that with
Table 2: CPU utilization and CoW breaks at different scan rates for kernel compile and RUBiS workloads.

lower scan rates most of the sharing is long lasting. However, with intermediate rates the sharing duration behavior is not significantly different indicating that having a higher scan rate can achieve the maximum possible savings quickly.

3.3 Heterogeneous workload

Virtual machines on a physical host are unlikely to be running the same kind of workloads. We experiment with a heterogeneous workload which consists of four VMs—one VM executing the kernel compile workload, two VMs executing the RUBiS workload, and one VM executing the Twitter workload, with a total memory allocation of 2GB.

With the heterogeneous workload, maximum possible savings are very low (around 20% of 2GB). With the default KSM setting, very small portion of the sharing is realized, as shown in Figure 9. With scan rates of 800 pages and 4000 pages per 200ms, average savings obtained by KSM are 12% and 13% respectively. Further, savings do not increase significantly for higher scan rates. An abrupt increase in CoW breaks (by a factor of 3) is observed when scan rate increases from 4000 pages/scan to 8000 pages per 200ms (refer to Table 3).

Figure 10 plots the cumulative distribution of the sharing durations with the heterogeneous workload. As can be seen, a high scan rate of 8000 pages/scan results in a large number of short term sharing (90% pages shared for less than a minute). While scan rates of 100 pages/scan and 400 pages/scan yield larger median sharing durations (50% pages are shared for more than 4 minutes). The corresponding average sharing due to KSM is 4% and 11%, respectively. Taking into account both the median sharing durations and extent of sharing, a scan rate of 400 pages/scan seems to work the best.

3.4 Dynamic workload

A physical machine hosts different VMs at different points in time. VMs get created, destroyed, migrated and swapped between physical hosts, which changes the opportunities for sharing. To test the behavior of KSM under changing memory availability for sharing, we create a dynamic workload.

The dynamic workload starts with the kernel compile workload on two VMs, each configured with 512 MB of RAM and one idle Windows XP VM configured with 1GB RAM. After 25 minutes the kernel compile VMs are shutdown. Two VMs configured with 768MB RAM each running the Twitter workload are booted at 40th minute.
<table>
<thead>
<tr>
<th>Scan rate (pages per 200ms)</th>
<th>CPU utilization (%)</th>
<th>CoW breaks per minute (avg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.9</td>
<td>406</td>
</tr>
<tr>
<td>200</td>
<td>1.5</td>
<td>1136</td>
</tr>
<tr>
<td>400</td>
<td>2.7</td>
<td>2304</td>
</tr>
<tr>
<td>800</td>
<td>4.8</td>
<td>4463</td>
</tr>
<tr>
<td>1000</td>
<td>6.1</td>
<td>4575</td>
</tr>
<tr>
<td>2000</td>
<td>10.7</td>
<td>6426</td>
</tr>
<tr>
<td>4000</td>
<td>18.3</td>
<td>8490</td>
</tr>
<tr>
<td>8000</td>
<td>30.6</td>
<td>23930</td>
</tr>
<tr>
<td>10000</td>
<td>35.4</td>
<td>33520</td>
</tr>
<tr>
<td>20000</td>
<td>51.4</td>
<td>91346</td>
</tr>
<tr>
<td>40000</td>
<td>66.2</td>
<td>146267</td>
</tr>
<tr>
<td>80000</td>
<td>76.4</td>
<td>197668</td>
</tr>
</tbody>
</table>

Table 3: Average CPU utilization and CoW breaks at different scan rates with heterogeneous workloads.

Figure 9: Memory savings at different scan rates (fixed KSM sleep time of 200 ms) with heterogeneous workloads.

Figure 10: Sharing duration distribution at different scan rates with heterogeneous workloads.

Figure 11: Memory savings at different scan rates of KSM with the Dynamic workload.

With increased scan rates, KSM achieved savings are close to the potential maximum savings during the kernel compile phase (Figure 11). However, for with the Twitter workload, the savings improve with increase in scan rate but do so slowly, with a scan rate of 1000 pages/scan even after 10 minutes KSM lags behind the maximum possible sharing. The different memory usage patterns of workloads demonstrate that KSM configurations need to change to adapt to changing sharing opportunities.

3.5 Summary of results

Based on experiments with different workloads we summarize our findings as follows,

- With default settings, KSM can seldom achieve the maximum possible savings. This was observed over all workloads.
- Very high scan rates are not suited to achieve maxi-
The adaptive approach uses a set of KSM-based metrics for adaptation. Figure 12 shows different metrics estimated along a time axis. At a time instant \( t \), the following KSM-based metrics are estimated, \( p(t) \): number of pages saved at time \( t \) (KSM effective sharing less number of unique pages that are shared), \( c(t) \): number of CoW breaks till time \( t \), \( u(t) \): KSM cpu time till time \( t \) and \( s(t_1,t_2) \): KSM scan rate in the interval \([t_1,t_2]\). Based on these metrics, the adaptive approach first calculates a set of secondary metrics to be used for adaptation. These metrics are,

- usable sharing duration (\( usd \)) is an input which specifies the minimum duration of sharing opportunities we want KSM to exploit. Any sharing opportunity that lasts below this duration is not considered useful.
- new pages shared (\( nps \)) represents the number of pages shared by KSM in a given time duration and is given by,

\[
nps(t_0,t_1) = p(t_1) - p(t_0) + c(t_1) - c(t_0)
\]

- unusable pages (\( up \)) represents the number of page sharings that last below usable sharing duration. We approximate this value at decision point \( t_1 \) by taking the number of sharing breaks during \( t_0 \) and \( t_1 \) that last below usable sharing duration.
- usable shared pages (\( usp \)) represents the number of useful pages shared in a given time duration and is given by,

\[
usp(t_0,t_1) = nps(t_0,t_1) - up(t_0,t_1)
\]

- ksm cpu utilization (\( ksu \)) represents the CPU utilization of KSM thread in a given time duration.
- sharing gain (\( sg \)) represents the increase in usable shared pages (\( usp \)) per unit increase in KSM CPU utilization and is given by,

\[
sg(t_1,t_2) = \frac{usp(t_1,t_2) - usp(t_0,t_1)}{usp(t_0,t_1) \times [ksu(t_1,t_2) - ksu(t_0,t_1)]}
\]

The adaptive approach operates at discrete time intervals and takes decisions to adapt the KSM scan rate at these decision points. The scan rate adaptation is split into four states—(i)ExpInc, a transition into this state doubles the scan rate, (ii)ExpDec, a transition into this state halves the scan rate, (iii)Constant, a transition into this state maintains the current scan rate, and (iv)Transient, a transition into this state set the scan rate to the average of the last two scan rates.

Table 4 shows the conditions checked in each state (at a decision point) to determine the next state (to adapt the KSM scan rate). While in the ExpInc state, if the sharing gain increases beyond a threshold (\( \text{ExpThreshold} \)), it implies that sharing opportunities are increasing and hence
a higher scan rate is desirable. Hence, the same state is maintained and the scan rate doubled. On the other hand, if sharing gains are decreasing below the threshold, the scan rate is already high and needs to be decreased. This condition changes the scan rate state to Transient, where scan rate is set to the average of the previous two scan rates.

Decrease in sharing gain below ExpThresh between two decision points in ExpDec state means the KSM scan rate is lower than the desired level and need to be increased. In this condition, the scan rate is set to the average of previous two scan rates by performing a transition to Transient state. Otherwise, if the exponential decrease of scan rate is not impacting the sharing gain drastically, ExpDec is continued and the scan rate is halved. ExpThresh, the threshold for the increase in usable sharing per unit additional CPU increase between two scan rates is set to 0.2 for both the exponential states in our implementation.

In the Transient state, the average scan rate of last two decision points is used to find out whether the scan rate is ideal and can be continued in future. If the transition from the exponential states was because of a temporary workload behavior change or the workload behavior changes during Transient state, the usable sharing will not be bounded by the lower(usplow) and upper limit(usphigh) values of usable sharing determined in the last two decision points. In this case the exponential states are entered to quickly adapt to the workload change. Otherwise, the average scan rate used in this state is considered to be an approximate to the desired scan rate and is continued by entering the Constant state.

The Constant state represents a steady scan rate where the usp is not fluctuating beyond a threshold(uThresh). Change in usable shared pages between last and current decision points(represented as delta(usp)) in the constant state represents the change in sharing behavior in the workload. If there is a drastic change in the workload while in this state, a new scan rate should be explored quickly. For this reason, if the delta(usp) is beyond uThresh, we jump to either of the exponential states depending on the direction of change in usp. uThresh is set to 0.1% of the aggregate RSS in our implementation. In some cases (like kernel compile), we found that staying at a low scan rate does not give any indication of change in sharing behavior. So, the number of times the Constant state is continued is bounded by a threshold (RoundThresh) beyond which ExpInc is entered to do a force check on new sharing opportunities.

In our implementation, the KSM scan rate is changed while keeping the scanning interval fixed at 200ms. The lowest possible scan rate in the adaptive sharing technique is bounded by 100 pages for 200ms. So, if at any state the scan rate goes below 100 pages, Constant state is entered with scan rate of 100 pages per 200 ms.

### 4.2 Evaluation

To evaluate the adaptive approach for setting KSM scan rate, we execute a micro-benchmark within single virtual machine with 1GB RAM. The benchmark allocates 400MB of memory and writes same content after 300 seconds from the start of the experiment into all the memory pages at a rate of 300 pages/second. Once the content of all the pages is made same, the benchmark allocates 400MB of memory and writes same content after 300 seconds. Half of the pages are dirtied at the rate 300 pages/second by changing their content to different values (sharing halved). After that, It sleeps for another 300 seconds before finally recopying the same content to all pages to achieve the original level of sharing.

After 300 seconds into the experiment when the sharing

<table>
<thead>
<tr>
<th>Current State</th>
<th>Transition Condition</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExpInc</td>
<td>sg &gt; ExpThresh</td>
<td>ExpInc</td>
</tr>
<tr>
<td></td>
<td>sg ≤ ExpThresh</td>
<td>Transient</td>
</tr>
<tr>
<td>ExpDec</td>
<td>-sg &lt; ExpThresh</td>
<td>ExpDec</td>
</tr>
<tr>
<td></td>
<td>-sg ≥ ExpThresh</td>
<td>ExpDec</td>
</tr>
<tr>
<td>Transient</td>
<td>usp &gt; uspHigh</td>
<td>ExpInc</td>
</tr>
<tr>
<td></td>
<td>usp &lt; uspLow</td>
<td>ExpDec</td>
</tr>
<tr>
<td></td>
<td>uspLow ≤ usp ≤ uspHigh</td>
<td>Constant</td>
</tr>
<tr>
<td>Constant</td>
<td>-uThresh ≤ delta(usp) ≤ uThresh</td>
<td>Constant</td>
</tr>
<tr>
<td></td>
<td>delta(usp) &lt; -uThresh</td>
<td>ExpDec</td>
</tr>
<tr>
<td></td>
<td>delta(usp) &gt; uThresh</td>
<td>ExpInc</td>
</tr>
<tr>
<td></td>
<td>ConstantRounds &gt; RoundThresh</td>
<td>ExpInc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time (sec)</th>
<th>Adaptive KSM savings</th>
<th>Max savings</th>
<th>KSM scan rate (pages per 200ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>500</td>
<td>500</td>
<td>90</td>
<td>100</td>
</tr>
<tr>
<td>1000</td>
<td>100</td>
<td>80</td>
<td>200</td>
</tr>
<tr>
<td>1500</td>
<td>100</td>
<td>70</td>
<td>300</td>
</tr>
<tr>
<td>2000</td>
<td>100</td>
<td>60</td>
<td>400</td>
</tr>
</tbody>
</table>

Table 4: Transition logic of the adaptive page sharing approach.

Figure 13: Effectiveness of the adaptive approach for setting KSM scan rates.
opportunities increase (Figure 13), the scan rate is increased exponentially, up to 800 pages per 200 ms. At 600 seconds from the start of the experiment, the sharing opportunities saturate and no additional sharing is available. The adaptive approach brings down the scan rate quickly to the lowest scan rate (100 pages per 200 ms). When sharing is halved, no action is taken as KSM is already at the lowest scan rate. When the sharing increases in the final phase, the adaptive approach increases the scan rate to 400 pages per 200 ms to take advantage of new sharing opportunities. Once the sharing stabilizes at 1500 seconds, scan rate exponentially comes down to the default value. This shows that the adaptive scheme can respond to dynamic sharing opportunities quickly without unnecessarily scanning at a higher rates.

Next, we evaluate kernel compile workload with 4 VMs each configured with 512 MB memory. Figure 14 compares the memory savings achieved with default KSM configuration (100 pages in 200 ms) and the adaptive sharing technique. KSM with rate adaptation achieves 80% of the maximum potential savings (on average) compared to 8% of maximum potential savings achieved with default KSM settings. We observed an average CPU utilization of 5% for KSM with adaptive scheme compared to 1% with the default KSM scan rate. However, there is a significant increase (10x) in memory savings between the default and the adaptive configurations of KSM with 4% additional CPU. Figure 15 shows the sharing duration distribution with the adaptive sharing technique. 50% of sharing for KSM with rate adaptation last longer than approximately 6 minutes while around 25% of savings last for more than 6 minutes with the default KSM scan rate. Considering difference in effective memory saving between adaptive sharing technique and default settings, it is clear that the adaptive scheme is more profitable. For the higher scan rate (40000 pages per 200 ms), not only all the presented memory sharing opportunities are captured at a price of more CPU utilization but also 20% sharing lasts for more than 6 minutes. Considering both the amount of sharing achieved with KSM rate adaptation and the sharing duration distribution, it can be easily seen that adaptive technique performs much better than both default KSM scan settings and aggressive scan settings with very little CPU overhead.

5 Related Work

The Disco [9] virtual machine monitor implements an out-of-band content based page sharing technique. However, the implementation required modifications to guest OS. The VMware ESX server [8] implements page sharing transparent to guest OS using a hash based technique to identify duplicate pages. Similarly, KSM [15] implements page sharing transparent to the guest OS. Further, the Difference engine [10] extended page sharing to sub-page granularity along with compression techniques to improve sharing efficiency. KSM++ [13] adopts technique through I/O hints, by identifying pages in the host that are most likely also present in the guest’s buffer cache. Identifying such pages and proactively scanning them helps locate identical pages quickly. While our empirical evaluation was specific to KSM, the methodology can be extended to characterize other sharing techniques as well, and a characterization with sub-page sharing and with the KSM++ technique would be interesting.

Another useful technique for memory overcommitment has been the use of ballooning [8]. With the help of balloon driver within the guest the balloon can be inflated or deflated to control memory available to a guest OS. Ballooning can impact the feasibility of memory sharing, as it can change the set of pages in memory—inflation of
the balloon results in pages being swapped out. Characterizing behavior of sharing with the ballooning procedure (for different balloon-size management techniques) would be an interesting extension to our work.

Memory Buddies [14] uses bloom filters to determine sharing opportunities across a set of virtual machines. Their work focuses on sharing aware co-location of VMs. Our analysis can be used to extend their work by considering sharing durations of workloads as inputs to achieve efficient co-location. For example, a co-location decision should be taken only if more than 50% pages are shared for more than 5 minutes. Our work, can answer such questions and can compliment resource management techniques that rely on extent of sharing for resource allocation decisions.

Singleton [16] extends KSM to find duplicate pages in host and guest page caches. It scrubs duplicate pages in the host page cache to maintain a single copy of each page and reduce the memory footprint of the virtualized setup. Both anonymous and page cache pages of a guest VM are backed up by anonymous pages of host. We would like to analyze the contribution to page sharing from guest anonymous pages and guest page cache pages. Such an analysis would allow us to improve the adaptation technique to make differentiated decisions based on whether the shared pages are anonymous pages or are from the page cache.

[6] focused on identifying sources of sharing opportunities in a virtualized setup. Their aim was to compare sources of sharing opportunities in intra-VM and inter-VM cases. Our study differs from this work as we empirically evaluate the potential of the sharing sub-system(KSM) itself. [7] analyzed the memory savings obtained using KSM for different benchmarks. While this work overlaps with [7], our focus is to determine the extent of benefits and overheads for KSM for workloads with different memory usage patterns and to develop an adaptive technique to choose the right KSM configuration to maximize usable sharing.

6 Conclusions

As part of this work, we studied the trade-off between extent of sharing opportunities (compared to actual sharing potential) that can be identified and the overheads for doing the same for Kernel Samepage Merging. For several combinations of workloads, exhibiting different memory usage characteristics, we benchmarked KSM performance (achieved savings vs. total possible savings) and CPU overheads for different KSM configurations. The benchmarking results were used to develop an adaptive scheme to dynamically reconfigure KSM configuration parameters (scan rate, number of pages to scan per scan round), to maximize sharing opportunities at minimal overheads. We used the notion of useful sharing to guide the adaptive sharing technique. We evaluated the adaptive technique and compared it with default settings of KSM. Our experiments demonstrate that it faithfully adapts scan rate based on sharing potential and yielded up to 10x more savings with negligible increase in CPU utilization.

References