J Inf Process Syst, Vol.15, No.1, pp.151~158, February 2019
https://doi.org/10.3745/JIPS.01.0037

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

MBS-LVM: A High-Performance Logical Volume Manager for Memory Bus-Connected Storages over NUMA Servers

Yongseob Lee* and Sungyong Park*

Abstract

With the recent advances of memory technologies, high-performance non-volatile memories such as non-volatile dual in-line memory module (NVDIMM) have begun to be used as an addition or an alternative to server-side storages. When these memory bus-connected storages (MBSs) are installed over non-uniform memory access (NUMA) servers, the distance between NUMA nodes and MBSs is one of the crucial factors that influence file processing performance, because the access latency of a NUMA system varies depending on its distance from the NUMA nodes. This paper presents the design and implementation of a high-performance logical volume manager for MBSs, called MBS-LVM, when multiple MBSs are scattered over a NUMA server. The MBS-LVM consolidates the address space of each MBS into a single global address space and dynamically utilizes storage spaces such that each thread can access an MBS with the lowest latency possible. We implemented the MBS-LVM in the Linux kernel and evaluated its performance by porting it over the tmpfs, a memory-based file system widely used in Linux. The results of the benchmarking show that the write performance of the tmpfs using MBS-LVM has been improved by up to twenty times against the original tmpfs over a NUMA server with four nodes.

Keywords

Logical Volume Manager, Memory Bus Connected Storage, Non-volatile Memory, NUMA, NVDIMM

1. Introduction

The development of high-performance non-volatile memories such as phase change memory (PCM), non-volatile dual in-line memory module (NVDIMM), and 3D XPoint has enabled the use of memory backend storages that are faster than motor driven storages. Among these, a memory bus-connected storage (MBS), such as NVDIMM-N, is directly connected to a memory bus and has the same performance characteristics as those of dynamic random access memory (DRAM). Due to its performance and non-volatile nature, the MBS is becoming an attractive alternative to high-performance server-side storages in human-centric computing [1–3], edge computing, the Internet of Things (IoT) [4], and cloud computing. When these MBSs are installed in non-uniform memory access (NUMA) servers, the access latency varies depending on the distance from the NUMA nodes. For example, Fig. 1 depicts benchmarking results that show the variation in access latency over a 4-node NUMA system, and

※ This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.
Manuscript received October 1, 2018; accepted November 10, 2018.
Corresponding Author: Sungyong Park (parksy@sogang.ac.kr)
* Dept. of Computer Science and Engineering, Sogang University, Seoul, Korea (tel.05056622231@gmail.com, parksy@sogang.ac.kr)

Copyright© 2019 KIPS

the difference in throughput when an MBS is installed at node 0.

As shown in Fig. 1(a), it is worth noting that the overall throughput can be improved if access to the MBS is localized. Moreover, in an environment where MBSs are installed in a specific node of a NUMA system, local access is possible only from the node where the MBSs are installed, whereas other nodes should access the MBSs remotely, which may result in poor application performance. However, if MBSs are scattered over different NUMA nodes and the storage space provided by each MBS is carefully managed, the file processing performance can be maximized. Therefore, the efficient management of storage spaces in NUMA servers where MBSs are distributed across different NUMA nodes has posed a challenge with regard to improving the performance of applications running on NUMA servers.

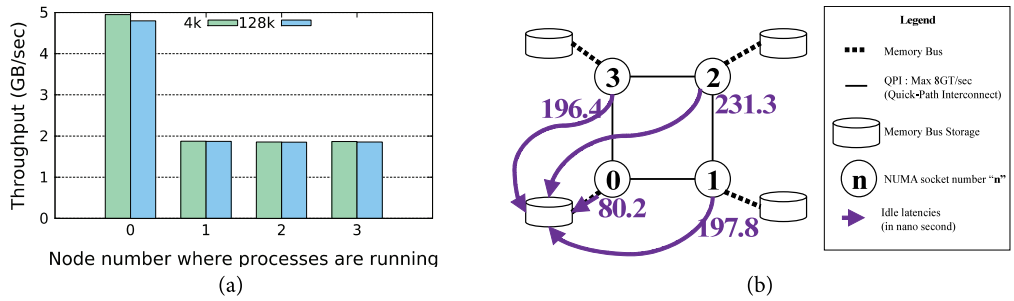

Fig. 1. Access latency and throughput over a 4-node NUMA system. (a) Random write on the MBS which is installed at node 0 (block size 4 kB and 128 kB). (b) NUMA idle latencies (in nanosecond) from node {0,1,2,3} to node 0.

Although several research efforts have proposed new non-volatile memory (NVM) file systems and optimization techniques [5–7], few studies have focused on improving file processing performance by considering the use of the MBS in the NUMA environment. Moreover, studies aimed at improving the performance in NUMA systems using DRAMs are not directly applicable to the MBS environment as file data should be persistently stored and because it is difficult to guarantee the consistency of files. Linux currently supports the use of MBSs by mounting them as block devices. In NUMA systems, each MBS installed in a different node can be mounted as a separate block device or a logically one block device using Linux logical volume manager. However, this approach cannot utilize the performance characteristics of the MBS because it is not NUMA-aware and the overheads incurred by using traditional software stacks are considerably heavy.

This paper presents the design and implementation of a high-performance logical volume manager for MBSs called MBS-LVM. MBS-LVM is lightweight in the sense that it removes all unnecessary software stacks and allows applications to directly access MBSs with high performance. It consolidates the address space of each MBS into a single address space and enables the allocation of storage space in a local MBS as much as possible so as to improve the write performance in a NUMA system. The MBS-LVM was implemented in the Linux kernel, and its write performance was evaluated by porting it over the *tmpfs*, a memory-based file system in Linux. The benchmarking results show that the performance of the modified *tmpfs* using MBS-LVM is almost twenty times greater than that of the original *tmpfs* over a NUMA server with four nodes.

The rest of this paper is organized as follows. Section 2 presents the design of the MBS-LVM and discusses various implementation issues. Section 3 compares its performance and provides the benchmarking results and Section 4 presents the conclusion.

2. Design and Implementation

2.1 Design Overview

The MBS-LVM is composed of three components, as shown in Fig. 2, namely the MBS cluster manager, MBS monitor, and MBS space manager. The MBS cluster manager virtualizes the address space of each MBS and combines them as a single address space. The MBS monitor checks whether the available space used by each MBS exceeds the pre-defined threshold or maximum capacity, and determines the target MBS with the lowest access latency among the candidates. The MBS space manager is responsible for allocating storage spaces from the target MBS determined by the MBS monitor. The implementation details of each component are given as follows.

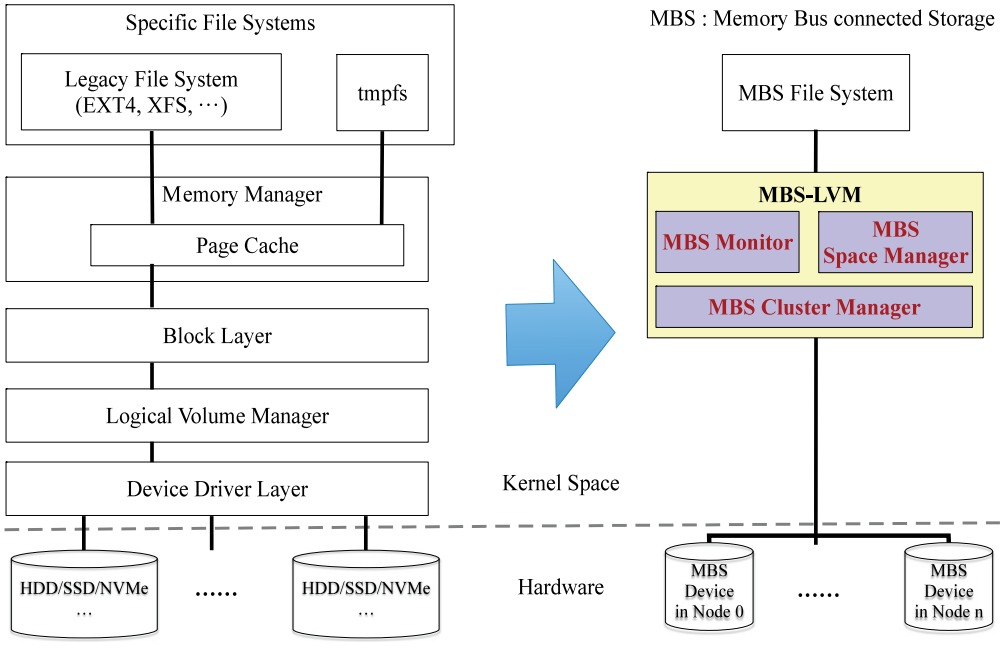

Fig. 2. Overview of MBS-LVM.

2.2 MBS Cluster Manager

In order to consolidate the address spaces of the MBSs installed in different NUMA nodes into a single address space, the MBSs should be composed as a cluster. From the basic input/output system (BIOS), MBS installation information can be obtained, and the MBS cluster manager combines the MBSs using that information. For this, the *sparsemem* model technique used in Linux kernel is used. For example, the information of the device installed in a DIMM slot is delivered to the Linux kernel by the BIOS during the booting process, and the kernel can distinguish MBS from DRAM through the e820 map. The Linux kernel manages the contiguously installed DIMMs in a *memblock_region* structure. To configure the MBS cluster, the installation information of the MBS, such as NVDIMM-N, is stored in a *memblock* structure through the *memblock_region* structure. To distinguish the MBS from the conventional DRAM, a new memblock_type and MBS zone are added, as shown in Fig. 3.

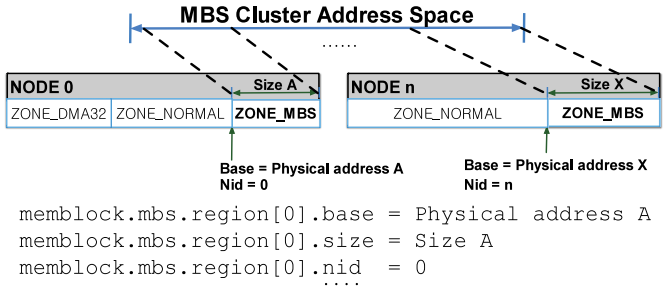

Fig. 3. MBS zone, MBS cluster, and region for MBS.

2.3 MBS Space Manager

The MBS space manager allocates the storage space from the MBS with the lowest access latency. It was confirmed through various preliminary experiments that the file processing performance can be improved if a local MBS is used as much as possible. The MBS space manager maintains the storage spaces provided by the MBS with the buddy system and allocates the space from the target MBS determined by the MBS monitor.

2.4 MBS Monitor

The MBS monitor periodically checks each NUMA node and classifies them as either *FatNode* or *CandidateNode*. *CandidateNode* is a set of nodes with MBSs that have enough free space, whereas *FatNode* is a set of nodes with MBSs that do not have free space and thus cannot be selected as a candidate node. *CandidateNode* is maintained as a sorted linked list where the head of the list represents the MBS node with the lowest access latency. If files are deleted from the MBSs, it is possible that any nodes in the *FatNode* list can be included in the *CandidateNode* list. As shown in Fig. 4, this scheme guarantees that the local MBS is the target MBS until it does not have enough free space. When the local MBS is full, the MBS with the lowest latency is chosen as a candidate MBS.

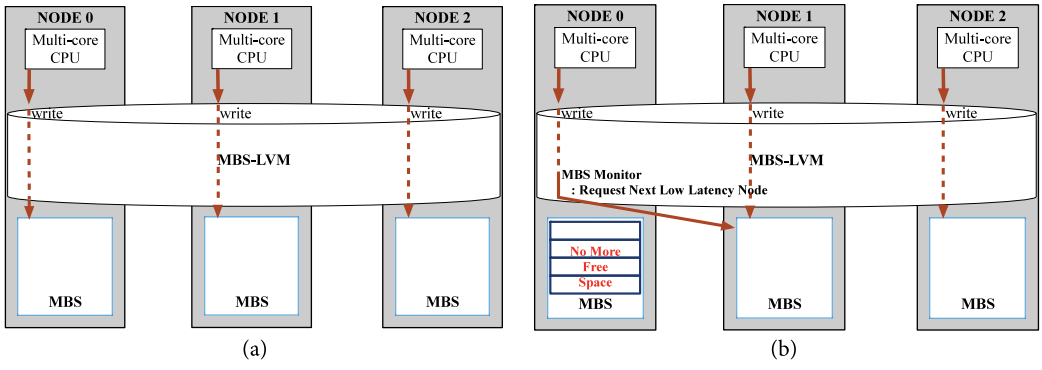

Fig. 4. Step for deciding the target MBS. (a) Local allocation and (b) New low access latency node. allocation.

When a write operation is invoked, it asks the MBS monitor to determine a target MBS with the lowest access latency. Then, the MBS monitor returns the target MBS's ID to the write function, which in turn

asks the MBS space manager to allocate a storage space in the target MBS. The detailed algorithm in the MBS monitor is summarized in Algorithm 1.

Algorithm 1. MBS Monitor

```
procedure init CandidateNode
   calculate the distance and access latency between nodes
   sort the nodes by access latency
   save the order list to the CandidateNode
end procedure

procedure MBS Monitor
 call init CandidateNode
 while (1) { /* run periodically */
   for ( i = 0 ; i < MAX_NODE; i++ )
      if Node(i) has no free space then
          remove Node(i) from CandidateNode
         add Node(i) to FatNode
      else if Node(i) in FatNode has enough free space then
          remove Node(i) from FatNode
         add Node(i) to CandidateNode
      end if
   end for
 end while

procedure Decide_Target_MBS
 if Node(x) is in CandidateNode != NULL then
      RequestNode=Node(x)
 else if Next_all(CandidateNode) != NULL has enough space then
    RequestNode=Next(CandidateNode)
 else
    RequestNode=NULL
 endif
 Return the target MBS id
end procedure
```

3. Evaluation

Testbed: To evaluate the performance of the MBS-LVM, a NUMA system with four CPUs (10-cores in each CPU) running Linux was used. The MBSs were emulated using DRAMs with the Linux kernel parameter. Each node was configured to be equipped with 16 GB of MBS and 64 GB of MBS clusters. A new file system was also implemented by modifying the Linux *tmpfs* (called *mbsfs*) in such a way that it could directly store and read files from the MBS clusters.

Workload: Both synthetic and real workloads were generated to evaluate the performance of the MSB-LVM. In the synthetic workload tests, the throughput and scalability of the *mbsfs* were compared with those of the *tmpfs* using the *fio* benchmark by varying the block size (from 1K to 2M) and the number of nodes (from 1 to 4). The number of threads was set to 40. To measure the scalability of the MBS-LVM, the strong scale test, a parallel processing scalability evaluation technique for high-performance computing systems, was used. In order to evaluate the MBS-LVM over real workloads, the MongoDB YCSB (Yahoo! Cloud Serving Benchmark) benchmark, which generates heavy I/O load and is generally used to evaluate NoSQL databases, was used. The type of generated workload was *workloada* (with a read and write ratio of 50:50) and the number of records was set to 2 million.

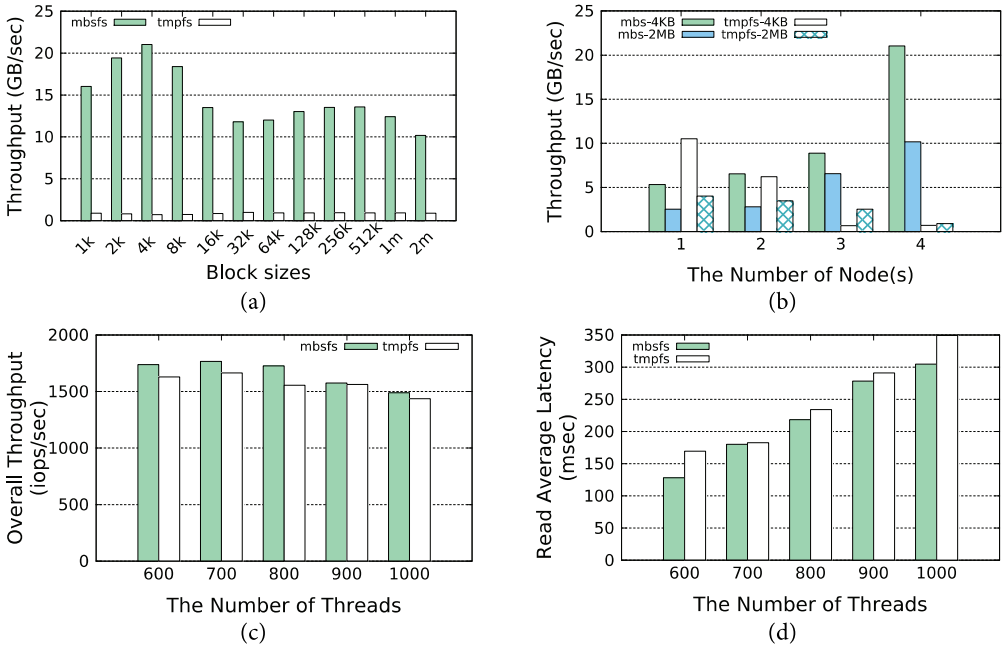

Fig. 5. Performance comparison with random write and YCSB *workloada* (read and write ratio of 50:50). (a) Random write throughput, (b) strong scalability test, (c) YCSB (*workloada*) throughput (high is better), and (d) YCSB (*workloada*) average latency (low is better).

Result: Fig. 5 shows the performance comparison of the *mbsfs* and *tmpfs* by using the *fio* random write and MongoDB YCSB benchmarks. In the synthetic workload tests using *fio* with random write, as shown in Fig. 5(a) and (b), the *mbsfs* outperformed the *tmpfs* by about 20 times, and scaled well up to 4 nodes compared with the *tmpfs*. This is because the MBS-LVM allocates the spaces for writing from the MBS with the lowest access latency, while the *tmpfs* pre-allocates a large amount of memory for writing in advance and uses the memory for all write requests. In this case, remote access is mandatory if a thread requesting write operations is running on different NUMA nodes. If the storage and thread are located on the same node in a NUMA system, it is helpful in improving the performance, but if the storage and thread are located in different nodes, it lowers the performance due to remote access latency. It is also worth noting that the throughput varied as the block sizes were increased, with the *mbsfs* showing the best performance in the 4 kB block size. This is because I/O operations are sometimes optimized for the

page size (i.e., 4 kB) used in the operating systems. In real workload tests using the MongoDB YCSB benchmark with a read and write ratio of 50:50, as shown in Fig. 5(c) and (d), the *mbsfs* also outperformed the *tmpfs* by up to 10% in terms of overall throughput, and even showed better performance than the *tmpfs* in the read operations. Since the write operations preceded the read operations in the benchmark tests, the read latency could be minimized if the threads invoking the write operations read the data from the same NUMA node. It is expected that the performance can be improved further still by increasing the write ratio.

4. Conclusion

This paper proposes an MBS-LVM designed to improve the performance of write operations in NUMA-based servers. The MBS-LVM virtualizes the address space of each MBS and combines them into a single address space. By allocating storage spaces to a local MBS node as much as possible, the MBS-LVM improves the performance of applications running on top of the NUMA systems. The superiority of the MBS-LVM was proved by porting it into the *tmpfs*, a memory-based file system that is used in Linux. The results of benchmarking with both synthetic and real workloads showed that a new file system (*mbsfs*) using the MSB-LVM outperformed the traditional *tmpfs* by up to twenty times. The MBS-LVM proposed in this paper provides many benefits with regard to write performance, but does not guarantee a higher performance for read operations compared to existing methods. Further research will be conducted to improve its read performance as well.

Acknowledgement

This paper was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. NRF-2017R1D1A1B03032763).

References

[1] J. P. Martin, A. Kandasamy, and K. Chandrasekaran, "Exploring the support, for high performance applications in the container runtime environment," *Human-centric Computing and Information Sciences*, vol. 8, article no. 1, 2018. https://doi.org/10.1186/s13673-017-0124-3.

[2] E. Gultepe and M. Makrehchi, "Improving clustering performance using independent component analysis and unsupervised feature learning," *Human-centric Computing and Information Sciences*, vol. 8, article no. 25, 2018. https://doi.org/10.1186/s13673-018-0148-3.

[3] A. G. Finogeev, D. S. Parygin, and A. A. Finogeev, "The convergence computing model for big sensor data mining and knowledge discovery," *Human-centric Computing and Information Sciences*, vol. 7, article no. 11, 2017. https://doi.org/10.1186/s13673-017-0092-7.

[4] H. Cai, B. Xu, L. Jiang, and A. V. Vasilakos, "IoT-based big data storage systems in cloud computing: perspectives and challenges," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 75-87, 2017.

[5] D. Niu, Q. He, T. Cai, B. Chen, Y. Zhan, and J. Liang, "XPMFS: a new NVM file system for vehicle big data," *IEEE Access*, vol. 6, pp. 34863-34873, 2018.

[6] J. Xu and S. Swanson, "NOVA: a log-structured file system for hybrid volatile/non-volatile main memories," in *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST)*, Santa Clara, CA, 2016. pp. 323-338.

[7] J. W. Kim, J. H. Kim, A. Khan, Y. Kim, and S. Park, "ZonFS: a storage class memory file system with memory zone partitioning on Linux," in *Proceedings of 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, Tucson, AZ, 2017, pp. 277-282.

Yongseob Lee https://orcid.org/0000-0001-5469-1538

He is a PhD student in the Graduate School of Sogang University, Seoul, Korea. He received B.S. degree in Computer Engineering from Hongik University and M.S. degree in Graduate School of Information and Technology, Sogang University. Since September 2010, he is with the Graduate School of Computer Science and Engineering from Sogang University as a PhD candidate.

Sungyong Park https://orcid.org/0000-0002-0309-1820

He is a professor in the Department of Computer Science and Engineering at Sogang University, Seoul, Korea. He received his B.S. degree in computer science from Sogang University, and both the M.S. and Ph.D. degrees in computer science from Syracuse University. His research interests include cloud computing and systems, virtualization technologies, autonomic computing, and embedded system software.