J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr

Hybrid in-memory storage for cloud infrastructure☆

Dae Won Kim[1*] Sun Wook Kim[1] Soo Cheol Oh[1]

ABSTRACT

Modern cloud computing is rapidly changing from traditional hypervisor-based virtual machines to container-based cloud-native environments. Due to limitations in I/O performance required for both virtual machines and containers, the use of high-speed storage (SSD, NVMe, etc.) is increasing, and in-memory computing using main memory is also emerging. Running a virtual environment on main memory gives better performance compared to other storage arrays. However, RAM used as main memory is expensive and due to its volatile characteristics, data is lost when the system goes down. Therefore, additional work is required to run the virtual environment in main memory. In this paper, we propose a hybrid in-memory storage that combines a block storage such as a high-speed SSD with main memory to safely operate virtual machines and containers on main memory. In addition, the proposed storage showed 6 times faster write speed and 42 times faster read operation compared to regular disks for virtual machines, and showed the average 12% improvement of container's performance tests.

☞ keyword: cloud computing, hybrid in-memory, virtual machine, container

1. Introduction

In-memory computing technology has begun to emerge again from the requirement for data storage, which is the biggest bottleneck in processing big data and artificial intelligence systems[1], [2]. Unlike conventional disk-based computing, in-memory computing does not store and manage data on a hard disk, but means loading and using the entire data in memory [3].

A typical example of in-memory computing is well-known as a caching system in a memory hierarchy. Due to the recent development of cloud and edge computing, various approaches using memory are being implemented, and as a technology suitable for the current cloud and edge computing technology, its utilization is rapidly increasing [4].

For example, technologies such as in-memory data grids [5],[6] and in-memory databases [7],[8],[9],[10] are developing, but most of the in-memory computing services are currently in their infancy, and some companies also provide streaming engines, machine learning, distributed processing, SQL, using in-memroy computing and provides an platform which is integrated with above functions.

When using disk storage or SSDs in a cloud computing, if the number of virtual machines or containers per physical machine increases, the system load is concentrated on the storage with the virtual disks, and performance deteriorates. This technology can lead to I/O bottlenecks and performance degradation in cloud environments. Therefore, virtual machines or containers used in cloud computing can be stored in main memory to improve performance [11],[12]. And the host utilizes main memory using a memory-based temporary filesystem or storage (such as tmpfs) for containers [13].

However, this technique has disadvantages in that it cannot store large amount of data due to the limited capacity of the main memory and it is difficult to overcome the volatile characteristics of the memory. Of course, in the case of Optane memory recently announced by Intel [14], it can be a good example to improve performance and overcome the shortcomings of memory device. However, there are still problems in terms of price and capacity for the commercialization. Therefore, the above-mentioned

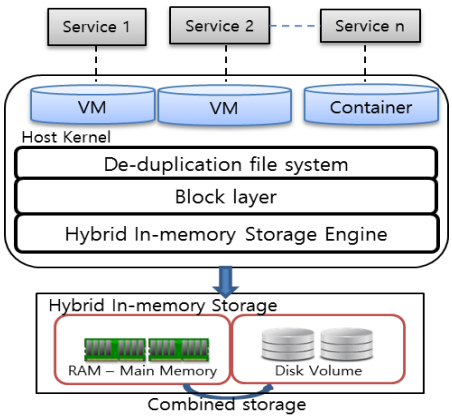
[1] Electronics and Telecommunications Research Institute. 218 Gajeong-ro, Yuseong-gu, Daejeon, South Korea
* Corresponding author (won22@etri.re.kr)
[Received 25 June 2021, Reviewed 10 July 2021(R2 6 October 2021), Accepted 13 October 2021]
☆ This work was supported by the Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) in 2021. [No. 2020-0-00116 » *Development of Intelligence Cloud Edge SW Platform*]
☆ A preliminary version of this paper was presented at ICONI 2020.

limitations and disadvantages should be overcome in order to implement in-memory computing using the main memory in spite of the high cost and resource constraints.

In this paper, data storage and operation using main memory are supported for high-speed data access when a customer uses a virtual machine or container environment, and as the user creates data, the memory capacity increases rapidly to reduce the total memory capacity. Introduce research and development to solve the problem of excess and eliminate data loss due to the volatile nature of memory.

In this paper, we propose a hybrid in-memory storage that integrates main memory (e.g. RAM) and storage (e.g. SSD) to solve the drawbacks of main memory due to capacity limitation and volatile characteristics for in-memory computing. It implements hybrid in-memory storage by uniting main memory and storage into a single storage view, and provides high-performance virtual environment and large-capacity storage for cloud services. We also want to apply hybrid in-memory storage to our cloud infrastructure to utilize it for high performance. Section 1 provides an overview of hybrid in-memory storage and section 2 describes the detailed structure and architecture and operation. Section 3 describes the virtual machine and container applications and experimental results, and section 4 provide the further studies. At last, section 5 make conlusion.

2. Overview of Hybrid In-memory Storage



(Figure 1) Structure of hybrid in-memory storage

Hybrid in-memory storage is a virtual block storage device that uses a specific unit (in gigabytes) of memory as a RAM disk and backs up data either synchronously or asynchronously to a high-performance non-volatile block device such as an SSD. Figure 1 shows the overall structure of the hybrid in-memory storage proposed in this white paper.

The entire system consists of a hybrid in-memory storage combined with virtual disks, a hybrid in-memory storage engine, main memory (RAM) and disk storage (SSD). Virtual machines create and operate virtual images (eg OS images, disk images, etc.) [15] on hybrid in-memory storage. And for containers, place the container filesystem on a virtual disk on hybrid in-memory storage. The hybrid in-memory storage engine creates a single storage and combines main memory and disk storage into a single storage for single view of storage. Hybrid in-memory storage also provides standard block storage I/O interfaces, allowing existing virtual machines to operate without modification.

Virtual disk access commands created by virtual machines or containers are forwarded to hybrid in-memory storage. Because the hybrid in-memory storage engine combines main memory and disk storage to create hybrid in-memory storage, the hybrid in-memory storage engine selects the integrated main memory and disk storage to process disk access commands. In addition, hybrid in-memory storage can significantly reduce the physical virtual desktop image by using a file system with deduplication for limited memory operations.

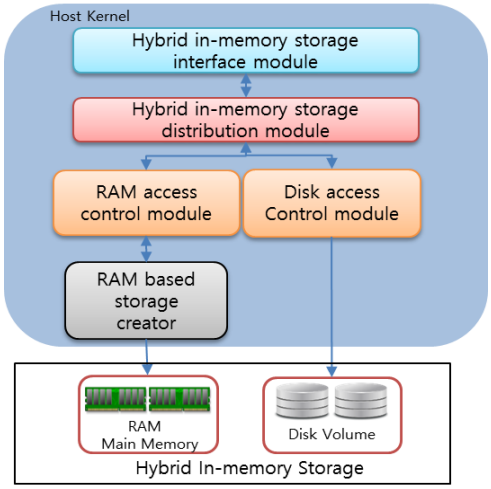
2.1 Architecture of Hybrid In-memory Storage Engine

Figure 2 shows the architecture of a hybrid in-memory storage engine. The hybrid in-memory storage engine consists of a hybrid storage interface module, a hybrid storage deployment module, a RAM access (main memory) control module, a RAM-based storage generator, and a disk access control module.

The hybrid in-memory storage interface module provides a standard block interface and receives virtual disk access commands created by virtual machines or containers. Received commands are forwarded to the hybrid in-memory

storage distribution module. According to the nature of the command, the hybrid storage distribution module determines whether to use main memory or disk storage to perform the command, and sends the command to the RAM access control module or the disk access control module.

The RAM access control module uses main memory as a disk to process disk access commands to provide high-speed access. The main memory storage creation module performs the actual read/write operations on main memory that can be accessed on a address basis with disk access commands sent in blocks. This allows the data from the virtual disk to be stored in main memory. The disk storage control module uses disk storage to process virtual disk access commands.

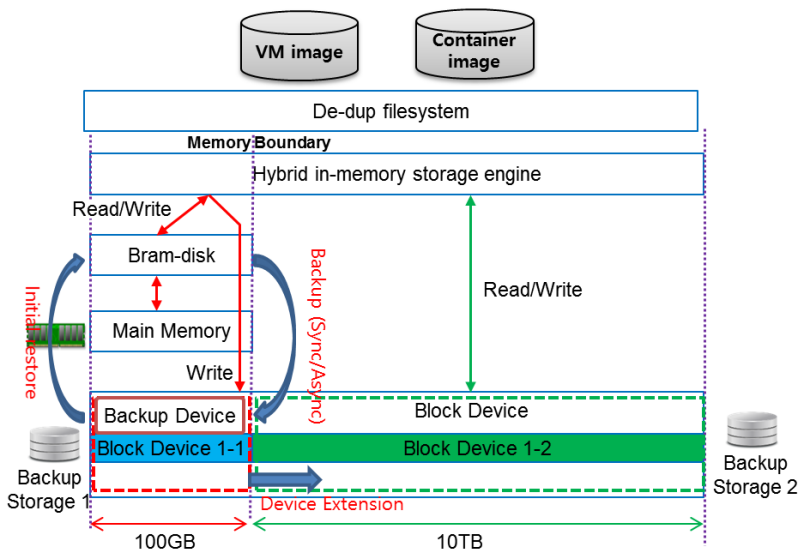


(Figure 2) Structure of hybrid in-memory storage engine

2.2 Operation of Hybrid In-memory Storage

Figure 3 shows how hybrid in-memory storage initially works. When the system starts up, the data in the storage is restored to the main memory (RAM disk) of the hybrid in-memory storage.

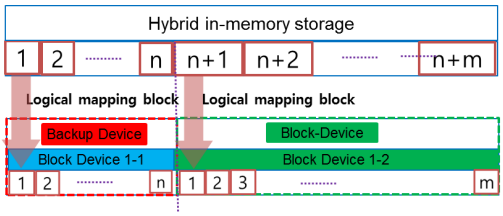
Even during restoration, the hybrid in-memory storage engine can continue service without interruption. After restoration, the hybrid in-memory storage engine performs read/write (backup) operations. In synchronous mode, all data transferred to the hybrid in-memory storage engine is simultaneously stored on RAM disk and SSD. When the save to both devices is complete, the write to the upper layer is complete. However, in asynchronous mode, data transferred to the hybrid in-memory storage engine is only stored on the RAM disk. A kernel thread inside the hybrid in-memory



(Figure 3) The operation of hybrid in-memrory storage

storage engine stores data from RAM to the SSD in Least Recent Written (LRW) order.

In figure 3, hybrid in-memory storage supports a storage expansion to overcome the limitation of memory capacity. As described above, two types of disk image operation (in-memory operation and disk operation) are supported and used by merging them. As shown in the figure 3, when the memory capacity is full, newly created data is written to Block device 1-2 without writing to the main memory, and the data written to Block device 1-2 is read in the disk operation, not in the main memory. The data written to Block device 1-1 is operated through the existing in-memory operation..



(Figure 4) Logical mapping blocks.

Figure 4 shows how to configure hybrid in-memory storage with RAM and block storage. Hybrid in-memory storage provides a standard block storage device format to provide users with a common interface and physically map the RAM disk area at the front and the disk (SSD) area at the back of the hybrid in-memory storage logically.

If there is RAM blocks with n block IDs and a disk (SSD) blocks with m block IDs, then block ID 1 to n of RAM are mapped to block ID 1 through n of hybrid in-memory storage to combine the two regions. Block ID 1 to m disks (SSDs) are mapped to hybrid in-memory storage with block ID n+1 to n+m.

And a storage boundary is established between the hybrid in-memory storage block ID n and n+1. When a disk access command is received, the block ID of the command is checked. If the block ID is less than or equal to the storage boundary, the command is sent to the RAM access control module and the command is processed using high-speed main memory. If the block ID is greater than the storage boundary, the command is passed to the disk access control module, where it can be processed in large amounts of disk storage.

2.3 Read/Write

There are two mode in read/write operation.

In synchronous mode, read operations are independent of the SSD. Internally, it is only processed from kernel memory in hybrid in-memory storage. The DMA engine [16] transfers data from hybrid in-memory storage to user buffers. In other words, the DMA Engine runs in memory without copying data. In synchronous mode, write operations are simultaneously written to main memory and SSD. The requested write is finally completed only when the DMA copy has completed and the SSD write has also completed. All data movement is handled by DMA.

In asynchronous mode, all write operations are performed in kernel memory, and a completion message is returned when the write operation is completed in kernel memory. Data written to the memory is later written to the SSD according to the delayed write method. Even if the write speed of the SSD is slow, higher write requests are not blocked, so the final write speed will be similar to the speed of the RAM disk.

2.4 De-duplication

Read/Write commands to hybrid in-memory storage are passed to the hybrid in-memory storage engine, and then passed to memory through deduplication [17]. In the case of a virtual machine, memory loss due to the size of the OS image can be very large. To solve this problem, a deduplication function is provided. The deduplication function of hybrid in-memory storage deduplicates the image of the virtual machine and reduces it so that it can be stored in main memory. Deduplication of the virtual machine image is performed whenever a write operation to the virtual machine image occurs. When a write command to disk occurs inside the virtual machine, it is sent to the deduplication module in the host as shown in Figure 1. Therefore, the deduplication function performs the deduplication function by capturing the write event that occurs during image operation of the virtual machine in real

time. After calculating the required fingerprint, it is stored in the DB. After examining the DB, if the same block to be written exists, only the metadata necessary for writing is recorded and the write operation is completed. If not, write the write block to that memory and store the metadata at the same time.

3. Applied Tests & Results

3.1 Management of Hybrid In-memory Storage

Hybrid in-memory storage is installed in the form of a kernel module. To facilitate installation and management, it is installed in the Linux kernel's Proc file system and managed through it as shown in figure 5. It is provided in the form of /proc /<device_name> as shown below. And you can check various information in /proc/sys/dev/<device_name>/ <device_ID>/. Also, GUI or CLI management tools are provided for user convenience. It is designed to utilize this tool to check, create, delete and restore status

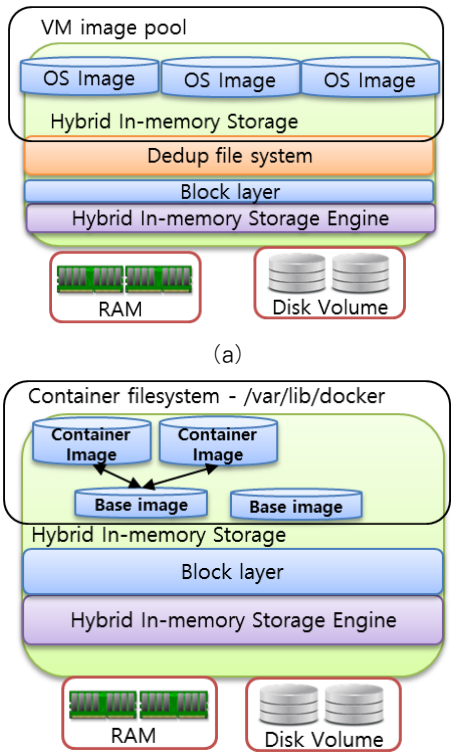
```
root@storage4:~/sbr/cli# cat /proc/sbr
 -------)sbr status (ver. 1.0) -------

ID: 0
backing storage: sdd1
ram size: 80 [GiB]
device size: 81920 [MiB]
ramdisk chunk size: 65536
under restoring:  7%(6467/81920)
Counting I/Os: disabled
asynchronous backup: on
dirty ratio: 0%(0/1310720)
write back watermark: high 131072, low 65536
activity : idle
# of pending io: 0
strict consistency: off
```

(Figure 5) Proc filesystem view for hybrid in-memeory storage.

3.2 Test Environments

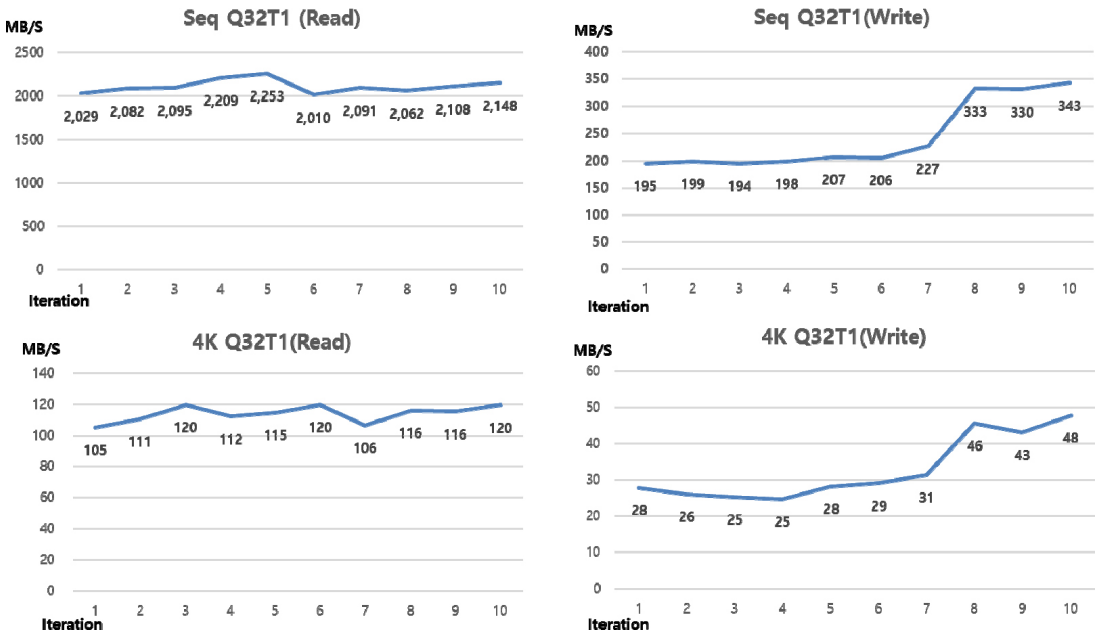
The architecture of the two systems for applying hybrid in-memory storage is shown in Figure 6.

The platform for applying the virtual machine system was configured using Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz 64Core, CentOS 7.0, qemu-kvm hypervisor [18]. The virtual machine's operating system was configured as window10 and installed on hybrid in-memory storage. Virtual machines run on hybrid in-memory storage with the entire operating system image. These operating system images are available in sizes ranging from tens to hundreds of gigabytes, and user disks are also provided in divided portions. Deduplication is essential to provide these disks in memory space. In the case of operating system images, the same image used by default is provisioned through the clone method as the primary base. So we were able to reduce the 30G operating system to 512M.



(Figure 6) VM & container(docker) application ((a)VM and (b) container)

For testing the container system, a container (docker)[19] creates a hybrid in-memory container storage, and configures the storage volume of the container on this storage. A container creates and operates a file system (eg



(Figure 7) The benchmark results for virtual machine. (Crystal disk Mark)

/var/lib/docker for docker) volume where the container runs on hybrid in-memory container storage.

The container file system is composed of an unifying file system (eg overlayfs [20]) and is composed of layers. It consists of a merged access area, a container layer, and an image layer. Each layer operates by creating and mounting a specific directory on the in-memory container storage.

The container layer is a writable layer, and each container is created on the top layer, allowing each container to have its own state. After the container is created, all changes are made in this layer. However, the R/W speed is fast because it is done in memory. And for the efficiency of file management, the difference information between the actual image and the container image is included.

The image layer is a read-only layer that can be shared with other containers. In addition, multiple images shared with other layers can be operated in the container layer. And the integrated access area includes link information of the layer accessible to all file systems of the container layer and the image layer and is shared with users. This allows access to the file. The image layer can be shared with many different systems to increase its efficiency. Container images in the image layer should be pulled from public repositories (eg github [21]) when containers are deployed. In this case, it is efficient to store the image used in the container system locally or to bring it in advance to ensure performance. In this paper, the deployment speed is also fast because the already pooled image exists on the hybrid in-memory disk. In this paper, we tested using docker's overlayfs and proceeded without performing deduplication on the system.

(Table 1) Testing environments for hybrid in-memory storage

Testing Environment		
Type	VM	Container
CPU	Intel® Xeon® CPU E5-2697A 2.6GHz 64Core (16)	
RAM	755 GB	
Storage	2.6 TB (SSD)	
OS	CentOS 7.2.1511	Ubuntu 16.04
Kernel	3.10.0-327	4.4.0-generic
Virtual environment	qemu-kvm,	Docker-ce
Network	1G Ethernet	
No. of Node	3 (1 Management Node + 2 Operating Node)	

Test – NginX Deploy

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: disktype
            operator: In
            values:
            - mem
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
```

Web.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: disktype
            operator: In
            values:
            - ssd
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
```

Web_ssd.yaml

The deployment of Seleted node

```
won22@kubemaster:~$ sudo kubectl get nodes --show-labels
NAME         STATUS   ROLES    AGE    VERSION   LABELS
kubekdw1     Ready    <none>   279d   v1.18.3   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,disktype=mem,kubernetes.io/arch=amd64,kubernetes.io/hostname=kubekdw1,kubernetes.io/os=linux
kubekdw2     Ready    <none>   279d   v1.18.3   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,disktype=ssd,kubernetes.io/arch=amd64,kubernetes.io/hostname=kubekdw2,kubernetes.io/os=linux
kubemaster   Ready    master   279d   v1.18.3   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=kubemaster,kubernetes.io/os=linux,node-role.kubernetes.io/master=
```

(Figure 8) The test for nginx deployment in kubernetes

3.3 Test Results

3.3.1 Test for virtual machine

In a virtual machine system, the most commonly used crystal disk mark [22] for storage testing was installed and tested on the OS (window 10) of the virtual machine. For the Seq Q32T1, it refers to the speed when sequentially reading and writing files in the queue, i.e. the speed when reading and writing a size of 128 KiB with 32 instructions.

4K Q32T1 for Windows operating system. Since it uses 4 KB as a cluster on an NTFS system, which is the usual format, we measure the speed while randomizing the queue and the number of threads respectively.

For CrystalDiskMark, the Seq Q32T1 is considered the maximum for read/write speed and the 4K Q32T1 is considered the actual read/write speed. These experiments were tested using a benchmark tool (CrystalDiskMark) and are the result of 10 periodic tests for each read/write operation as shown in figure 7. The measured results were used to obtain the average of the two benchmark results. Average values are given in Table 2.

Comparing to the average values as shown in Table 2, it can be seen that the two tested benchmark cases have significantly improved performance compared to normal storage. But more importantly, it's a performance boost. The 4K Q32T1 has a better performance boost than the Seq Q32T1, with performance gains of 42x for read operations and 6x for write operations. Due to the nature of Windows systems and asynchronous writes using DMA, it has been found that write operations provide less performance improvement than read operations.

(Table 2) The average Speed (MB/S) of Read/write operation

	Disk based VM		Ours	
	Read	Write	Read	Write
Seq Q32T1*	86.12	208.30	2,108.70	243.18
4K Q32T1**	2.70	4.98	114.02	32.82

* Seq Q32T1: Sequential (Block Size=128KiB) Read/Write (Queue:32/Thread:1)

** 4K Q32T1: Random 4KiB Read/Write (Queue:32/Thread:1)

3.3.2 Test for container

For the test of the container system, kubernetes [23] was used, and the test was conducted by dividing a single master node, a node equipped with an SSD, and a node equipped with a hybrid in-memory storage. And, Nginx [24] image, which is the most used web server application on hybrid in-memory storage, was deployed as shown in figure 8.

When the deployed image is first configured, the image is pulled through github and distributed on the hybrid in-memory storage from the completed image when redistributed. The test results are shown in Table 3 below and the test results showed a performance improvement of about 30%.

(Table 3) Nginx deployment test result (sec)

	Ours	Docker in SSD	Perf. Up
Nginx with pulling	0.690	1.026	32%
Nginx without pulling	0.49	0.71	28%

(Table 4) Nginx load test result

Max value comparison	Ours	Docker in SSD	Perf. Up
Time taken for tests(s)	0.72	1.09	30%
Requests per second (mean) [#/sec]	6950	4607	33%
Time per request (mean)[ms]	71.9	108.5	33%
Time per request (mean, across all concurrent request) [ms]	0.14	0.22	33%
Transfer rate [Kbytes/sec]	5735	3802	34%
Connection Time(ms)	13~227	11~1024	
50 Iteration Average	**Ours**	**Docker in SSD**	**Perf. Up**
Time taken for tests(s)	1.28	1.55	17%
Requests per second (mean) [#/sec]	3901	3457	12.8%
Time per request (mean)[ms]	128.3	154.6	17%
Time per request (mean, across all concurrent request) [ms]	0.26	0.31	16%
Transfer rate [Kbytes/sec]	3219	2853	12.8%
Connection Time(ms)	12.7~1198	1485.56	

As a second test, we conducted a web load test of the deployed nginx as shown in Table 4. It was conducted through the ab test of httpd-tools [25], and ab is a benchmarking tool for the performance of the Apache HTTP server. In the test, 500 simultaneous connections to 5000 requests were repeated 50 times. As shown in the table, as a result of the test, the performance of nginx web showed a performance improvement of about 30% in the comparison of the maximum value, and an average performance of 50 times showed a performance improvement of about 12%.

4. Further Researches

We have tests to apply virtual machines and container environments, which are often used in cloud computing, to in-memory computing. As mentioned above, use cases for the utilization of volatile memory in in-memory computing are limited to the extent of databases.. In the future, this system will maximize the stability for applying and commercializing in-memory computing technology to edge computing by developing storage utilizing main memory by software. Also, large distrubuted system such as edge computing and cloud computing clould be needed to develop memory to share for fast data transaction or high avalabilty [26]. Another research direction is to develop a high-speed technologies that is more convenient and easier than using technologies using high speed memory devices such as PMEM device currently being developed.

5. Conclusion

In this paper, we developed a hybrid in-memory storage for stability and high performance of cloud computing infrastructure and tested it by applying a hypervisor-based virtual machine system and a container-based cloud-native environment. Hybrid in-memory storage combines the characteristics of main memory and general storage (SSD) to overcome the disadvantages of data loss due to the volatile characteristics of memory, and virtual This has resulted in improved performance of machines and containers. In the case of the presented virtual machine system, the OS image, which has the greatest influence on the speed of the virtual

machine, is stored in the main memory so that high-speed virtual machines can be executed. This allows the container image to run in main memory. When a virtual machine is deployed, users continue to create data while using the virtual machine, and when this exceeds the main memory capacity, the data is stored on disk storage. Additionally, for container file systems, image data due to continuous pooling is stored on disk storage. Because user data is stored on disk storage, file access speed is slower than in-memory, but suitable for efficient operation and storage of large data. Therefore, it is stored in the main memory to support high performance of the cloud infrastructure, and large user data is stored in disk storage to overcome the capacity limit of the main memory, so that an efficient cloud environment can be built. Also, due to the volatile nature of main memory, it can reasonably cope with backup/restore.

References

[1] Hao Zhang, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, Meihui Zhang, “In-Memory Big Data Management and Processing: A Survey,” IEEE Transactions on Knowledge and Data Engineering, Vol 27, issue 7, pp. 1920-1948, July, 2015. https://doi.org/10.1109/TKDE.2015.2427795

[2] Soroosh Khoram, Yue Zha, Jialiang Zhang, Jing Li, “Challenges and Opportunities: From Near-memory Computing to In-memory Computing,” Proceedings of the 2017 ACM on International Symposium on Physical Design, pp. 43 – 46, March 2017. https://doi.org/10.1145/3036669.3038242

[3] S. Robbins, “RAM is the new disk,” InfoQ News, Jun. 2008. https://www.infoq.com/news/2008/06/ram-is-disk/

[4] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazieres, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman, “The case for RAMClouds: Scalable high-performance storage entirely in dram,” ACM SIGOPS Operating Syst. Rev., vol. 43, pp. 92 – 105, 2010. https://doi.org/10.1145/1713254.1713276

[5] Arora, I. and Gupta, A., “Improving performance of cloud based transactional applications using in-memory data grid,” International Journal of Computer Applications, Vol 107, pp. 14-19. 2014. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.684.9087&rep=rep1&type=pdf

[6] Bahl, B., Sharma, V. and Rajpal, N., “Boosting geographic information system’s performance using in-memory data grid,” BVICAM’s International Journal of Information Technology, Vol 4, pp. 468-473, 2012. https://www.semanticscholar.org/paper/Boosting-Geographic-Information-System%27s-using-Data-Bahl-Sharma/a2f85cc1c536169ecf9747662a95a2f87b1eb744

[7] A. Kemper and T. Neumann, “HyPer: A hybrid OLTP & OLAP main memory database system based on virtual memory snapshots,” in IEEE 27th Int. Conf. Data Eng., pp. 195 – 206, 2011. https://doi.org/10.1109/ICDE.2011.5767867

[8] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi, “H-store: A high-performance, distributed main memory transaction processing system,” Proc. VLDB Endowment, vol. 1, pp. 1496 – 1499, 2008. https://doi.org/10.14778/1454159.1454211

[9] C. Diaconu, C. Freedman, E. Ismert, P.-A. Larson, P. Mittal, R. Stonecipher, N. Verma, and M. Zwilling, “Hekaton: SQL server’s memory-optimized OLTP engine,” in Proc. ACM SIGMOD Int. Conf. Manag. Data, pp. 1243 – 1254, 2013. https://doi.org/10.1145/2463676.2463710

[10] Lee Kyu Woong, “Management of Data base replication in main memory DBMS ALTIBASE for high availability,” Journal of Internet Computing and Services, vol 6 issue 1, pp73-84, 2005. https://www.koreascience.or.kr/article/JAKO200516610543538.pdf

[11] Soo-Cheol Oh and Seong Woon Kim, “Design and implementation of high-performance virtual desktop system managing virtual desktop image in main memory,” KIISE Transactions on Computing Practices, Vol 22, No. 8, pp. 363-368, Aug, 2016. https://doi.org/10.5626/KTCP.2016.22.8.363

[12] In-Memory Storage Driver https://docs.docker.com/registry/storage-drivers/inmemory/

[13] Docker – Volumes
https://docs.docker.com/storage/volumes/

[14] Memory Optimized for Data-Centric Workloads
https://www.intel.co.kr/content/www/kr/ko/architecture-and-technology/optane-dc-persistent-memory.html

[15] "QEMU Git tree - docs/interop/qcow2.txt," qemu.org, May 29, 2018.
https://github.com/qemu/qemu/blob/master/docs/interop/qcow2.txt

[16] Jonathan Corbet, Alessandro Rubini and Greg Kroah-Hartman, "Memory Mapping and DMA," Linux Device Drivers, 3rd Edition.
https://www.oreilly.com/library/view/linux-device-drivers/0596005903/ch15.html

[17] OpenDedup – Opensource Dedupe to cloud and Local storage
https://opendedup.org/odd/overview/

[18] "QEMU System Emulation User's Guide,"
https://www.qemu.org/docs/master/system/index.

[19] Use containers to Build, Share and Run your applications
https://www.docker.com/resources/what-container

[20] Aurora, Valerie, "Union file systems: Architecture, features, and design choices," LWN.net. Retrieved 2018-01-17.
https://lwn.net/Articles/324291/

[21] GitHub: Where the world builds software · GitHub
https://github-landing-page.netlify.app/

[22] CrystalDiskMark
https://osdn.net/projects/crystaldiskmark/

[23] Kubernetes- Container runtimes
https://kubernetes.io/ko/docs/setup/production-environment/container-runtimes/

[24] NGINX Application Platform
https://www.nginx.com/products/

[25] ab - Apache HTTP server benchmarking tool
https://httpd.apache.org/docs/2.4/en/programs/ab.html

[26] AL-Harbi Fahad Jazi, Kangseok kim, and Jai-Hoon Kim, "Design and Cost Analysis for a Fault-Tolerant Distributed Shared Memory System," Journal of Internet Computing and Services, vol 17 issue 4, pp1-9, 2016.
https://doi.org/10.7472/jksii.2016.17.4.01

◐ Authors ◑

Dae Won Kim

received the B.S., M.S., and Ph.D. degree in Electrical Engineering, Kyungpook National University, Daegu, South Korea in 1998, 2000, and 2004 respectively. He joined Electronics and Telecommunications Research Institute (ETRI) in Daejeon, Korea in 2004 and he is working as a Principal Researcher. He has developed a cloud edge platform technology for ultra-low latency edge service. His research interests include cloud computing, edge computing and AI edge service. Also, he has deveopled a national standard for ITU-T SG13 as an editor and ISOIEC JTC1/SC 38.
E-mail: won22@etri.re.kr

Sun Wook Kim

received the B.S. degree from Chungbuk National University, Korea, the M.S. degree from Hanyang University, Korea in 1996 and 2001 respectively, all in Computer Science. And he received his Ph.D. degree from Korea University, Korea in 2011. He joined Electronics and Telecommunications Research Institute (ETRI) in Daejeon, Korea in 2001 and he is working as a Principal Researcher. He has developed a cloud edge platform technology for ultra-low latency edge service. His research interests include cloud computing, edge computing and AI edge service.
E-mail: swkin99@etri.re.kr

Soo Cheol Oh

received the B.S., M.S., and Ph.D. degree in Computer Engineering, Pusan National University, Seoul, South Korea in 1995, 1997, and 2003 respectively.
He had worked a researcher in LG Electronics from 1997 to 1998. He is currently a principal researcher of Electronics and Telecommunication Research Institute (ETRI) in Daejeon, South Korea. His research interests include quantum computing, cloud computing and virtual desktop infrastructure system.
E-mail: ponylife@etri.re.kr