Group-based memory oversubscription for virtualized clouds

Sangwook Kim, Hwanju Kim, Joonwon Lee, Jinkyu Jeong*

College of Information and Communication Engineering, Sungkyunkwan University, South Korea

ABSTRACT

As memory resource is a primary inhibitor of oversubscribing data centers in virtualized clouds, efficient memory management has been more appealing to public cloud providers. Although memory oversubscription improves overall memory efficiency, existing schemes lack isolation support, which is crucial for clouds to provide pay-per-use services on multi-tenant resource pools. This paper presents group-based memory oversubscription that confines both mechanism and policy of memory oversubscription into a group of virtual machines. A group is specified as one of service level agreements so that a cloud customer can control the memory management mechanism within its own isolated domain. We introduce group-based memory deduplication and reprovisioning with several policies based on per-group workload behaviors. The proposed scheme is implemented on the KVM-based prototype and evaluated with realistic cloud workloads such as MapReduce and MPI applications. The evaluation results show that our group-based memory oversubscription ensures strict inter-group isolation while achieving intra-group memory efficiency, compared to a system-wide scheme, by adapting oversubscription policies based on per-group workload characteristics.

1. Introduction

Current Infrastructure-as-a-service (IaaS) clouds regard trustworthiness and performance isolation as a major requirement because of their multi-tenant nature. Such isolation demand, however, inhibits the level of oversubscription that allows independent virtual machines (VMs) to share underlying resources, thereby losing the opportunities of gaining more profits. In order to save total cost of ownership, it is important for cloud providers to efficiently share limited resources while guaranteeing service level agreements (SLAs). Several proposals have addressed the issues on isolation over several virtualized resources in cloud environments [26, 12, 19, 11].

Among multi-tenanted resources, memory is a primary inhibitor of oversubscribing data centers due to the nontrivial cost of extension and power consumption [14]. Although modern multi-core processors enable high consolidation ratio, the limitation of memory capacity extension cannot help drawing a line at a lower level of consolidation. In order to alleviate this limitation, many researchers have introduced efficient memory management schemes, such as memory deduplication [36, 14, 27] and dynamic memory balancing [18, 23, 45], which allow memory oversubscription by flexibly reprovisioning redundant and unused memory. Memory oversubscription has been considered to be an attractive feature to cloud providers by effectively hosting the increasing number of customers.

Although memory oversubscription improves memory efficiency, it is not trivial to be employed for cloud providers, since existing schemes weaken trustworthiness [27, 33] and performance isolation [40, 11]. Current memory oversubscription is provided as a system-wide hypervisor (i.e., virtual machine monitor) service, which cannot be controlled and isolated by cloud users. For example, system-wide memory deduplication can impose security breaches by allowing sensitive memory contents to be shared among independent customers [27, 33]. In addition, deduplication overhead from one user could interfere the performance of another who is not being involved in the deduplication. Finally, various
policies of memory oversubscription can neither be adapted nor customized to the workload and specific need of each cloud user.

This paper introduces a group-based memory management scheme for virtualized clouds to achieve inter-group isolation as well as intra-group efficiency on memory oversubscription. Fundamentally, the proposed scheme confines the mechanism and policy of memory oversubscription to a group of VMs in order to ensure security and performance isolation between different groups. Group specification is delegated to cloud customers so that they can decide the level of isolation. In order to enable per-group memory management, our scheme isolates memory deduplication and repriorizing on a group basis. By doing so, memory contents and capacity are securely protected within a group, and oversubscription policy is configurable and adaptive for workload demand of each group.

In order to improve intra-group memory efficiency besides inter-group isolation support, we present three group-based memory oversubscription policies: adaptive scan rate, demand-based memory repriorization, and hypervisor-level secondary cache. First, the adaptive scan rate policy dynamically adjusts the rate of memory scanning, which is required to find identical pages for deduplication, for each group by monitoring workloads. Our algorithm takes CPU utilization, swap activities, and deduplication rates into account to figure out effective scan rates. Second, demand-based memory repriorization policy distributes per-group surplus memory, which is unused and reclaimed by deduplication, to its group members based on their memory demands. Finally, hypervisor-level secondary cache allows the per-group surplus memory to be used as an exclusive cache that stores pages evicted by VMs within the group.

The group-based oversubscription, however, would degrade system-wide memory efficiency at the expense of strict inter-group isolation. For instance, the group-based scheme prevents a group, which suffers from memory insufficiency, from exploiting underutilized surplus memory of another group. To deal with this limitation, we additionally propose a demand-based overhead migration as a complementary policy for the group-based policies to improve system-wide memory efficiency without breaking inter-group isolation.

The proposed scheme is implemented and evaluated on the KVM hypervisor [22]. We extended KSM (Kernel Samepage Merging) [1] to be a subsystem of Linux group [25] for group-based memory deduplication and repriorization, with which the group-based policies are implemented as user-level daemons. The evaluation results show that our group-based memory oversubscription ensures strict inter-group isolation while achieving intra-group memory efficiency over various realistic scenarios using MapReduce, MPI, and a file-intensive workload.

The rest of this paper is organized as follows: Sections 2 and 3 describe the background and motivation behind the proposed scheme, respectively. Sections 4 and 5 explain the design and implementation of the group-based memory management. Section 6 presents evaluation results on our proposed scheme, and the complementary policy for improving system-wide memory efficiency is explained in Section 7. In Section 8, we discuss the applicability of our scheme in cloud environments. Finally, we present related work in Section 9 and conclusions in Section 10.

2. Background

This section presents background on VM memory management and isolation techniques in clouds.

2.1. VM memory management

VM memory management schemes have been mostly focused on memory oversubscription, which allows the hypervisor to allocate more memory to colocated VMs than actual physical memory. The first part of memory oversubscription is to find surplus memory, which is allocated but not actually needed. Memory deduplication and working set estimation can be used in this part. Once surplus memory is secured, it can be re provisioned to memory-hungry or newly-instantiated VMs in order to increase memory utilization.

Memory deduplication [36,14,27] is a well-known technique that reclaims redundant memory by sharing identical pages. To transparently search identical pages, most schemes periodically scan and compare the contents of existing VM memory. Once identical pages are found, they are merged into a single unique page while redundant ones are reclaimed by the hypervisor. The page table entries referenced to the unique page are marked as copy-on-write, by which modification of each VM is isolated to its own page. During the lifetime between merging and copy-on-write breaking, the hypervisor can maintain additional memory that can be used for other purposes if necessary.

In collaboration with memory deduplication, reclaimed memory can be re provisioned to existing VMs. The memory ballooning mechanism [36] allows the hypervisor to dynamically increase and decrease VM memory with the aid of guest OS extension. Using this mechanism, surplus memory can be provisioned to a VM that demands more memory beyond its allocation. To identify memory-hungry VMs, existing schemes monitor paging operations such as swap activity [45,40,42]. Reprovisioned memory is used to preserve more working set in memory so that expensive disk I/O operations can be reduced.

2.2. Isolation in clouds

Isolation is an essential support for cloud computing, since cloud providers offer pay-per-use resources to their customers who need certain levels of SLAs. The simplest way of ensuring isolation is to allot dedicated resources to each customer. Many cloud providers, however, strive to save infrastructure costs by oversubscribing their data centers, so that underutilized capacity is effectively provisioned via resource sharing. Therefore, isolation support on shared resources is a key issue cloud providers should deal with in order to maximize their profits with QoS guarantee.

Current IaaS providers have been relying on resource-level isolation policies where SLAs are specified in the form of resource requirement. In this policy, resource capacity for which a customer pays is believed to be isolated by the cloud provider. The resource-level isolation limits the extent to which a cloud provider oversubscribes data center, but ensures strong isolation guarantee. On the other side, several researchers have argued that performance-level isolation can increase oversubscription ratio for more profits [11, 26,12]. This policy aims to achieve the performance specified by a customer, regardless of how much resource capacity is provisioned. Although this type of isolation is attractive to providers, it is challenging to estimate dynamically varying performance and identify relationship between performance and various types of resources.

3. Motivation

In this section, we first show the significance of the overhead arising from memory deduplication in terms of performance interference. Next, we estimate the impact of group-based memory deduplication in aspects of shareable memory since serious reduction of shareable memory can offset the advantage of memory isolation. Then, we clarify the need for group-based customization of memory oversubscription by describing the workload heterogeneity with inherent memory characteristics in virtualized clouds. Finally, we argue the limitation of existing memory oversubscription schemes from the perspective of isolation and flexibility.
3.1. Overhead analysis

Memory deduplication inherently consumes physical resources such as CPU, because it periodically scans the registered memory regions and conducts byte-by-byte comparison of each memory page. Hence, conducting memory deduplication on a same physical core with a CPU-intensive workload can degrade the performance of that workload. In order to quantify the impact of memory deduplication on a colocated workload, we conducted an experiment using the Pi benchmark running on a VM with one vCPU and 1 GB of memory. As a memory deduplication technique, we used Linux KSM [1]; detailed description of our experimental platform is presented in Section 6.

Fig. 1 shows execution time of the Pi benchmark normalized to the baseline, which is the case where the Pi benchmark solely ran without the interference of memory deduplication, as scan rate and relative scheduling priority (i.e., nice value in Linux) of memory deduplication change. As we can see in Fig. 1(a), the performance of the Pi benchmark is gradually degraded when the scan rate and priority are increased (i.e., nice value is decreased). This impact can be explained through increased CPU utilization of ksmd, which is a kernel thread conducting memory deduplication, as the scan rate increases (Fig. 1(b)). Note that the fluctuation of CPU utilization occurs because ksmd periodically sleeps for a while (20 ms for the default configuration).

To further analyze the behavior of memory deduplication, we measured per-function CPU cycle consumption in KSM for the NICE-5 with 50 pages/ms case using perf in Linux. Fig. 2 shows the per-function breakdown of CPU cycle consumption of KSM. As we can see in the figure, the dominant sources of CPU cycle consumption are checksum calculation (calc_checksum) and byte-by-byte comparison (memcmp_pages). In KSM, two red–black trees, stable and unstable tree, are used to conduct memory deduplication. The stable tree contains only already shared and write-protected pages while the unstable tree contains only pages that are not shared yet, but that are tracked by KSM. Since checksumming is iteratively conducted for all registered memory pages in order to avoid inserting frequently modified pages to the unstable tree, it inevitably consumes a lot of CPU cycles. Byte-by-byte comparison also consumes a lot of CPU cycles, because it is frequently carried out to identify identical pages using their contents for both the stable and unstable trees. The remaining portions of CPU cycle consumption are mostly comprised of functions related to finding target pages, searching the two red–black trees, and handling semaphores.

3.2. Shareable memory between individual groups

Memory deduplication technique has been proposed as a way of reclaiming redundant memory in virtualized environments based on the fact that there are numerous opportunities for sharing memory between VMs [36]. On the contrary, more recent work [2] has reported that sharing within individual VMs often accounts for over 90% of the sharing potential within a set of VMs, with inter-VM sharing contributing only a small amount. In addition to the large amount of intra-VM sharing potentials, the heterogeneous software stacks (e.g., OS, library) and workloads are common for individual groups in a virtualized cloud. Therefore, we can expect the amount of shareable memory across groups is commonly insignificant.

In order to investigate the sharing potentials across individual groups, we ran realistic workloads for cloud environments comprising the MapReduce Pi estimator (PI) and the NPB BT (BT); see Section 6 for more details. In addition to the realistic workloads, we also experimented a case where two groups of VMs are freshly booted, but are not running any further workload for 5 min (IDLE). Table 1 shows the difference of shareable memory between system-wide and group-based memory deduplication for various workloads using four VMs each of which has one vCPU and 1 GB of memory, for each group; the amount of shared zero pages is given in parenthesis for each configuration. We also measured the amount of shareable memory when memory deduplication on all the eight VMs is isolated from each other (i.e., per-VM group).

We first compared the difference of shareable memory when installed OSes for two groups are either same or differ from each other in the IDLE case. As we expected, the amount of shared memory is decreased (14.82%) with the group-based memory deduplication compared to the system-wide one, when the exact same version of Ubuntu is installed for both groups. On the other hand, there is no noticeable difference between the system-wide and

Fig. 1. Analysis results for the interference of memory deduplication to the colocated workload; X pages/ms and NICE-X indicate how many pages are scanned per millisecond (i.e., scan rate) and the priority of ksmd (lower value means higher priority and the default value is 5 in KSM), respectively.

Fig. 2. CPU cycle breakdown for memory deduplication in Linux KSM.

1 http://www.numberworld.org/y-cruncher/.
group-based memory deduplication when Ubuntu and Windows are installed respectively for each group. For the per-VM group configurations, the amount of shared memory in Ubuntu+Ubuntu is dropped significantly whereas the reduction of shared memory in Ubuntu+Windows is relatively small. This is because the main source of memory deduplication is zero pages, mostly generated by Windows, in Ubuntu+Windows.

Next, we compared the difference of shared memory when both groups run their applications. Similar to the IDLE case, the amount of shared memory is decreased for both PI + PI (18.52%) and PI + BT (11.29%). Per-VM group-based memory deduplication additionally reduces the amount of shared memory by 52.39% and 65.19% for PI + PI and PI + BT, respectively, since common code and data from the identical application or OS cannot be shared across VMs. Unlike the IDLE case, zero pages take a large portion of shared memory in the both cases because PI generates a large amount of zero pages during its execution.

In summary, the degree of reduction in shareable memory with the group-based memory deduplication is determined by the software stack (e.g., OS, application) of each group. Hence, the group-based memory deduplication can reduce the amount of shared memory at the expense of inter-group isolation if independent groups of VMs have common software stacks. However, considering both homogeneity inside a group and heterogeneity among independent groups in virtualized clouds (e.g., Ubuntu + Windows), the impact of group-based scheme on the amount of shareable memory would be insignificant.

3.3. Workload characteristics in virtualized clouds

In a virtualized cloud (i.e., IaaS cloud), different groups (i.e., customers) typically run different workloads on the allocated set of VMs. For instance, a group runs a memory-hungry workload whereas another group runs a CPU-intensive workload. In this case, the memory-hungry group can exploit aggressive memory deduplication to rapidly resolve its memory insufficiency. On the other hand, it would be better for the CPU-intensive group to disable memory deduplication so that the interference from memory deduplication can be minimized. Therefore, a memory management scheme for virtualized clouds should consider the workload characteristic of each group in order to improve memory efficiency while minimizing the interference from memory deduplication.

Furthermore, the asymmetry of resource demand can also exist within a single group. As an example, a customer rents a set of VMs to construct a virtualized cluster that can be shared by different types of workloads [17]. Since each VM in this virtualized cluster can host different types of workloads (e.g., MapReduce or MPI job), memory demand can be varied across the group’s VMs. Moreover, the asymmetry of resource demand can exist even in a single workload if each VM involving that workload has different role. For example, MapReduce framework is comprised of a master node and slave nodes for executing a MapReduce job. The main role of the master node is coordinating the slave nodes to do the actual work. Hence, the master node shows lower memory demand in general than the slave nodes (see Fig. 10 for the empirical result). As a result, the asymmetry of resource demand can exist in a single MapReduce workload when the master VM and the slave VMs belong to the same group.

3.4. Limitations of memory oversubscription

Although VM memory management improves overall memory efficiency via oversubscription, existing schemes cannot guarantee isolation among independent customer instances. First, system-wide memory deduplication allows a malicious VM to inspect the existence of sensitive information (e.g., vulnerable applications) of another VM [27, 33]. Second, deduplication overheads, including scanning, comparison, and copy-on-writes, are not properly accounted to the only VMs that are involved in deduplication process. Third, memory reprovisioning cannot be controlled within a permitted boundary by customers. In general, customers do not want their surplus memory to be provided to others without appropriate compensation. Finally, unified policies are inflexible, since the demands of memory oversubscription are varied depending on workloads of each customer.

In order to address the limitations, we introduce group-based isolation to VM memory management. Basically, our scheme provides resource-level isolation between independent groups so that each group’s memory resource is strictly guaranteed without being provided to another group. Although this type of isolation may degrade overall memory efficiency by prohibiting inter-group memory sharing, it guarantees strong security and performance isolation with group SLAs instead of group SLAs. Instead, our scheme improves per-group memory efficiency through sophisticated policies that are adapted to workload characteristics for each group.

4. Design

This section explains our design of the group-based memory oversubscription scheme. We first present the mechanism for achieving isolation of the deduplication process among individual groups (i.e., independent customers). Next, we explain the three policies to efficiently manage the memory resources that reflecting a group’s characteristic and demand to the management scheme.

4.1. Group-based memory deduplication

Nested virtualization [4, 41] is a feasible way of isolating a group of VMs by running them in a nested hypervisor hosted on a bottom-layer VM. By allowing a cloud user to manipulate its own hypervisor services, isolated and flexible resource management such as isolated memory deduplication is viable. However, nested virtualization entails nontrivial performance overheads and restrictive assumptions on the current implementation [4, 41].

In this work, therefore, we enable the hypervisor to run multiple deduplication threads, each of which is designated to a group of VMs, in order to isolate the memory deduplication process. Specifically, in our group-based memory deduplication, a VM registers its allocated memory to a deduplication thread that is in charge of its group. Per-group deduplication thread is only involved in memory regions that are registered to its group, so that all deduplication-related operations are exclusively carried out on a delegated memory space of each group. Since a deduplication thread only merges pages that belong to the same group, a malicious user cannot attempt to infer another user’s memory contents.
Group-based exclusive deduplication allows the hypervisor to properly account the computational overheads of a deduplication thread into its corresponding group. The majority overhead of a deduplication thread arises from content-based comparison, which includes hashing (or checksumming) and byte-by-byte comparison of scanned pages. Since this operation becomes more compute-intensive as a scan rate increases, it could degrade the performance of main workloads. By accounting the CPU usage of a deduplication thread to its group, such overhead cannot affect the performance of VMs in other groups. This overhead isolation lets a customer control the trade-off between deduplication overheads and additional available memory on the basis of its workloads.

Per-group deduplication thread exposes interfaces in order for a policy daemon to control a scan rate and to query the amount of surplus memory. A scan rate means how many pages are inspected during a certain period (e.g., pages/ms). The amount of surplus memory indicates how many pages are reclaimed via memory deduplication; it is decreased once a deduplicated page is copy-on-written. This information guides the policy daemon to the decision of the amount of pages to be re provisioned or used as a secondary cache. Fig. 3 shows the overall architecture of the group-based memory oversubscription scheme. The next subsection explains our group-based memory oversubscription policies.

4.2. Memory oversubscription policy

We present three memory oversubscription policies in order to improve memory efficiency on a group basis. As mentioned earlier, group-based memory management allows memory oversubscription to be adapted to the need of each group. The group-based policies control memory oversubscription by monitoring dynamic memory demands and resource usage.

4.2.1. Adaptive scan rate control

A scan rate is the main parameter that determines the behavior of a deduplication thread. A high scan rate induces CPU overheads by amplifying compute-intensive comparison operations; the CPU overheads include direct interference (e.g., scheduling delay) and indirect interference (e.g., CPU cache pollution). In order to alleviate these negative effects, commodity hypervisors conservatively set a low scan rate so that deduplication progresses slowly. Such a slow scan rate, on the other hand, reduces the opportunities of reclaiming redundant pages by deferring deduplication. In this case, since the rate at which surplus memory is procured and re provisioned is also slow, memory efficiency can be degraded when memory demand is high. Our scan rate controller takes the trade-off between CPU overheads and memory demands into account to figure out an effective scan rate.

Algorithm 1 shows how our adaptive scan rate controller performs; GetUtil, GetSwap, and GetDedup return CPU utilization, the number of pages swapped in and out, and the number of pages saved by memory deduplication during a given period of time, respectively. The scan rate control is basically a feedback-driven algorithm similar to TCP congestion control with additive-increase/multiplicative-decrease. Specifically, the scan rate controller first inspects the current CPU utilization. If the current CPU utilization is sufficiently low (below a predefined threshold), a scan rate is increased for fast memory reclamation (line 4–5). This policy lets a scan rate be retained as high as possible while idle CPU is available.

If the CPU utilization is higher than the threshold, the controller examines whether the amount of currently allocated memory is sufficient. To this end, the controller monitors swap activity of VMs for each group. If the current swap activity is negligible, the controller decreases a scan rate to ease CPU contention, assuming that additional memory is not required at this time (line 7–8).

If both high CPU utilization and memory demands are detected (line 9), the controller finally checks recent deduplication rates, which mean how many pages are reclaimed lately. Since memory insufficiency exists, it cannot be resolved by unproductive deduplication that does not
dispense reclaimed memory. In this regard, if recent deduplication rate is higher than a predefined threshold, the controller increases a scan rate with the expectation that a large amount of identical pages are rapidly reclaimed and reprovisioned to memory-hungry VMs (line 10–11). If a deduplication rate is less than or equal to zero, a scan rate is decreased for unproductive scanning not to interfere CPU-intensive VMs; a negative deduplication rate means that shared pages are being copy-on-write broken.

The algorithm is designed to prefer the resolution of memory insufficiency to that of CPU contention. Note that the final decision of increasing a scan rate (line 10–11) is made in order to rapidly create surplus memory while tolerating possible CPU contention. The rationale behind this policy is that the performance degradation caused by memory insufficiency is typically much higher than that of CPU contention, since expensive disk I/O is accompanied by swap activity. With this general knowledge, the controller makes an effort to resolve memory insufficiency if it has the conviction of productive deduplication.

4.2.2. Demand-based memory reprovisioning

Once surplus memory is reclaimed by the hypervisor via memory deduplication, it can be reprovisioned to existing VMs for effective memory utilization. For the group-based isolation, surplus memory can be reprovisioned to the only group from which the memory is derived, since our scheme complies with resource-level isolation specified by group SLAs. The isolation of memory reprovisioning can also be achieved by the entitlement-based policy [27]. In this policy, surplus memory is given to a VM in proportion to its sharing entitlement, which indicates how many surplus memory pages the VM contributes. Although this policy is reasonable if each individual VM belongs to an independent customer, it is too strict when multiple VMs are cooperative within a group. In this respect, the entitlement-based policy could degrade memory efficiency within a group since sharing entitlement cannot be directly correlated with the need of memory.

Our group-based memory reprovisioning aims at efficiency within a group, while strict isolation is guaranteed between different groups. To this end, per-group surplus memory is reprovisioned to the VMs of the group in a demand-based fashion. As with the scan rate controller, we regard swap activity as an indicator of memory demands. Once swap activity is detected and per-group surplus memory is available, the reprovisioning daemon provides additional memory to the VM that suffers from memory insufficiency; a certain amount of memory is provided periodically in response to swap activity. Since a copy-on-write re-requests a reclaimed page to the hypervisor, a group can have deficit memory; for instance, a copy-on-write happens at the time when all surplus memory has been already reprovisioned. When this situation occurs, an appropriate amount of reprovisioned memory is returned to the hypervisor for covering the deficit.

4.2.3. Hypervisor-level secondary cache

Our demand-based reprovisioning scheme cannot recognize the memory demands that are not implied by swap activity. Commodity OSes maintain disk caches that store disk blocks recently loaded in memory to reduce future disk I/O. The eviction of disk cache is performed without swap activity by discarding the page if it is clean or has already been synchronized with the disk; even dirty block eviction is recognized as a normal disk write. Previous research proposed ghost buffer techniques to estimate disk working set by means of gray-box knowledge [18] or OS modification [23]. An alternative approach is hypervisor-level secondary caches [24,20], which store the pages evicted by guest OSes at the hypervisor without tracking working set. We chose the latter approach to cover memory insufficiency of file-intensive workloads due to its simple adoption with small engineering cost; the difference between the former and latter approaches is beyond the scope of this paper.

To further improve memory efficiency for file-intensive workloads, we allow per-group surplus memory to be used as hypervisor-level secondary caches. This scheme enables the surplus memory that has not been reprovisioned to absorb the overwhelmed disk working set so that evicted disk caches are given a second chance to reside in the memory. Since evicted disk caches are guaranteed to be either clean or synchronized with the disk, they can be simply discarded when expelled from the secondary caches. The eviction from the secondary caches takes place when the surplus memory is shrunk or cannot afford to store all evicted pages.

5. Implementation

We implemented the proposed scheme on KVM-based prototype of Linux kernel 3.1.0. KVM [22] is a Type-II (i.e., hosted) hypervisor implemented as a Linux kernel module. In the KVM-based virtualization, a guest VM is running in a process, within which a virtual CPU (vCPU) operates as a thread. Linux provides a thread-based grouping method, called cgroup [25], to manage resources with the granularity of thread groups. Linux exports a unified grouping interface via virtual file system for a user to specify a group, while implementing various subsystems for each class of resources such as CPU time, memory capacity, and so forth. Since KVM encapsulates a VM within a process, all threads that belong to the VM can be grouped for resource isolation.

For group-based memory deduplication, we implemented a cgroup subsystem on KSM [1], which is the Linux component of content-based memory deduplication. The original KSM carries out all the operations in a system-wide manner while a single thread, called ksmd, periodically deduplicating the system memory registered by any VMs. Our extension enables KSM to run multiple ksmds, each of which is designated to a user-specified group, so that a deduplication thread only involves an isolated memory region. Each ksmd is grouped with its corresponding VMs’ threads in the same CPU cgroup, so that its computation overhead is isolated. For the isolation of deduplication-related data, system-wide data structures are divided into per-group ones to which cgroup interface is linked. The cgroup interface offers knobs to control and query deduplication operations and status for each group.

Our group-based policies were implemented in user-level daemons based on the information gathered via the cgroup interface (i.e., scan rates and the number of reclaimed pages) and the Linux proc interface (e.g., CPU usage and swap activities). Memory adjustment of exiting VMs was conducted by using a KVM balloon driver through libvirt.

Finally, we used Transcendent memory (Tmem) [24] as per-group secondary caches. Tmem provides guest OSes with well-defined APIs to convey an evicted page to the hypervisor; the Linux kernel adopts the Tmem support in the mainline. Tmem employs LRU policy to maintain the pages evicted by guest OSes. In order to adjust the size of Tmem according to per-group surplus memory, we modified the Tmem for KVM\(^2\) to support dynamic capacity. Since the original version of Tmem for KVM is designed to compress evicted pages only if its compression ratio is high, we also modified it to store the plain pages without compression.

6. Evaluation

Our prototype is installed on a Dell PowerEdge 710 server machine equipped with two Intel Xeon E5607 quad-core processors and 16GB of RAM. In order to evaluate the effectiveness of our scheme, we used the following workloads consisting of MapReduce jobs, MPI-based benchmarks, and a file I/O intensive workload.

\(^2\) https://github.com/akshaykarle/kvm-tmem.
memory demands. Regarding the unit of memory reprovisioning, a larger unit can quickly resolve memory insufficiency, but might lead to overprovisioning if memory demand is transient, and vice versa.

6.1. Control parameters

Our group-based policies have several parameters to which the performance is sensitive. Among the parameters, the control period of the policy daemon and the unit of memory re-provisioning are main factors to affect the performance. The control period determines how frequently a scan rate is adapted and memory is re-provisioned, while the unit of memory re-provisioning is how much memory is supplied in every period. If the period is too short, it entails CPU overheads due to fine-grained monitoring. Otherwise, the policy daemon cannot immediately follow the recent resource usage and demands. Regarding the unit of memory re-provisioning, a larger unit can quickly resolve memory insufficiency, but might lead to overprovisioning if memory demand is transient, and vice versa.

Fig. 4 shows the performance effect as the parameters are changed. We ran MapReduce PI with four VMs with insufficient memory and measured average execution time. As the control period increases, the performance is degraded because the scan rate control and memory re-provisioning cannot quickly adapt varying workload behaviors. In the case of 0.1 s period, the performance is also slightly degraded due to CPU overheads. We chose 1 s period as a reasonable parameter for reactive policy with negligible CPU overheads (1% CPU usage). For the re-provisioning unit, Fig. 4(b) shows the performance as the unit increases with the period of 1 s. As shown in the figure, a small unit around 1 MB results in lower performance due to the slow resolution of memory insufficiency. We selected 10 MB as the unit of re-provisioning, since it shows reasonable performance with less possibility of over-provisioning. We omit the results of the other workloads because they show the similar trends to PI.

For adaptive scan rate control, we empirically decided the threshold values of CPU utilization, swap activity, and deduplication rate. First, we set the threshold of CPU utilization to 90% of total CPU capacity available to each group. Since a deduplication thread has lower priority than normal vCPUs by default (nice 5 in Linux), such an aggressive threshold is reasonable to detect CPU contention. Second, the threshold of swap activity is set to zero, which means that the presence of swap operations is deemed memory demands, since the policy favors the resolution of memory insufficiency. Third, the threshold of deduplication rate is set to the minimum unit of ballooning, 1 MB, multiplied by the number of VMs in a group. This value means the amount of memory that can be re-provisioned to every VM by ballooning. Finally, we configured the minimum scan rate as the default value of KSM (5 pages/ms) and the maximum scan rate as 10x of the minimum scan rate (50 pages/ms). The adjustment unit of the scan rate is 5 pages/ms, which makes nine steps from the minimum to maximum rates.

6.2. Intra-group efficiency

In virtualized clouds (i.e., IaaS clouds), a customer typically rents a set of VMs for a parallel and distributed job in order to utilize unlimited computing resources without physically purchasing them. In this respect, we evaluated single workload scenarios using MapReduce jobs and the set of NPB benchmarks. Along with the single workload evaluation, it is essential to evaluate a scenario when a group is consisted of VMs having different resource demands. For example, when a customer leases tens of VMs to build own virtual clusters, different types of workloads can be assigned to the cluster. Meanwhile, the cloud provider could consolidate those VMs, which belong to the same group, in the same physical host based on their consolidation policy. In this section, therefore, we examine impacts of our proposed policies on intra-group efficiency using both single and mixed workload scenarios.
We configured a VM with one vCPU and 512 MB of memory, and then we used four VMs and eight VMs for single workloads and mixed workloads, respectively; all VMs are colocated in the set of four physical cores. In MapReduce workloads, one VM works as a master node and the other VMs work as slave nodes. In NPB workloads, the configuration is similar except that the master VM in NPB also carries out the same amount of work as slave VMs. In the following experiments, we ran all the workloads three times, and then we used normalized execution time as the performance metric of each workload.

6.2.1. Adaptive scan rate control

We first evaluated the following control algorithms in order to show the impact of each control factor on the workloads’ performance. Adaptive-C adjusts a scan rate based only on CPU utilization. Adaptive-CS additionally considers the presence of memory demands (i.e., swap activity) even though CPU utilization is high. Adaptive-CSD controls a scan rate based on CPU utilization, swap activity, and deduplication rate (Algorithm 1). We used the demand-based policy for surplus memory redistribution except for the baseline.

Fig. 5 shows normalized execution time of MapReduce and NPB workloads with various scan rate controllers; Static-X denotes that its scan rate is statically set to X pages per milliseconds. In MapReduce workloads (Fig. 5(a)), the performance is dramatically improved when surplus memory is distributed even though the fixed scan rate is minimal (Static-5) because of the insufficient memory configuration. Since the group does not fully utilize the provisioned CPU resources, the three adaptive controllers show similar performance to Static-50.

In most of NPB benchmarks (Fig. 5(b)), on the other hand, the performance is decreased as a scan rate is increased. In these workloads, each VM fully utilizes its vCPU because it has sufficient memory to cover its working set. Hence, the deduplication procedure constantly disturbs the workloads when the scan rate is fixed (Static-X). The adaptive controllers, on the contrary, decrease their scan rates in response to the high CPU utilization. Note that the performance of NPB DT is improved as a scan rate is increased because DT shows both high CPU utilization and frequent swap activities simultaneously. Accordingly, Adaptive-C shows no performance improvement since it keeps the scan rate as low as possible due to the high CPU utilization. Both Adaptive-CS and Adaptive-CSD achieve about 10% performance improvement compared to the Adaptive-C case because they increase their scan rates in response to the frequent swap activities. Based on the results, it is imperative to observe not only CPU utilization but also memory demand of a group for intra-group memory efficiency.

The different behavior among the three adaptive algorithms is shown clearly in the mixed workloads. For mixed workloads, we chose a set of workloads consisting of the MapReduce workloads and the NPB CG benchmark (MapReduce + NPB CG) because these combinations show mixed characteristics in a group clearly; MapReduce shows high memory demand while NPB CG runs a CPU-intensive job without memory insufficiency. We used the geometric mean of the normalized execution times as the metric of a group performance for the mixed workloads.

Fig. 6 presents normalized execution time of MapReduce + NPB CG with the three adaptive controllers. Adaptive-CS achieves better performance than Adaptive-C in MapReduce workloads (Fig. 6(a)). The performance of NPB CG, however, is degraded when using Adaptive-CS as compared to the Adaptive-C case (Fig. 6(b)). Adaptive-CSD additionally improves the performance of NPB CG compared to Adaptive-CS (up to 6%). As a result, Adaptive-CSD improves the overall performance up to 40% and 19% compared to the baseline and Adaptive-C, respectively, because of the significant performance gain in MapReduce workloads (Fig. 6(c)).

Fig. 7 can be used to explain the main reason for the above result. Adaptive-C reduces the scan rate as soon as it detects high CPU utilization from NPB CG, and gradually increases after termination of NPB CG. On the other hand, Adaptive-CS increases the scan rate in response to the swap activities from MapReduce SORT. Adaptive-CSD shows similar behavior to Adaptive-CS until
the midpoint of the workload execution, and then it decreases the scan rate when there is little chance to further deduplications. Though Adaptive-CSD shows insignificant performance gain compared to Adaptive-CS in MapReduce + NPB CG, the performance difference between the two algorithms would be more significant in pure CPU-bound workloads. In addition, the reduced number of CPU cycles in Adaptive-CSD can help to save money in a cloud which charges by the number of CPU cycles a user consumes. By using Adaptive-CSD, therefore, a memory-hungry VM can be favored while minimizing unnecessary CPU consumptions.

6.2.2. Demand-based memory reprovisioning

Next, we evaluated MapReduce + NPB CG using Adaptive-CSD in combination with various reprovisioning policies. In addition to our demand-based policy, we evaluated two alternatives: fair- and entitlement-based policies. The fair-based policy reprovisions surplus memory evenly to VMs without considering demands and contributions (i.e., communism). The entitlement-based policy provides VMs with surplus memory in proportion to their contribution to the reclamation of the memory (i.e., capitalism). Finally, the demand-based policy reprovisions surplus memory only to memory-hungry VMs (i.e., utilitarianism).

As we can see in Fig. 8, the demand-based policy improves the performance of MapReduce workloads. Especially, SORT shows about 20% and 11% improvement compared to the fair- and entitlement-based policy, respectively. Since the performance of NPB CG is bounded to the CPU resource in our configuration, the overall performance of the MapReduce + NPB CG workloads could be improved with the demand-based policy.
In order to further analyze the behavior of each policy, we traced the total amounts of memory allocated to the two sets of VMs which run SORT and NPBCG each. From Fig. 9, we can see that all the reclaimed memory is distributed to the SORT VMs when the demand-based policy is applied. This intensive reprovisioning to the memory-hungry VMs could make performance improvement as compared to the fair- and entitlement-based policy. In addition, the heterogeneity of memory demand is existed within SORT (Fig. 10). Specifically, the master VM does not take additional memory because it has lower memory requirement than the slave VMs.

6.2.3. Hypervisor-level secondary cache

Observing swap activity, however, is not sufficient to detect memory demand of a VM which does not exhibit frequent swap activities though additional memory could help to improve its performance. In this regard, the hypervisor-level secondary cache can be used to mitigate the limitation of the demand-based reprovisioning policy.

In order to validate the effectiveness of the secondary cache, we additionally selected a mixed workload consisting of PI and FIO. We used four VMs with one vCPU and 1 GB memory each for PI, and one VM with one vCPU and 512 MB memory for FIO. The FIO VM ran concurrently with PI VMs for 3 min and we measured the average throughput over the three runs.

Table 2 shows the average throughput of FIO using various memory oversubscription policies. Both the entitlement- and demand-based policies get little benefit from the group’s surplus memory. In the entitlement-based policy, the FIO VM takes only little portion of the total surplus memory because it contributes little. The demand-based policy cannot reprovision the large portion of the surplus memory to the FIO VM because it performs the intensive file I/O workload. On the other hand, by using surplus memory
Table 2

<table>
<thead>
<tr>
<th>Oversubscription policy</th>
<th>AVG throughput (KB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>2301.61</td>
</tr>
<tr>
<td>Fair-based</td>
<td>2813.44 (22% improvement)</td>
</tr>
<tr>
<td>Entitlement-based</td>
<td>2365.92 (3% improvement)</td>
</tr>
<tr>
<td>Demand-based</td>
<td>2444.97 (6% improvement)</td>
</tr>
<tr>
<td>Demand-based + secondary cache</td>
<td>3030.12 (32% improvement)</td>
</tr>
</tbody>
</table>

6.3. Inter-group isolation

Besides intra-group efficiency, guaranteeing inter-group isolation is a crucial factor for our group-based scheme. Therefore, we evaluated multiple-group scenarios where two groups are colocated in a physical machine and independently run their own workloads.

In the following experiments, a group is comprised of four VMs each with one vCPU and 512 MB of memory. For comparison, we used the same policy (Section 4.2) for both the system-wide and group-based oversubscription. We assume that a group SLA is established for the memory leased by a group not to be shared by and provisioned to another group.

The first scenario we evaluated is the worst case of group SLA violation by the system-wide oversubscription, in which a memory-hungry group aggressively extorts surplus memory from another group. To simulate this scenario, we ran a synthetic workload in a group that intentionally imposes memory pressure causing swap activity while producing little amount of surplus memory; the memory-hungry program (MemHog) repeatedly writes random numbers to a large amount of memory in each VM. In the other group, we ran MapReduce workloads generating a large amount of surplus memory.

Fig. 11 shows the result of MapReduce workloads and the total amount of memory allocated to each group, for the system-wide (NOGRP) and group-based (GRP) oversubscription. For all MapReduce workloads (Fig. 11(a)), the group-based scheme achieves higher performance than the system-wide one by up to 40%, since the surplus memory of the MapReduce group is used only for itself to relieve its memory insufficiency. The surplus memory created by the SORT group is deprived by the MemHog group in the case of NOGRP.

We additionally ran NPB DT instead of MemHog (Fig. 12). Since NPB DT shows considerable swap activity in its initial phase of execution, the performance of MapReduce workloads in NOGRP is worse than that of GRP’s, and vice versa for NPB DT. As a result, GRP improves system-wide overall performance by up to 20% compared to NOGRP. Though group SLA violation in the system-wide scheme would be avoided by using the entitlement-based policy, this policy hurts system-wide efficiency, which is the most critical factor for a system-wide scheme, because it does not consider the memory demand of each VM.

We next evaluate another multiple-group scenario where each group shows different resource demands. We chose MapReduce...
workloads for a memory-hungry group, and NPB SP benchmark for a CPU-intensive group. In this evaluation, we focus on the performance of the MapReduce group since performance of the NPB SP group is not affected by memory oversubscription. In contrast to the former scenarios, the performance of MapReduce workloads is decreased when GRP is applied (Fig. 13(a)). Since the NPB SP group has sufficient memory, all of the surplus memory from the group is distributed to the MapReduce group in the NOGRP case (Fig. 13(b)). As a result, NOGRP achieves better performance of the MapReduce group without any performance impact on the NPB SP group.

This scenario implies that the group-based oversubscription would degrade system-wide memory efficiency at the expense of inter-group isolation. With a system-wide scheme, however, it is possible that a malicious group can get a large portion of total surplus memory even though it does not suffer from memory insufficiency, by intentionally inflating its memory demand. Hence, a group’s surplus memory should not be distributed to another group without a reasonable inter-group policy.

7. Utilization of idle surplus memory

From the evaluation, we found that idle surplus memory (i.e., underutilized surplus memory) from a group can be used to enhance another group’s performance. In order to fully utilize idle surplus memory in a system, inter-group redistribution of idle surplus memory can be used as a complementary policy for our group-based scheme. Using idle surplus memory across different groups, however, can sacrifice isolation functionality. Hence, in order to preserve inter-group isolation, provided surplus memory should be returned immediately as soon as its owner needs it. In addition, memory deduplication overheads should be compensated for the owner of surplus memory.

7.1. Demand-based overhead migration

To deal with the issue on idle surplus memory, we can use a proportional share-based policy comprising a compensation mechanism. In this policy, a group’s contribution to the total amount of surplus memory in a system can be expressed as a share of that group. When idle surplus memory exists, a group which demands more memory beyond its share can additionally exploit that memory in a work-conserving manner. For the compensation mechanism, similar to [38], compensation shares can be given to a group who provides idle surplus memory to another group, in proportion to both the amount of memory and leased duration. Then, the compensation shares can be used later to get more memory beyond the group’s contribution to the total amount of surplus memory in the system. The proportional share-based policy, however, cannot easily be integrated into our group-based scheme, since deciding on proper compensation rate and quantifying the benefit from getting more memory as compensation are difficult in practice.

Alternatively, we can use a surplus memory economic model that considers the surplus memory from memory deduplication as a financial asset of each group. In this model, idle surplus memory from a group can be loaned to other groups that are willing to pay rent for resolving its memory insufficiency. As a means of compensation, we can use monetary fund from a debtor group to pay an appropriate amount of money as a rental fee to the creditor group in proportion to the amount and the duration of borrowed memory. By using monetary funds, the customer owning the creditor group earns money by renting its idle surplus memory, and only the customer who has available funds can borrow memory according to its requirement. However, the economic model needs complex auction mechanism to control the price of surplus memory as in the previous work [37] and frequent customer-involvements to decide the amount of monetary funds.

To inherently eliminate the need for a complex compensation mechanism, therefore, we introduce a demand-based overhead migration as a complementary policy for our group-based scheme. This policy basically imposes deduplication-related overhead on a corresponding group. When there is a group demanding additional memory beyond its allocation (demanding group) and if a group, which has idle surplus memory (idling group), is found, the memory deduplication overhead starts to be accounted to the demanding group. After migrating the deduplication overhead, the demanding group can exploit the idle surplus memory to resolve its memory insufficiency with the expense of CPU overhead from memory deduplication. Then, exploited idle surplus memory is revoked when the idling group needs additional memory as the memory demand of the running workload is changed. Finally, the overhead accounting is migrated back to the idling group when there is no more idle surplus memory in the idling group.

By adding the demand-based overhead migration to our group-based scheme, idle surplus memory can be effectively utilized for improving system-wide memory efficiency without compromising inter-group isolation. In addition, the demand-based overhead migration has two additional benefits for a group providing its idle surplus memory to another group. First, identical pages can be rapidly deduplicated even if a workload in the idling group has no CPU margin for memory deduplication, because this CPU overhead is imposed to the demanding group. Accordingly, if a CPU-intensive workload in the idling group turns out to be memory-hungry, already deduplicated memory can be quickly reused to satisfy the memory insufficiency without requiring the time-consuming deduplication phase. Second, the CPU cache footprint of the workload in the idling group can be reduced by merging identical pages into a single page without causing additional CPU contention. Consequently, the performance of a cache-sensitive workload in the idling group would be improved by reducing the number of cache misses in shared CPU caches of multi-core processors.
7.2. Experimental results

In order to validate the effectiveness of the demand-based overhead accounting, we evaluated the added feature for the overhead migration (denoted as GRP-M) to the policy daemon using two additional multiple-group scenarios. Specifically, we modified the original policy daemon to dynamically migrate an idling group’s ksmd to a demanding group’s resource pool by using the group interface. In addition, the adaptive scan rate controller of the idling group monitors both CPU utilization and swap rate of the demanding group to control the ksmd until there is idle surplus memory in the idling group. In the following experiments, we used the identical configuration in Section 6.3.

In the first scenario, we consecutively ran NPB DT (memory-hungry) after the execution of NPB BT (CPU-intensive) in the NPB BT-DT group, and we ran MapReduce wordcount (memory-hungry) in the WC group. Fig. 14 shows average execution time normalized to GRP for each group and geometric mean for the overall system-wide performance; the performance of NPB BT in the NPB BT-DT group is omitted because the performance is not affected by memory oversubscription. Both GRP-M and NOGRP enhance the performance of WC compared to the GRP case, by 28% and 32%, respectively. For NPB DT, GRP-M reduces the average execution time compared to the GRP case by 15%, whereas NOGRP results in the degraded performance by 27%. Overall, GRP-M improved the system-wide performance by 22% and 15% compared to GRP and NOGRP, respectively.

GRP-M achieved the performance improvement of WC by enabling the utilization of idle surplus memory from the NPB BT-DT group during the execution of NPB BT (Fig. 15(b)). The reason behind the performance improvement of NPB DT is that GRP-M rapidly reprovisions surplus memory to the NPB BT-DT group in response to the frequent swap activity in the initial phase of NPB DT, after the termination of NPB BT (Fig. 15(a)). On the other hand, GRP could not reprovision surplus memory as fast as GRP-M did because the scan rate of ksmd was throttled during the execution of NPB BT in order to minimize the CPU interference from memory deduplication. Moreover, NOGRP did not have enough surplus memory to be given to the NPB BT-DT group, because the surplus memory in the system is deprived by the WC group already. By migrating the deduplication overhead from the idling group to the demanding group, system-wide memory utilization could be improved while reasonably compensating to the idling group.

Next, we evaluated the second scenario where four instances of SPHINX3, which is included in the SPEC CPU2006 benchmark suite, are run in the first group and MapReduce SORT is run in the second group. Fig. 16(a) and (b) show the average execution time normalized to GRP and group memory changes during the execution in SORT group, respectively. Similar to the first scenario, the performance of the SORT group is improved in both GRP-M and NOGRP by exploiting the idle surplus memory from the SPHINX3 group. The performance of SPHINX3 is improved by 30% in GRP-M and by 32% in NOGRP, even though the SPHINX3 did not suffer from memory insufficiency.

In order to further investigate the reason behind the performance improvement, we recorded the number of cache misses in the last-level CPU cache of SPHINX3. Table 3 shows the number of load and store misses as the policy changes. As shown in the table, the number of total cache miss is significantly reduced by 62% when either GRP-M or NOGRP is applied. This is because deduplicating identical pages made the working set of SPHINX3 to fit into the last-level CPU cache by reducing the cache footprint of the SPHINX3 group. By using GRP-M, both SPHINX3 group and SORT group could gain performance benefit from idle surplus memory.

![Fig. 14. Normalized execution time of NPB DT group and WC group.](image_url)

### Table 3

<table>
<thead>
<tr>
<th>Policy</th>
<th># Load miss</th>
<th># Store miss</th>
<th># Total</th>
<th>Reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRP</td>
<td>68 359</td>
<td>6279</td>
<td>74 639</td>
<td>—</td>
</tr>
<tr>
<td>GRP-M</td>
<td>24 634</td>
<td>30 63</td>
<td>27 698</td>
<td>62.89</td>
</tr>
<tr>
<td>NOGRP</td>
<td>24 720</td>
<td>30 71</td>
<td>27 792</td>
<td>62.76</td>
</tr>
</tbody>
</table>

8. Discussion

This section discusses various applicability of the group-based memory oversubscription.

8.1. Group SLAs

The group-based memory oversubscription enables several group SLAs for memory resources. For strong isolation, a group should cover the only VMs that are leased by an individual customer. However, not all customers need strict level of isolation. For example, a group can encompass the VMs of different customers who share their data for cooperative services. CloudViews [9] illustrates the growing demands of rich data sharing among cloud-based Web services.

Virtual appliance-based grouping can also be an option to customers, since the same virtual appliances are likely to produce a large amount of surplus memory due to the identical reference image [28]. A customer can request this type of group SLA by allowing surplus memory to be flexibly used by the VMs with the same virtual appliance.

Dynamic grouping based on sharing opportunities is one of the ways for increasing the amount of surplus memory by colocalizing VMs having similar memory fingerprints in the same group but in different physical machines. The idea of tracking memory fingerprints and finding similarity can be borrowed from the previous work [43]. In addition, dynamic grouping for filtering memory hog VMs out to the isolated group can be a good option when the memory hog VMs deprive the surplus memory within a group. However, since arbitrary grouping by a provider is not permitted in public clouds, memory tracking and grouping must be conducted by means of SLAs specified by customers.

8.2. VM colocation

In order for our scheme to be more memory-efficient beyond isolation, multiple VMs that belong to the same group should be colocated in a physical node. As novel hardware (e.g., multi-core processors and direct I/O support) has been improving the scalability of consolidation in data centers, the likelihood of colocating VMs that belong to the same group is expected to increase. In addition, colocating VMs that communicate each other can improve the

---

network performance by avoiding the expensive cost of external network links [31, 32, 41]. Since VMs within a group tend to communicate each other especially for distributed computing, colocation of such VMs can improve network resource efficiency.

Even in a case where a single VM of a group is solely located in a physical machine, memory deduplication is still effective due to intra-VM memory sharing. Barker et al. [2] showed that self-sharing accounts for considerable amount of sharing potential in real world scenarios. In addition, a large amount of duplicate pages appear in some scientific workloads [22]. Nevertheless, our group-based memory oversubscription is not limited to local memory resources. Although surplus memory is reclaimed within each physical machine, it can be globally used regardless of VM colocation. Network memory allows a VM to use the surplus memory in a remote machine [40] and has known to be faster than local disk I/O, with the aid of fast network in data centers [7,8]. Accordingly, per-node surplus memory can be aggregated as a cloud-wide group memory by the network memory mechanism.

In order to show the potential advantage of the aggregated group memory, we enable the VMs in a local machine to use the remote surplus memory as a swap space. We ran MapReduce workloads in the local node with our oversubscription policies. In this setup, inevitable swap requests in spite of locally replicated memory take advantage of the remote swap avoiding disk I/O. Fig. 17 shows the performance improvement (8%–30%) by the remote group swap. As a result of this experiment, network memory is an effective solution to further enhance memory efficiency.

8.3. Application to other platforms

Though we focus on IaaS model in this paper, our group-based memory oversubscription can be applied to other cloud service models such as Platform-as-a-Service (PaaS) model. As an example, a PaaS service provider can build upon IaaS by applying specific services or configurations to a set of tenant VMs. In this case, VMs hosting PaaS are allocated to independent customers in order to provide specified services. Hence, if the IaaS service provider offers an interface for classifying internal groups within a group, it would be better for the PaaS service provider to dynamically reclassify its allocated VMs into a set of internal groups based on customer usage in order to apply the group-based memory oversubscription for each customer using PaaS. Another example is the VM-based cloud service for mobile devices where heavy computation is offloaded to the cloud connected via high bandwidth wireless network [29,6]. In this scenario, a set of VMs can be allocated to a user for servicing the user’s various mobile devices (e.g., smartphone and tablet PC). Therefore, our group-based memory oversubscription can be exploited in order to provide isolation between independent users’ VMs located in a single physical machine.

Our group-based scheme also can be used for different task models such as Directed Acyclic Graph (DAG) model. MapReduce is a good example of DAG task model because it is a special form of
DAG that is applicable in a wide range of use cases. In a MapReduce job, reduce tasks wait for output from map tasks, and in turn, map tasks of the next round wait for intermediate output from reduce tasks of the previous round. Since memory demand varies over time depending on the phase (i.e., map or reduce) of a MapReduce job [10], our adaptive scan rate controller can help to adjust the scan rate according to the memory demand for each node in a cluster, in order to secure additional memory quickly while minimizing interference from memory deduplication. In addition, if several MapReduce jobs are concurrently executed on a virtualized cluster, each VM in the virtualized cluster would show different memory demand because different types of tasks can be run at the same time. Hence, in this case, our memory oversubscription policies can play a role in improving overall performance within the boundary of the virtualized cluster.

9. Related work

This work presents various group-based memory oversubscription policies by extending our preliminary prototype [21]. This section describes previous work on memory oversubscription, resource consolidation, and isolation.

9.1. Memory oversubscription

Memory deduplication was first introduced by VMware ESX server [36]. Then, a large volume of studies has been presented especially for further improvement of deduplication efficiency [35, 14,27]. In addition, Wood et al. [43] presented content-based VM colocation in order to take advantage of memory deduplication. Dynamic memory balancing has been proposed in two approaches: explicit and implicit balancing. The explicit approach allows the hypervisor to directly adjust VM memory based on memory demands. The ghost buffer mechanism [18,23] and swap activity monitoring [45] have been employed to estimate effective memory utilization. Considering that the explicit approach is effective only with accurate working set estimation, the implicit approach globally maintains surplus memory pool to absorb overwhelmed working set at the hypervisor [24,20].

9.2. Resource consolidation and isolation

Resource consolidation is a well-known technique for dynamically reducing the number of nodes in a cluster, by liberating nodes that are not needed for the current computation phase. Since first proposed by Walsh et al. [39], a general two-tier architecture that uses utility functions has been commonly adopted in the context of autonomous resource consolidation management [5,3,34]. For example, the Unity project [5] has adopted this approach to allow complex computing systems to manage themselves to be self-configuring, self-optimizing, self-protecting, and self-healing. On the other hand, Yazir et al. [44] proposed the IMPROMPTU model to address the scalability and feasibility issues arising from the previous studies, by adopting a distributed approach that uses Multiple Criteria Decision Analysis (MCDA) during configuration processes.

Similar to the traditional cluster environments, resource consolidation schemes are widely explored for virtualized cluster environments based on the fact that it is convenient to host each task in a VM in order to make consolidation transparent. Hermenier et al. [15] have proposed the Entropy resource manager, which performs dynamic consolidation based on constraint programming and take migration overhead into account. Recently, a resource consolidation framework proposed for the scenario where multiple clusters of VMs are hosted in a cloud [16]. In this work, VMs belonging to a virtual cluster are treated as moldable entities rather than rigid entities, because QoS is delivered by a virtual cluster as a single entity in the targeted environment.

On the other side, resource management schemes that reduce performance interference between VMs have been proposed. Gupta et al. [13] enforced isolation support of CPU scheduler while Nathuji et al. [26] and Govindan et al. [12] focused on LLC interference among consolidated VMs. In addition, Shieh et al. [30] proposed the hypervisor-based rate controller for network performance isolation while Keller et al. [19] presented a new architecture that partitions all hardware components that can be shared by VMs in order to achieve strict isolation.

Several researchers have presented QoS-conscious memory oversubscription in clouds. Gordon et al. [11] proposed application-driven memory oversubscription based on the fact that applications know the best knowledge about memory demands. Williams et al. [40] introduced memory overload migration to alleviate performance degradation caused by oversubscription. None of the proposals did consider memory deduplication and group-based policy.

10. Conclusion and future work

This paper introduces group-based isolation to existing VM memory management schemes for multi-tenant cloud environments. As many service providers have been considering IaaS clouds to be their cost-effective infrastructure, group SLAs are reasonable offering for both inter-group isolation and intra-group efficiency. The group-based memory management allows a customer or cooperative customers to efficiently oversubscribe the leased memory resources in an isolated domain guaranteed by a cloud provider. The strict isolation of memory resources between independent customers eliminates known security breaches and unexpected performance degradation. Using per-group management policies, isolated memory can be controlled by a customer, not dictated by a provider. Strong memory isolation sometimes comes at a cost of memory wastage when multiple groups have different memory demands. Our demand-based overhead migration complements this limitation by exploiting idle surplus memory in different groups without compromising isolation functionality. The result of additional experiments validates the effectiveness of the demand-based overhead migration in combination with the group-based scheme.

We plan to implement flexible management of per-group surplus memory regardless of VM colocation. We expect that cloud-wide group memory management further enhances memory efficiency by reducing underutilized surplus memory for each node.

Acknowledgments

This work was partly supported by the IT R&D program of MKE/KEIT [10041244, SmartTV 2.0 Software Platform] and by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) / No. 2012-000148.

References
