Adaptation of Distributed File System to VDI Storage by Client-Side Cache

Cheiyol Kim[1*], Sangmin Lee[1], Youngkyun Kim[1], Daewha Seo[2]

[1] Storage System Research Team, Electronics and Telecommunications Research Institute, Daejeon, Korea.
[2] Electrical Engineering and Computer Science Department, Kyungpook National University, Daegu, Korea.

* Corresponding author. Tel.: 82428601678; email: gauri@etri.re.kr

Manuscript submitted March 10, 2015; accepted May 30, 2015.
doi: 10.17706/jcp.11.1.10-17

Abstract: Distributed File system was widely used as a cloud computing backend storage with high scalability, low cost, and reasonable sequential I/O performance. VDI I/O workload is mostly composed of small random I/O and distributed file system does not have enough performance for this I/O pattern. Compensating for this gap, we applied a cache using memory and SSD (Solid State Drive) to distributed file system. This cache was implemented in the file system's client side. It uses memory as write cache and SSD as read cache with write-back policy for minimizing the I/O response time. Based distributed file system was implemented on user-level using Fuse (File system in User space) framework.

When the client-side cache was used, the file server which previously could not support even 1024 users was improved enough to support more than 3000 users. By applying the client-side cache, it is possible to solve the performance drawback of the distributed file system used for VDI storage.

Key words: Cache, distributed file system, virtual desktop, VDI storage.

1. Introduction

As the importance of enterprise information security is emphasized, VDI becomes one of the most important IT resources. VDI technology is composed of hypervisor, graphic transfer protocol, service broker, and shared storage. Among these technologies, the storage is usually less scalable and more performance effective and cost heavy point than others. It is commonly said that the storage cost is more than 20% of total VDI cost. SAN(Storage Area Network) is the most popular stable and high performance storage. But because of the high cost and low scalability, it is not easy to adopt SAN to a VDI storage.

Distributed file system like a Google file system [1] is used for large scale cloud storage with high scalability and low cost. Glory file system [2] is also famous distributed file system which is used in enterprise cloud storage. To solve the scalability and cost problem of SAN, much efforts of using distributed file system for VDI storage are shown. Ceph [3] was started as unified object storage. Now Ceph can support both the POSIX API and block storage, thus it can support VDI. In addition to these interfaces, Ceph can also support additional VDI specific functionalities such like a snapshot and a fast clone.

Flash memory based devices like SSD or PCI-e based card like Fusion-IO [4] are used as I/O cache devices to accelerate the I/O speed of slow magnetic disk. Mercury [5] uses SSD as virtualization host server block device cache for data center environment. Mercury discusses about the effective position of cache and resolves the cache coherence problem of sharing storage by write-through policy. Mercury saves all read/write data on SSD cache. By write-through policy, it can preserve the data reliability from data loss.

S-cave [6] also supports SSD cache on virtualization host, but they proposes the new policy of cache space allocation which collects the relation of cache space change and cache hit ratio change and reflects its tendency to cache space allocation.

Because flash memory has a weak point that has a limitation of the number of data block write, flash based cache device must be considered for this lifetime. [7] is the result of the study of how to minimize the number of cache write. [7] focuses on the selection of data to insert into the cache. They do not insert all the read/write data to SSD cache directly. Instead, they temporarily store the data block in memory until the data deserves to be inserted into the SSD cache. Therefore, the cache eviction will be also decreased. It is called "Lazy Adaptive Replacement".

In this paper, we introduce the client-side cache of distributed file system for VDI storage. The client-side cache can reduce the network latency and decrease the backend file system burden. This cache consists of not only SSD but also memory. We use memory as write cache for fast response time and adopt SSD as read cache for repeated read. Different from other cache design, we use write-back cache policy. Write-back policy has a risk of data loss when the system or its network fails. Nevertheless, we choose memory as write cache for minimizing the write request's response time. Avoiding the data loss problem, we selectively adopt this write-back policy to each volume. For the valuable user data, we uses write-though policy like other SSD cache system.

The last of this paper is organized as follows. In Section 2, we introduce the user level distributed file system which is used our backend storage system. Section 3 discusses about the design of our cache. The experiments and its analysis are shown in Section 4. Finally Section 5 concludes this paper.

2. Fuse Based Distributed File System

The file system implemented at the kernel level always has kernel dependency. This kernel dependency makes a trouble when porting it to another OS or kernel. Fuse can remove this problem, because Fuse based file system is implemented on user-level without kernel dependency. Fuse deals with the kernel changes instead of the file system. Distributed file system has separated client and data server, thus Fuse based client can easily support various operating systems. S3fs [8] is another fuse based cloud file system. S3fs makes users to use S3 [9] storage service like a local disk. Ceph [3] also supports fuse based client.

In this paper, we used MAHA-FS [10] as base distributed file system. MAHA-FS is the enhanced version of Glory-FS [2].

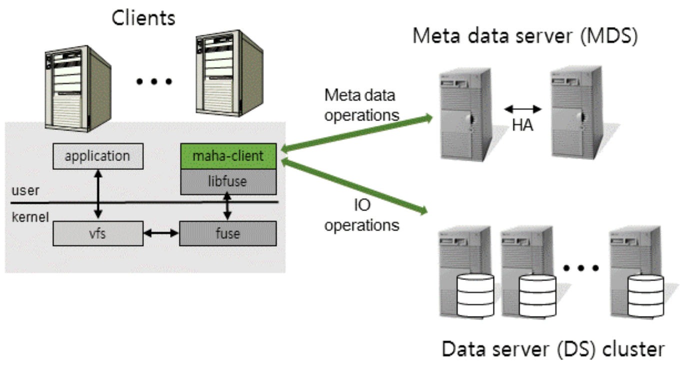

Fig. 1. System architecture of MAHA-FS.

Fig. 1 shows that MAHA-FS consists of client, MDS(Meta Data Server) and DS(Data Server). Client mounts remote MAHA-FS and uses it like local file system. MAHA-FS supports POSIX file API. Applications do not need to be modified or re-compiled to use MAHA-FS. MDS manages all metadata and DS manages I/O data service by saving file data into chunk file. Some features of MAHA-FS is described below

- Providing disk space more than petabytes
- Higher meta-data service performance in single MDS
- Its own meta-data management engine without DBMS
- Enhanced random I/O performance
- Supporting POSIX API

3. Client-Side Cache

Distributed file system has good scalability and sequential I/O performance, but it does not have good performance in small random I/O. Virtual desktop I/O access pattern is known that small random request is more than sequential request. In order to fill this performance gap, cache between client and backend storage is thought to be a one of good solution. Client-side cache is most effective when the cache hits, the request does not need to be sent to the backend storage.

3.1. Write Cache

Write cache data does not need to be hold for long time at the cache. It just diminishes the latency of backend storage writing. Low latency of memory makes we choose it as the write cache device. Despite SSD's reasonable performance and non-volatile feature, relatively high latency and write endurance problem of SSD prevent us from choosing it.

The data in the write cache is less expected to re-read. Write cache manages data size as user handled size not specific fixed size. When the write data is flushed to backend storage, it is advantageous to merge continuous data into single one and send it to backend storage. This cache data merging can reduce the number of times of network interaction between client and backend storage. When write cache inserts the data into memory cache, it checks whether the adjacent neighbor data already exists in the cache. If they are, the newly inserted data is merged with the adjacent one and the new merged one is stored in the cache.

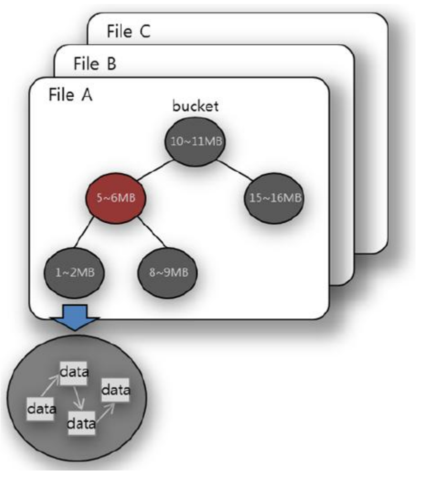

Fig. 2. Write cache data management.

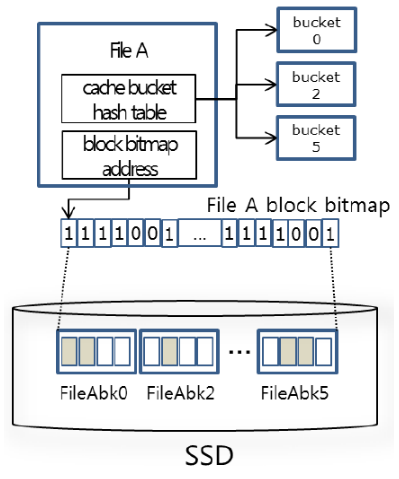

Fig. 3. Read cache data management.

The write cache data is grouped by file. In each cache file, data block is separated to bucket. Bucket is the basic unit of cache data. Each cached bucket is managed by Red-Black tree (RB-tree). A single bucket could have multiple data blocks. Data blocks in a bucket are connected by linked list. Fig. 2 describes this write cache data management unit.

In Fig. 2, bucket contains the cache data with 1MB unit, the 1~2MB bucket has four data blocks. Each data block size is not same. Data block size range is from 1bytes to maximum 1MB. Write cache management uses RB-tree. If the hash table which load-ratio is one used instead of RB-tree, the performance will be O①. Because the bucket size is so small (1MB), then load-ratio one hash table will be large and it occupies so much memory space. This memory occupation makes us to use RB-tree. RB-tree gives a performance of O(logN). We trade off the memory occupation and manipulation performance. Write cache flushes the cache data to backend storage by unit of bucket.

3.2. Read Cache

As read cache needs a large capacity of space to store cache data, SSD is the best choice. SSD is a flash memory based block device. It has short I/O latency and better data transfer bandwidth than HDD.

Read cache uses 4KB block size like a system page cache size. SSD has very larger size than system memory. Wrong cache management design will use excessive memory for manipulating read cache block. Read cache uses bucket like write cache. But the size of bucket is bigger than that of write cache. In this implementation, we fixed read cache bucket size as 10MB. The metadata of a bucket is allocated when the bucket contains a data block. When there is no data block in a bucket, that bucket metadata will be freed. For faster checking of caching, we use the bitmap of block per each cached file. For accessing the cached block after checking the existence of the block, each bucket's metadata would be referenced.

Cache eviction is executed when the cache space is lower than the specified limit. The cache eviction is also performed by bucket unit like write cache. LRU algorithm is applied to bucket eviction policy. Bucket eviction with LRU acquires the temporal locality and spatial locality together. Temporal locality can be gained from LRU algorithm. Spatial locality can be gained from bucket replacement, because oldest one cache block makes their neighbor blocks to be kicked out. Different from write cache, read cache uses hash-table with load-ratio one as their management data structure. When the bucket size of read cache is 10MB, the hash-table size is not so big enough not to make a burden. Fig. 3 shows the read cache management structure. Management of SSD is easily performed on the file system through the name of cache bucket file name. In Fig. 3, File A has cache data in bucket number 0, 2, and 5. Those buckets have each cache file in SSD. In those buckets, occupied blocks are indicated by looking at the block bitmap.

3.3. Read/Write Cache Coherence

Because of the separate read/write cache design, same position but different data could be exist in both read and write cache. To prevent this situation, we use the invalidation algorithm which invalidates read cache when the same block data is written. To minimize the burden of invalidating read cache, we just use the read cache bitmap. When the same block data was entered into the cache system, it will checks read cache by read cache bitmap. If the bitmap is set as cached, then it clears that bit of bitmap. After that, the request data is cached in write cache. While the SSD cache data block is invalidated by bitmap clean, its cache block on SSD is not removed immediately. The SSD's data removal is executed by cache eviction.

4. Experiments

4.1. Benchmark Tool

We used VDI-IOmark [11] as VDI storage benchmark tool. VDI-IOmark traces the real virtual desktop I/O and generates it to the test VDI storage. The number of users and workload pattern are configurable parameter of VDI-IOmark. The type of workload patterns are heavy, office, and boot. In this experiment, we used office workload pattern which is represents the general office worker's desktop I/O usage.

VDI-IOmark is running on the VMs (Virtual Machines). VMs consists of single master and multiple clients. Master VM throws the starting benchmark command to multiple clients and gathers all of client's results and summarizes it. Client VM generates the I/O workload and records the I/O results. When the test is done, each client sends the test result to master VM. The number of client VMs is determined by the number of users.

VDI-IOmark defines the minimum resource requirements of client VM for the proper testing. CPU, memory, and network bandwidth are those resources. We experimented satisfying the required conditions.

4.2. Experimental Environment

Our cache is implemented on the client of MAHA-FS distributed file system. The performance of distributed file system is determined by the number of backend data server. But we used a single file server with three raid controller for twenty-four SSD disks instead of many HDD disk based file servers. Table 1 shows some important server specifications.

Host server is hypervisor server running virtual machines for VDI-IOmark program. Host server uses local SSD as read cache. According to the benchmark requirements, 8 VMs need to be run on each host server. A single vcpu and 4GB memory are allocated to each VM.

Table 1. Server Specifications of Experiment

Type	Host server	File server
CPU	Intel Xeon E5620 2.4Ghz 4core * 2 socket (logically 16 cores)	Intel Xeon E5620 2.4Ghz 4core * 2 socket (logically 16 cores)
Memory	64GB	48GB
SCSI controller	On board SATA 3	LSI MegaRAID SAS 9266-8i * 3
Disks	OCZ Virtex3 SSD 240G	OCZ Virtex3 SSD 240G * 24
Storage NIC	Intel X520 (FCoE & 10G)	Intel X520 (FCoE & 10G)

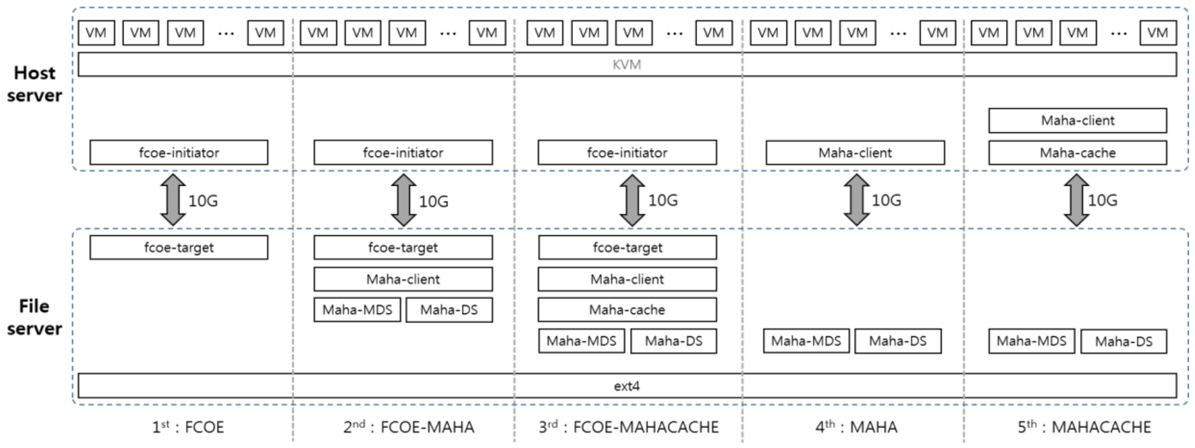

Fig. 4. SW stack of five experimental cases.

Many network storage protocols like FC and iSCSI are used to give block device interface to remote host. FCoE (Fibre Channel over Ethernet) protocol is selected to support block device interface to host server in this experiment. FCoE makes it possible to support FC on 10Gbps Ethernet networks. FCoE can support FC without TCP/IP overhead. This means that FCoE can get more reliability than iSCSI. FCoE simplifies the storage and host network card environment. Because FCoE NIC can be used as FC and 10G Ethernet also.

Intel X520 NIC card is used as FCoE and 10G Ethernet card at each experiment.

File server and host server uses Linux, CentOS-6.5. Host server's hypervisor is KVM, because it can support FUSE and software FCoE module named open-fcoe.

4.3. Experimental Cases

All test is done for 1024 office user workload of VDI-IOmark. If the percentage of response time under 50ms is more than 80%, this storage is determined to be sufficiently supportable to 1024 office users. We experimented five cases to investigate how to use this storage is most efficient.

Test consists of five cases. Each test is explained below.

① FCOE: providing block device to host server which is mapped from remote ext4 file by FCoE

② FCOE-MAHA: it is similar to "FCOE" but remote local file system was changed to MAHA-FS, MAHA client, MDS, and DS are all running in the file server.

③ FCOE-MAHACACHE : it adds MAHA-FS client-side cache to Test ②
④ MAHA: providing file system to host server as mount point by MAHA-FS
⑤ MAHACACHE: it adds MAHA-FS client-side cache to Test ④

Software stack of all test cases are shown in Fig. 4. All tests has same physical network burden because VDI-IOmark is running on VMs, VMs are on the host server which uses file system supported by remote file server.

The burden of each test case is presented in the Table 2. In the Table 2, "write-through" means that write requested data from VMs always have to go to file server directly. This makes the write operation latency is always bad. If the test uses the FCoE or FUSE for MAHA-FS. Each burden item is checked. MAHA-FS designed to operate on the TCP/IP network, so ② FCOE-MAHA and ③ FCOE-MAHACACHE uses local TCP/IP stack in file server.

Table 2. Burden of Each Experiment

Burden	Test Cases				
	①FCOE	②FCOE-MAHA	③FCOE-MAHACACHE	④MAHA	⑤MAHACACHE
Fcoe	O	O	O	X	X
Fuse	X	O	O	O	O
Local TCP/IP	X	O	O	X	X
Remote TCP/IP	O	O	O	O	X (when hit)
Write though	O	O	O	O	X

4.4. Experimental Results

Generally, when the percentage of under 50ms response time is more than 80% of the total, the VDI storage is said to have enough performance for tested the number of users. Fig. 5 and Fig. 6 show the CDF(Cumulative Distribution function) of response time of each test result. X-axis is the response time in millisecond and Y-axis is the cumulative percentage of under that response time.

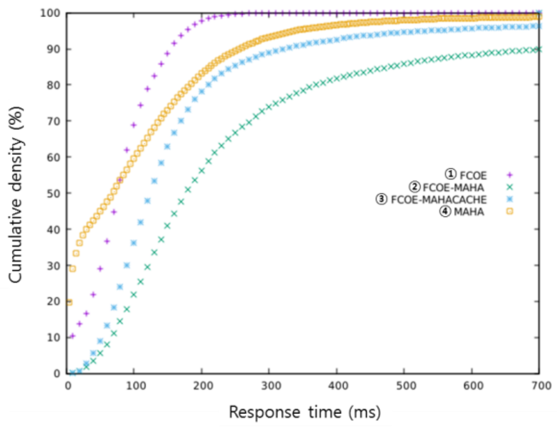

Fig. 5. Results of case ①~④.

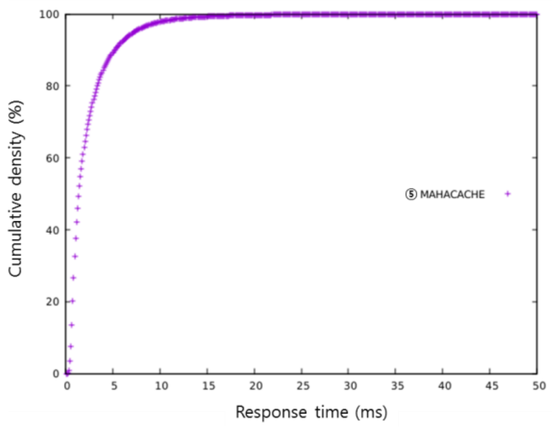

Fig. 6. Results of case ⑤.

Fig. 5 and Fig. 6 show CDF of all experiments and Table 3 shows some specific interesting results. Due to the different result of ⑤ MAHACACHE, we separated it from Fig. 5 to Fig. 6. Every cases that uses FCoE trough 10G ethernet could not support enough performance for 1024 users. ①FCOE shows better performance than ② FCOE-MAHA and ③ FCOE-MAHACACHE. In Table 2, we can see that ② FCOE-MAHA and ③ FCOE-MAHACACHE have to process additional FUSE and local TCP/IP overhead than ① FCOE. Different from ① FCOE, ② FCOE-MAHA and ③ FCOE-MAHACACHE, ④ MAHA does not use FCoE protocol, instead it directly mounts remote file server and export its files to VMs. ④ MAHA is better than ① FCOE in under 50ms, but in over 50ms, it shows worse performance result than ① FCOE. In Table 3, comparing response time of 90% of CDF of ① FCOE and ④ MAHA tells that 155ms of ① FCOE is better than 260ms of ④ MAHA.

Table 3. Specific Experimental Results

	Test Cases				
	①FCOE	②FCOE-MAHA	③FCOE-MAHACACHE	④MAHA	⑤MAHACACHE
Percentage of Under 50ms	29.03%	5.73%	8.96%	45.01%	100%
Response time of 80%	125ms	370ms	210ms	185ms	4ms
Response time of 90%	155ms	700ms	330ms	260ms	5ms
Max response time	354ms	112s	116s	2451ms	47ms

When comparing first four cases and ⑤ MAHACACHE in Fig. 6, ⑤MAHACACHE shows the dominant performance result which supports perfectly 1024 users. 100% of I/O is under 50ms, moreover 90% of it is under 5ms. We did more experiment with raising the number of users. Finally ⑤ MAHACACHE could support 3000 users keeping that 80% of response time is under 50ms.

From this experiments, we found that the client-side cache is very effective method to enhance I/O performance of VDI workload. Thus, the distributed file system could be used as VDI storage with client-side cache.

5. Conclusion

This paper proposes the client-side cache for adopting distributed file system to VDI storage. The cache was implemented in the client module of MAHA-FS, distributed file system. From several experiments, MAHA-FS with client-side cache shows the remarkable performance advance.

The proposed client-side cache uses RAM memory and SSD as cache media. And write-back policy for enhancing write response time was proper to VDI workload which has much of small random I/O.

The difference from other VDI storage caches is that this cache uses different cache media for write and read cache. This policy was determined by the each I/O's characteristics. Write request needs more fast response and read request needs bigger space for cache hit. The result of our experiments show this cache design is very effective for the VDI workload.

Though we gained the excellent performance than we expected, write-back policy still has the risk of losing data when host server crashes. To complement this weak point, the write data logging in SSD cache can be one of the candidate techniques. And in addition to the invalidation policy, write update policy of read cache can be considered as better policy for some workload situations.

Acknowledgment

This work was supported by the IT R&D program of MKE/KEIT. [10041730, A Cloud File System Development for Massive Virtual Desktop Infrastructure Service]

References

[1] Ghemawat, S., Gobioff, H., & Leung, S. T. (2003, October). The Google file system. *ACM SIGOPS Operating Systems Review, 37*(*5*), 29-43.

[2] Cha, M.-H., *et al.* (2008). Design and implementation of a consistent update method for multiple file replicas in a distributed file system. *Proceedings of the 10th International Conference on Advanced Communication Technology: Vol. 3.*

[3] Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D., & Maltzahn, C. (2006, November). Ceph: A scalable, high-performance distributed file system. *Proceedings of the 7th symposium on Operating Systems Design and Implementation* (pp. 307-320). USENIX Association.

[4] FUSION-IO. From: http: www.fusionio.com

[5] Byan, S., Lentini, J., Madan, A., Pabon, L., Condict, M., Kimmel, J., & Storer, M. (2012, April). Mercury: Host-side flash caching for the data center. *Proceedings of IEEE 28th Symposium on Mass Storage Systems and Technologies* (pp. 1-12).

[6] Luo, T., Ma, S., Lee, R., Zhang, X., Liu, D., & Zhou, L. (2013, October). S-cave: Effective ssd caching to improve virtual machine storage performance. *Proceedings of the 22nd International Conference on Parallel Architectures And Compilation Techniques* (pp. 103-112).

[7] Huang, S., Wei, Q., Chen, J., Chen, C., & Feng, D. (2013, May). Improving flash-based disk cache with lazy adaptive replacement. *Proceedings of IEEE 29th Symposium on Mass Storage Systems and Technologies* (pp. 1-10).

[8] Rizun, R. *S3fs: FUSE-Based file System Backed by Amazon S3*.

[9] Amazon Simple Storage Service (Amazon S3). From: http://aws.amazon.com/s3

[10] Kim, Y.-C., *et al.* (2013). MAHA-FS: A distributed file system for high performance metadata processing and random IO. *KIPS Transactions on Software and Data Engineering, 2(2),* 91-96.

[11] IOMAdmin. (2013). VDI-IOmark Releases New Version of VDI Benchmark. From http://www.iomark.org/content/vdi-iomark-version-202-released

Cheiyol Kim received the B.S. and M.S. degrees in the Department of Electronics Engineering, Kyungpook National University, Daegu, Korea in 1999, 2001 respectively. He worked for Electronics and Telecommunications Research Institute since 2001 as a senior researcher. He worked in the field of server system software like a storage system, virtualization, asynchronous network, and so on. Now his research area is mainly focused on storage and cache software system.

Sangmin Lee received the B.S. degree in the Department of Computer Science from In-Ha University, Incheon, Korea, in 1991. She has been working for Electronics and Telecommunications Research Institute since 1991. Her research interests are in the areas of RAID, storage system, and distributed file system.

Youngkyun Kim received the B.S. and Ph.D. degrees in computer science from Chonnam National University, South Korea in 1991 and 1995 respectively. Since 1996, he is a director of Storage System Research Team at Electronics and Telecommunications Research Institute (ETRI), South Korea. His research interests include massive high performance storage systems, distributed file systems, internet massive database systems, and non-volatile memory storage.

Daewha Seo received the B.S. degree in Kyungpook National University in 1981, the M.S. and Ph.D. degrees in the Department of Computer Science from Korea Advanced Institute of Science and Technology in 1983 and 1993 respectively. He was a researcher at Electronics and Telecommunications Research Institute from 1983 to 1995. He is now a professor at Kyungpook National University from 1995. And he is also a head of Center for Embedded Software Technology. His research interests include embedded software, parallel processing, and distributed operating system.