LETTER

Paging out Multiple Clusters to Improve Virtual Memory System Performance

Woo Hyun AHN[†a)], **Joon-Woo CHOI**[††], **Jaewon OH**[†††b)], ***Nonmembers***, **Seung-Ho LIM**[††††], ***and*** **Kyungbaek KIM**[†††††], ***Members***

SUMMARY Virtual memory systems page out a cluster of contiguous modified pages in virtual memory to a swap disk at one disk I/O but cannot find large clusters in applications mainly changing non-contiguous pages. Our proposal stores small clusters at one disk I/O. This decreases disk writes for paging out small clusters, thus improving page-out performance.

key words: *virtual memory system, page-out, page clustering, disk*

1. Introduction

Main memory sizes in computer systems have increased over the years. However, many scientific and engineering applications can still exhaust main memory [1], from which virtual memory (VM) systems replace VM pages to make room for new ones. When a dirty page is replaced, it is stored to swap disk (or swap) via page-out. Later, an access to the page incurs a page fault that requires a page-in fetching the page from swap. Hence, serious memory shortage leads to excessive page-outs as well as page-ins, which degrade the VM system performance.

The VM system of 4.4BSD OS [2] uses a page clustering scheme [3] to reduce the disk I/Os required to page out pages in VM areas of bss, stack, and heap. The scheme was introduced in Digital UNIX 1.2 based on Mach OS [4] two decades ago and is currently used in Apple Mac OS [5]. When a victim page is flushed out, the scheme composes a cluster* of the victim and dirty pages under the clustering condition that they are not only in the same VM area as the victim but also adjacent to the victim in a VM space. Then, the cluster is stored to swap at one disk I/O to improve the page-out performance. Furthermore, the clustering allows a prefetching scheme to reduce page faults: if accessing a page causes a page fault, the prefetching retrieves the page and others adjacent to the page, both in a VM space and on swap, at one disk I/O. If the prefetched pages are accessed before being evicted, the VM system avoids page faults that would need disk accesses to fetch the pages.

However, there has not been any research to evaluate how the clustering scheme of BSD VM affects the VM performance, even though the scheme has been used in some commercial OSes. More importantly, the clustering has overlooked the challenge of applications [6] not having a high spatial locality of page writes. Such applications can mostly write sets of only a few pages that are contiguous with each other in a VM space. This behavior disables the scheme to compose large clusters of enough pages to reduce disk writes for page-outs, thus degrading the performance improvement of page-outs. Moreover, the scheme can compose extremely small clusters of only one page if pages not contiguous with each other are mainly modified.

In related studies, Linux uses page clustering [7] to reduce page-outs rather than page-ins. The scheme flushes a cluster of 32 pages including a victim and dirty pages that follow the victim in the access order without considering the page adjacency in a VM space. Upon a page fault, Linux prefetches seven pages physically contiguous around the faulted page on swap. However, the pages may not be accessed if their access order differs from that in which the pages were accessed before being flushed. Another study [8] compresses multiple pages into a page to keep more pages in main memory, thus enabling the flushing of the compressed page to have a similar effect as if several pages were transferred at one disk I/O. However, the approach incurs large page access time because pages should be uncompressed before being accessed.

This paper proposes a new scheme called multiple-clustering (mClustering) to speed up page-outs of small clusters in BSD VM without degrading the performance of page-ins. mClustering is integrated into the 4.4BSD kernel with small change on the VM structure in order to support a key idea: multiple clusters to be paged out are collected into a large chunk, which is flushed to swap at a single disk I/O. Such a multiple-cluster transfer decreases disk writes that result from flushing small clusters out. Further, when collecting clusters into a chunk, our scheme composes each

Manuscript received November 19, 2013.
Manuscript revised March 4, 2014.

[†]The author is with the Department of Computer Science, Kwangwoon University, Seoul, Korea.
[††]The author is with TmaxSoft Inc., Seungnam-si, Gyeonggi-do, Korea.
[†††]The author is with the School of Computer Science and Information Engineering, The Catholic University of Korea, Bucheon, Gyeonggi-do, Korea.
[††††]The author is with the Department of Digital Information Engineering, Hankuk University of Foreign Studies, Yongin-si, Gyeonggi-do, Korea.
[†††††]The author is with the Department of Electronics and Computer Engineering, Chonnam National University, Gwangju, Korea.
a) E-mail: whahn@kw.ac.kr
b) E-mail: jwoh@catholic.ac.kr (Corresponding author)
DOI: 10.1587/transinf.E97.D.1905

*The BSD VM system can store a cluster of up to 16 pages at one disk I/O.

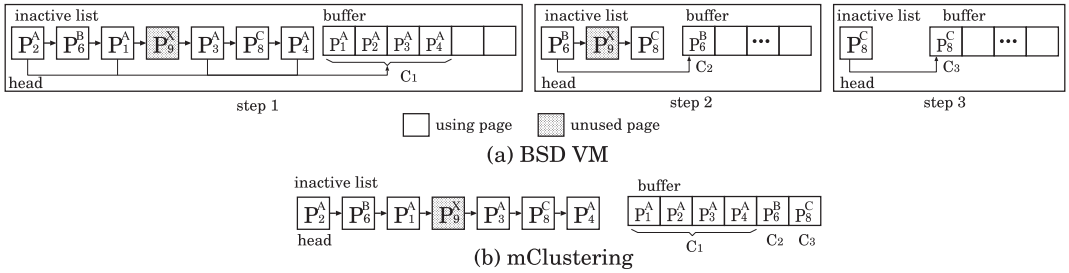


Fig. 1 The comparison of page-outs between BSD VM and mClustering. Three clusters C_1, C_2, and C_3 are composed of using pages as follows: $C_1 = \{P_1^A, P_2^A, P_3^A, P_4^A\}$, $C_2 = \{P_6^B\}$, $C_3 = \{P_8^C\}$.

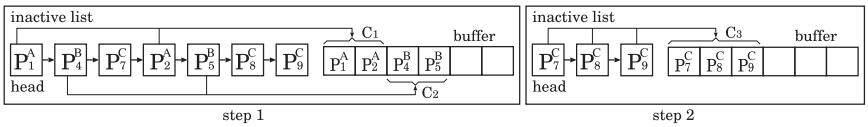


Fig. 2 Flushing clusters in a buffer that is not filled up with pages. Seven using pages compose three clusters: $C_1 = \{P_1^A, P_2^A\}$, $C_2 = \{P_4^B, P_5^B\}$, $C_3 = \{P_7^C, P_8^C, P_9^C\}$.

cluster in order to avoid making its pages positioned away from each other on swap, similarly to BSD VM. This allows the prefetching to retrieve a comparable number of pages to BSD VM on a page fault.

2. Paging out Multiple Clusters

In BSD VM, the page-out daemon runs when there are not enough pages in the free list where page faults are handled. The daemon scans from the least recently used page in the front of the inactive list that has pages not accessed for a long time. The scanning continues until a desired number of pages are evicted to the free list. When BSD VM intends to evict a using victim page from the inactive list, mClustering triggers a clustering operation with three phases: the first is to compose a cluster of the victim and using pages satisfying the clustering condition in the inactive list and to put the cluster into a buffer of 16 pages. The second is to scan from the next page of the victim in the inactive list to find another victim and to collect a cluster for the victim into the buffer. This operation continues until the buffer is either filled up with 16 pages or does not have available space for a new cluster. Finally, all pages of the buffer are stored to one of the large free spaces on swap at one disk I/O.

Figure 1 illustrates how mClustering reduces disk writes for page-outs, compared with the page clustering of BSD VM in a scenario: there are six using pages in the inactive list and a buffer has room for six pages. P_n^m represents a page that has a unique number n in a VM space and is included in a VM area m. As shown in Fig. 1 (a), when flushing victim P_2^A in the front of the list in step 1, BSD VM composes cluster C_1 of the victim and three pages adjacent to the victim in the inactive list, to store the cluster at one disk I/O. In step 2 and 3, however, victims P_6^B and P_8^C do not have any adjacent page in the list. Each of the victims composes clusters C_2 and C_3, which are flushed in two disk I/Os. Note that BSD VM moves an unused page P_9^X to the free list without flushing the page.

Figure 1 (b) shows a clustering operation of mClustering in the same scenario as Fig. 1 (a). When choosing page P_2^A as a victim to flush, the scheme puts cluster C_1 into a buffer, similarly to BSD VM. Then, C_2 and C_3 are collected into the available space of the buffer, differently from BSD VM. The three clusters collected into the buffer constitute a chunk, all pages of which are flushed to swap at one disk write. Hence, mClustering issues two fewer disk I/Os than BSD VM.

There may not be enough available room in a buffer to collect a new cluster after some clusters have been collected into the buffer, as shown in Fig. 2 where P_1^A, P_4^B and P_7^C are chosen as victims in the order. In step 1, mClustering collects clusters C_1 and C_2 for P_1^A and P_4^B into a buffer, but cluster C_3 for P_7^C cannot be done into the buffer with the available room for two pages. If only P_7^C and P_8^C are flushed along with C_1 and C_2 after being collected into the buffer, the two pages can be placed at a disk location away from P_9^C that will be flushed in a subsequent flushing. This placement disables the prefetching to retrieve the three pages at one disk I/O when one page of C_3 is faulted. To solve this problem, mClustering does not put the proper subset of C_3 into the buffer, but flushes only C_1 and C_2 that have already been put into the buffer. After the flushing, all pages of C_3 are collected into the buffer in step 2 and are stored to swap via the next flushing operation.

mClustering uses the same policy as BSD VM to decide which pages around a victim in a VM space compose a cluster. The policy traverses eight pages backward from the victim and then eight pages forward from the next page of the victim. If the traversed 16 pages meet the clustering condition, they are collected to compose a cluster; otherwise, if a page not meeting the condition is detected during either the forward or the backward traverse, the policy collects pages traversed till then and scans pages in the other direction until collecting a total of 16 adjacent pages. Nevertheless, if 16 pages are not collected, the pages collected till then compose a cluster.

When storing pages of a buffer to swap, mClustering uses a swap space allocation policy considering the multiple-cluster transfer. BSD VM attempts to allocate a contiguous free space large enough to contain pages of a cluster on swap. If not successful, it allocates the largest contiguous free space ranging from 15 pages to one page and flushes a proper subset of the cluster to the space. This flushing method is applied to the remaining pages of the cluster again, which can be stored away from the pages flushed before. In contrast, mClustering attempts to allocate a contiguous free space enough for all clusters in a buffer. If it fails, it allocates a contiguous free space to each of the clusters, which is flushed to its corresponding free space. Such an allocation prevents pages of a cluster from being separated from each other on swap. Nevertheless, if there is not any contiguous free space enough for a cluster, its pages are stored via the flushing method of BSD VM.

Algorithm 1 describes a clustering operation of mClustering implemented in the page-out routine of BSD VM, which is called when a page is chosen as a victim to be flushed in scanning the inactive list. The algorithm first composes a cluster of a victim and pages meeting the clustering condition (line 3). If a buffer has available space to contain the cluster (line 6), pages of the cluster are put into the buffer to be combined with other clusters that have been collected in the previous clustering operations. However, if the buffer does not have enough available room (line 8), the cluster is not put into the buffer, but pages of clusters collected in the buffer until then are flushed at a single disk I/O (line 9). After the flushing, the buffer is emptied to contain the cluster that could not be put into the buffer due to the lack of space (line 10–11). Finally, a buffer can contain the last using page from the inactive list (line 13). This indicates that there are no using pages to flush in the inactive list. Hence, mClustering flushes pages of the buffer to swap.

Algorithm 1 mClustering Algorithm

```
1:  // let m be the maximum number of pages that a buffer buf can contain
2:  // let p be a current victim page
3:  set cluster to a cluster of p and using pages that are in the same VM
    area with p and contiguous with p in VM space
4:  set c to the number of pages in cluster
5:  set t to the total number of pages in buf
6:  if c + t <= m then
7:      put cluster into buf
8:  else
9:      store pages of buf to disk
10:     reset buf
11:     put cluster into buf
12: end if
13: if buf includes the last using page from inactive list then
14:     do the same steps as lines 9–10
15: end if
```

3. Experimental Evaluations

3.1 Experimental Setup

Our experiments were conducted on a computer with 3.5 GHz Intel Core i7 CPU, 1GB DDR3 memory, and Seagate 160GB 7200RPM disk in FreeBSD 8.2 OS. The testing virtual memory systems were configured in the default setting: 4KB page size and 2GB swap partition, which uses 4KB block as a unit to store a page.

We evaluate the performance of mClustering and BSD VM using two memory-intensive benchmarks of LU and SOR, respectively, in nbench [9] and Scimark2 [10] benchmark suites that are popularly used to measure CPU and memory system speed. Each benchmark is set to have virtual memory usages of 1.3GB, 1.5GB, and 1.7GB. Such a setting is based on a study [8] that executes benchmarks with the three virtual memory usages on computer systems having 1GB main memory.

The following parameters are used as performance metrics in the three virtual memory usages: the number of page-outs and page-ins, the execution time of benchmarks. Further, the distribution of cluster sizes in 1.5GB virtual memory usage is measured to show how many disk I/Os are issued according to cluster size that represents the number of pages in a cluster. Each benchmark has a similar distribution of cluster sizes in the three virtual memory usages.

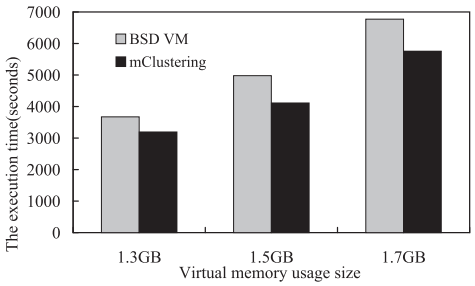
3.2 LU Benchmark

Figure 3 (a) shows that mClustering outperforms BSD VM by 13%, 16%, and 18% in the execution time on the experiment of 1.3GB, 1.5GB, and 1.7GB virtual memory usages, respectively. This is because mClustering generates fewer disk writes than BSD VM by 36%, 38%, and 41% each, as shown in Fig. 3 (b). Flushing multiple clusters reduces small disk writes that would be needed to flush small clusters of either one page or only a few ones in BSD VM. Figure 3 (c) shows how the multiple-cluster transfer affects the clustering behavior, as compared to BSD VM: disk writes in mClustering are dominantly for clusters of 16 pages, while BSD VM incurs a large portion of disk writes for clusters ranging between one and four pages. In particular, the significant reduction of disk writes for one page allows mClustering to improve the performance of page-outs. Moreover, the larger virtual memory usage gives mClustering more opportunities to reduce disk I/Os for page-outs, thus making the scheme further improve the page-out performance.

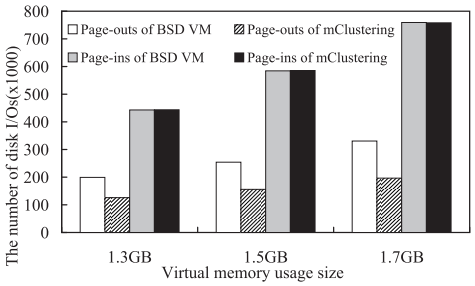
It is noticeable that mClustering is comparable to BSD VM in the number of page-ins in Fig. 3 (b). This result indicates that mClustering can improve the performance of page-outs without degrading that of page-ins. For the purpose, mClustering does not greedily flush as many pages as possible at a single disk I/O, but keeps efficiency of the prefetching by collecting pages into a buffer in a cluster unit.

3.3 SOR Benchmark

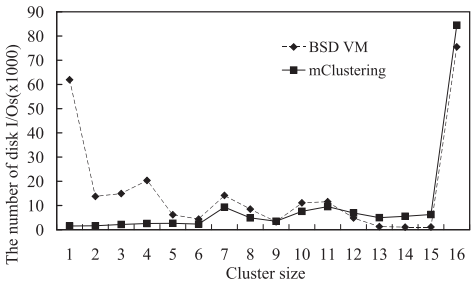
Figure 4 (a) shows that mClustering improves the execution time over BSD VM by 12%, 17%, and 23% in each virtual memory usage, with the help of decrease in disk I/Os for page-outs over BSD VM by 5%, 8%, and 13% in Fig. 4 (b).



(a) The execution time

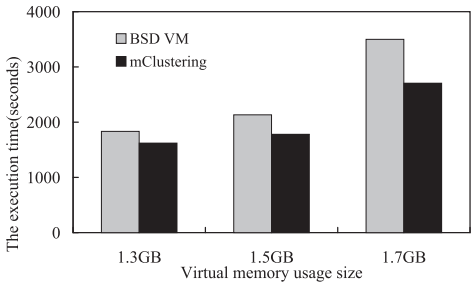


(b) The number of page-ins and page-outs

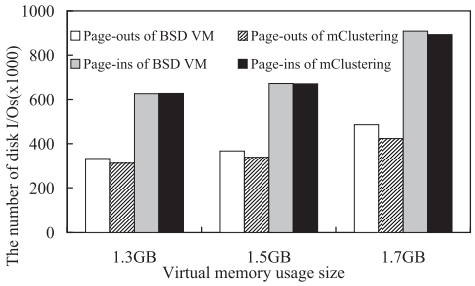


(c) The distribution of cluster sizes in 1.5GB virtual memory usage

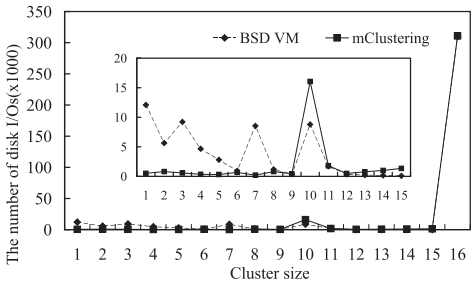
Fig. 3 LU benchmark performance.



(a) The execution time



(b) The number of page-ins and page-outs



(c) The distribution of cluster sizes in 1.5GB virtual memory usage

Fig. 4 SOR benchmark performance.

As shown in Fig. 4 (c), mClustering lessens disk writes for clusters of less than eight pages and increases those for clusters of 10, 13, 14, and 15 pages, but not for 16 pages. Besides clusters of one page, disk writes are largely dropped in clusters of two, three, four, five, and seven pages. However, such a cluster is likely to be larger than the available room of a buffer in which one or more clusters have already been collected. If mClustering attempts to collect the cluster into the buffer, it flushes the pages of the buffer that has not been filled up with 16 pages. Such a flushing makes mClustering fail to increase disk writes for clusters of 16 pages in SOR.

3.4 Discussion

mClustering may not improve the VM performance over BSD VM in the following situations. The first is when an application mainly changes contiguous pages in a VM space due to the highly strong locality of page writes. Such a write pattern makes both of the two schemes flush nearly 16 contiguous pages at one disk I/O, thus causing mClustering not to outperform BSD VM. However, the experiment on SOR described in Sect. 3.3 shows that mClustering can outperform BSD VM in an application that has the strong locality but causes not a few disk writes for small clusters. The second is when an application incurs page-ins more dominantly than page-outs. mClustering cannot improve the execution time of the application, even though the scheme significantly reduces disk I/Os for page-outs as compared with BSD VM. The third situation is when an application incurs the serious fragmentation of a swap disk. mClustering attempts to allocate a contiguous free space enough for multiple clusters, but can fail to find such a free space. The failure causes mClustering to flush each of the clusters at separate disk I/Os, similarly to BSD VM, thus disabling mClustering to reduce disk I/Os for page-outs.

4. Conclusions

mClustering reduces disk writes that BSD VM incurs in flushing dirty pages out, without sacrificing the performance of page-ins. This improvement is achieved by collecting small clusters of pages into large chunks, which are flushed in large disk I/Os. Our scheme significantly speeds up the execution of applications not having strong spatial locality

of page writes. Finally, we expect that mClustering can be easily applied to other OSes because only a small change is needed on the page-out routine to support paging out multiple clusters.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (No. 2011–0013781, No. 2012–0008457, No. 2010–0021094). The work reported in this paper was conducted during the sabbatical year of Kwangwoon University in 2012. This work was supported by the Catholic University of Korea, Research Fund, 2014.

References

[1] R. Azimi, L. Soares, M. Stumm, T. Walshand, and A.D. Brown, "PATH: page access tracking to improve memory management," Proc. Int. Symp. on Memory Management, pp.31–42, 2007.
[2] M.K. McKusick and G.V. Neville-Neil, The Design and Implementation of the FreeBSD Operating System, Addison-Wesley, 2005.
[3] D.L. Black, J. Carter, G. Feinberg, R. MacDonald, J.V. Sciver, P. Wang, S. Mangalat, and E. Sheinbrood, "OSF/1 virtual memory improvements," Proc. USENIX Mach Symp., pp.87–104, 1991.
[4] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young, "Mach: a new kernel foundation for UNIX development," Proc. USENIX Summer Conf., pp.93–112, 1986.
[5] A. Singh, Mac OS X Internals, Addison-Wesley, 2006.
[6] R.C. Murphy and P.M. Kogge, "On the memory access patterns of supercomputer applications: benchmark selection and its implications," IEEE Trans. Comput., vol.56, no.7, pp.937–945, 2007.
[7] M. Gorman, Understanding the Linux Virtual Memory Manager, Prentice-Hall, 2004.
[8] I.C. Tuduce and T. Gross, "Adaptive main memory compression," Proc. USENIX Annual Tech. Conf., pp.237–250, 2005.
[9] nbench benchmark http://www.tux.org/˜mayer/linux/bmark.html
[10] Scimark2 benchmark http://math.nist.gov/scimark2