Performance Evaluation of a RAM based Storage System *NGS*

YunHee Kang
BaekSeok University, 115 AnSeo-dong, Cheonan 330-704 Korea

JaeHa Kung
Korea University, 5-1 Anam-dong, Seongbuk-ku, Seoul, 136-701 Korea

SeungKook Cheong
ETRI, 161 Gajeong-dong, Yusung-gu, Daejon 305-350, Korea

ABSTRACT

Recently high-speed memory array based on RAM, which is a type of solid-state drive (SSD), has been introduced to handle the input/output (I/O) bottleneck. But there are only a few performance studies on RAM based SSD storage with regard to diverse workloads. In this paper, we focus on the file system for RAM based memory array based NGS (Next Generation Storage) system which is running on Linux operating system. Then we perform benchmark tests on practical file systems including Ext3, ReiserFS, XFS. The result shows XFS significantly outperforms other file systems in tests that represent the storage and data requests typically made by enterprise applications in many aspects. The experiment is used to design the dedicated file system for NGS system. The results presented here can help enterprises improve their performance significantly.

Keywords: *Benchmark, RAM based SSD storage, File system, Performance Evaluation*

1. INTRODUCTION

There are some application areas like multimedia, data warehouse, and real-time data capturing, which are to guarantee I/O performance to ensure desired throughput including Input Output Per Second (IOPS) and data transfer rate. But the I/O performance of storages like disk is relatively lower than the processing speed of CPU[1].

On-chip shared L2 cache architectures are common in today's multi-core processors. Shared caches have important advantages such as increased cache space utilization, fast inter-core communication via the high-speed shared L2 cache, and reduced aggregate cache footprint through the elimination of undesired replication of cache lines.

The effective average rotational delay may be lower on some disks when most of blocks are being read on one track. The time to perform these seeks is included in the overhead utilization so that the I/O performance can be achieved from the reserved disk. Especially the increasing prevalence of I/O-intensive applications mentioned above has placed growing pressure on computer storage systems.

A solid-state drive (SSD) is a kind of data storage devices that use solid-state memory to store persistent data. A SSD emulates a hard disk drive interface, thus easily replacing it in most applications. With no moving parts, SSDs are less fragile than hard disks and are also silent (unless a cooling fan is used). However, most of the performance analysis that has driven processor and system design has been directed towards application-level performance as quantified by the SPEC benchmarks. Relatively little research has been conducted successfully on guaranteeing storage performance for mixed workloads on RAM based storage device, a kind of SSD. By using RAM based SSD, volatile memory can be used to improve file system performance in many ways. For one, it can help to store all of the file system metadata, which is used for book-keeping information about data. This is an attractive feature, but it is a new approach in file system design. Also, it has not been known yet just how much can be gained from it[2], [3].

The goal of this paper is to provide a partial answer to the question of how to exploit main features and guidelines for designing a file system for NGS(Next Generation Storage) system which has a storage as RAM based memory array by comparing the performance of three of the more popular file systems with journaling available under Linux: Ext3, ReiserFS, XFS and SpadFS. To consider two requirements of a file system, performance and reliability, we try to evaluate the I/O performance with recovery mechanism and analyze the effects of I/O performance parameters at the file system level. In this experiment, we also introduce a file system SpadFS which has new recovery mechanism, crash

* *Corresponding author. E-mail : yhkang@bu.ac.kr*
Manuscript received Oct.08, 2009 ; accepted Nov. 19, 2009

count to maintain consistency across crashes.

For this work, we apply a macro-benchmark tool, PostMark, to measure transaction processing time and data transfer rate with regard to workloads. We discover that benchmarks need to consider L2 caching effects to provide proper understanding into file system performance. This paper also gives a number of recommendations on how to design file system in the memory array storage.

In the remainder of this paper, we first provide a brief overview of the file systems we are testing. We then describe the benchmarks we will use in this study. Next, we describe the machines we used to run the benchmarks and the rules that define how the benchmarks are run. Finally, we present the results of our experiments.

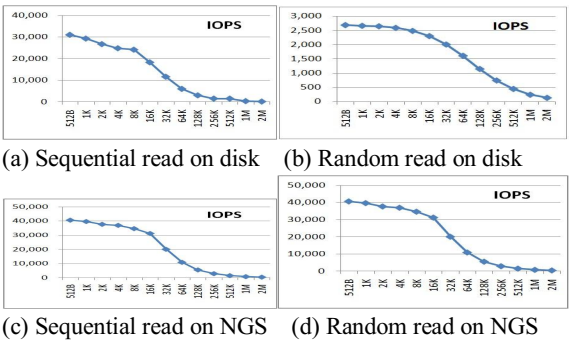
2. RELATED WORKS

2.1 RAM based memory array

There are two types of SSD which are Flash memory based SSD and RAM based SSD. Flash memory based SSDs have some characteristics like non-volatile and low power consumption. These flash memory-based SSDs do not require batteries. These are slower than but may perform better than hard drives with regard to reads because of negligible seek time.

SSD based on RAM is characterized by fairly fast data access, generally less than 0.01 milliseconds, and are used primarily to accelerate applications that would otherwise be held back by the latency of Flash SSDs or traditional HDDs [1], [2]. RAM-based SSDs usually incorporate internal battery and backup storage systems to ensure data persistence while no power is being supplied to the drive from external sources. If power is lost, the battery provides power while all information is copied from RAM to back-up storage.

Figure 1 shows the performance result with regard to sequential and random read between disk based storage and NGS system on a SAN environment. Hereto we called NGS as the RAM based storage. NGS system has a main characteristic that sequential and random performance values are almost identical. With respect to IOPS, disk read performance only has 10 % of NGS system. Especially there is no difference of performance between sequential read and random read on NGS system.



(a) Sequential read on disk (b) Random read on disk

(c) Sequential read on NGS (d) Random read on NGS

Fig. 1. Performance comparison with disk and NGS

2.2 File system

A file system is an organization of data and metadata on a storage device. A file system is a method for storing and organizing computer files and the data they contain to make it easy to find and access them. The file system implements the storage abstraction on top of the storage peripherals attached to the host computer-typically one or more hard disks [7]. It also exports an interface, allowing clients to read, write, and manipulate files [3], [4].

File system operations can broadly be divided into two categories, data operations and meta-data operations. Data operations act upon actual user data, reading or writing data from/to files. A metadata represents internal structure of file system. It characterizes the feature of the file systems and affects the I/O performance. Meta-data operations modify the structure of the file system, creating, deleting, or renaming files, directories, or special files [5].

Virtual File system (VFS) provides a uniform interface to all specific file systems supported by the UNIX kernel [3]. The kernel dispatches file system operations to the appropriate file system implementation based on pointers stored in its data structures. Linux manages to support multiple disk types in the same way as other UNIX variants. The idea of the VFS is to put a wide range of information in the kernel to represent many different types of file systems; there is a field or function to support each operation provided by any real file system supported by Linux. Figure 2 shows the architecture of file system on Linux.

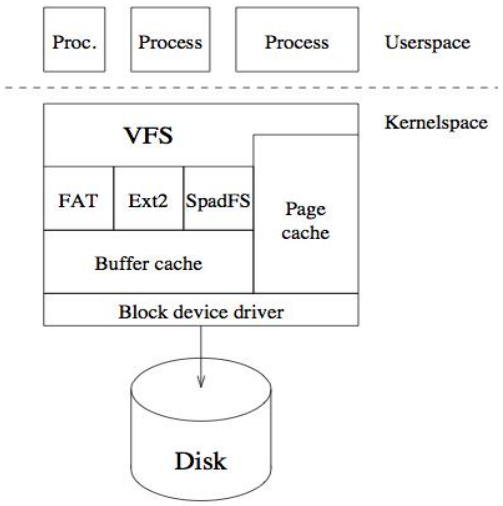


Fig. 2. File system architecture

A file system for high IOPS needs extremely fast throughput in many areas. Likewise, reliability of a file system is critical, and so a file system that supports journaling may be a requirement.

A journaling file system uses a separate area called a log or journal. Before metadata changes are actually performed, they are logged to this separate area. The operation is then performed. If the system crashes during the operation, there is enough information in the log to "replay" the log record and complete the operation [6].

File systems update their metadata by synchronous writes. Each metadata update may require many separate writes, and if the system crashes during the write sequence, metadata may be in inconsistent state. As shown in Figure 3, journaling file systems are fault-resilient file systems that use a journal to log changes before they're committed to the file system to avoid metadata corruption.

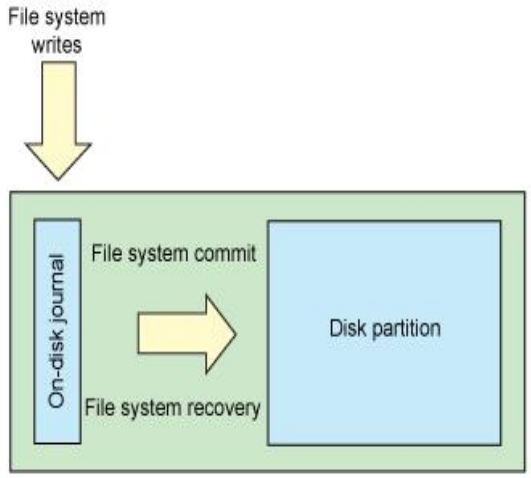


Fig. 3. Journaling file system

2.3 Specific file systems

In Linux operating system, file systems can be plugged into the kernel through a well-defined interface that is currently added to the kernel. As each file system is initialized, it registers itself with the VFS. This happens as the operating system initializes itself at system boot time. The real file systems are either built into the kernel itself or are built as loadable modules.

Ext3 is an enhancement of Ext2 developed by Stephen Tweedie, Ext3 uses the same disk format and data structures as Ext2, but in addition supports journaling. The default journaling mode, ordered, guarantees consistent writing of the metadata, descriptor and header blocks. This makes conversion from Ext2 to Ext3 extremely easy. It is block based, with sequential filename directory search [8].

ReiserFS is developed by Hans Reiser. It supports metadata journaling, and is especially noted for its excellent small-file performance. ReiserFS uses B* balanced trees to organize directories, files and data. This provides fast directory lookups and fast deletes operations. ReiserFS supports a space-saving option called tail-packing that packs small files into the leaves of the B* tree [11].

Released in May 2001, XFS is a journaling file system that supports metadata journaling. The high level structure of XFS is similar to a conventional file system with the addition of a transaction manager and a volume manager. XFS uses allocation groups and extent-based allocations to improve locality of data on disk. It contains a feature allocation group to support for parallel I/O. Each allocation group has its own partition and maintains files and directories. This results in improved performance, particularly for large sequential transfers [10]. XFS meets the requirements for large files systems, files, and directories through the following mechanisms:

- B+ tree indices on all file system data structures
- tight integration with the kernel, including use of advanced page/buffer cache features, the directory name lookup cache, and the dynamic vnode cache
- sophisticated space management techniques which exploit contiguity, parallelism, and fast logging

SpadFS [12] is a new file system, designed by Mikulas Pato ka, which brings features like crash recovery for fast recovery, indexed directories with Fagin's extendible hashing, etc.

3. EVALUATION TOOL AND WORKLOAD PARAMETERS

Benchmarking is the process of running a specific program or workload on a specific machine or system and measuring the resulting performance[14], [15]. This technique clearly provides an accurate evaluation of the performance of that machine for that workload.

Postmark is a macro-benchmark designed to use the file system as an e-mail server does, generating many metadata operations on small files. The benchmark we have chosen for this paper is postmark that was designed by Jeffrey Katcher to model the workload seen by Internet Service Provider under heavy load[14]. Specifically, the workload is meant to model a combination of electronic mail, net-news, and web-based commerce transactions [9].

The goal of macro-benchmarking is to demonstrate the impact of meta-data operations for several common workloads. Since there are an infinite number of workloads, it is not possible to characterize exactly how these systems will benefit all workloads[15]. The main function of macro-benchmark programs is to measure how real workloads perform on a file system. Thus many macro-benchmarks consist of executing some application with carefully specified parameters.

The advantage of the benchmark is as follows:

- supports a wide variety of workloads instead of the specific workload implemented by Bonnie
- is not trace driven so that they are readily scalable from small to large systems
- produces significant load on underlying file systems

To accomplish this, PostMark creates a large set of files with random sizes within a set of range. The files are then subjected to a number of transactions. These transactions consist of a pairings of file creation or deletion with file read or append.

We initially ran our experiments using the pre-defined configuration with regard to level of workload as shown in Table 1. In our experiments we set the size of file generated randomly in which range is from 512 byte to 32768 byte, and set the number of subdirectories to 10. The size of read and

write is varied from 512 byte to 8192 byte.

Table 1. Workload parameter

Workload types	The number of files	The number of transaction
Low	1000	50,000
Medium	20,000	50,000
Large	20,000	100,000

4. EVALUATION

This section describes the hardware platforms, operating system and software versions, and benchmarks used for testing efforts, as well as the results observed.

4.1 Experimental Platform

We perform benchmark tests on practical file systems, Ext3, XFS, ReiserFS, SpadFS, which are built by kernel module. For evaluating I/O performance, we get the result of I/O performance that each of three kinds of partitions set and conduct experiments of them. The memory array is emulated as logically SCSI type disk. Target file system is created and built as a single partition.

Table 2. Experimental platform

Component	Specification
CPU	Intel(R) Xeon(R) CPUE5472@ 3.00GHz 64 bit Xeon QuadCore × 2 L1 I cache: 32Kbyte L1 D cache: 32Kbyte L2 cache: 6144Kbyte × 2
Main memory	FBDIMM DDR2 1GB(PC2-6400) × 8
Process Bus	FSB 1600MHz
OS Disk	Seagate S-ATA2 500GB, 7200 RPM 16 MB disk buffer
SSD Device	Jetspeed 256G
External Data Bus	PCI Express 2.5Gb/s:Width × 4

On the NGS, file systems we built on partitions of the disks are mounted, and then we begin experiments. The disk of NGS named Jetspeed is used to support I/O operations SCSI emulator by dedicated device driver. Table 2 describes the platforms on which the testing was conducted. Postmark version 1.5 is used on the platform. First, there is no such thing as a standard configuration. In addition, different workloads exercise the system differently and results across research papers are not comparable.

4.2 Experimental Results

We conduct the experiment to measure the performance of different levels of journaling on the ext3 file system which can be specified as mount options. In this experiment, we examine a trade-off between data integrity and performance. The problem appears if there is a system crash of electric outage before the modified data in the cache have been written to disk. In the design of NGS file system, journaling is considered an essential part to recover the system crash.

In this experiment, according to journaling types, we do I/O performance tests on Ext3 file system. In writeback mode, only the metadata is written to the journal, and the data blocks are written directly to their location on the disk. This preserves the file system structure and avoids corruption, but data corruption can occur.

To solve this problem, you can use ordered mode. Ordered mode is metadata journaling only but writes the data before journaling the metadata. In this mode, data and file system are guaranteed consistent after a recovery. Finally, journal data can also be supported. In journal data mode, both metadata and data are written to the journal. This mode offers the greatest protection against file system corruption and data loss but can suffer from performance degradation. We focus on journaling time and do not consider partial failure due to kernel failure and rollback overhead.

With regard to journal types, transaction processing time is shown in Figure 4. In low workload, the higher the number of transaction, the more the I/O performance we get compared to the number of file. The delayed write is 4.5 times faster than journal data.

It is considered that a sequential write somehow warms up the L2 cache locality for all file systems we tested. L2 caching tries to handle busty allocations in a different manner. The effects of memory and L2 caching are easily visible at the application level, beyond file system micro-benchmarks.

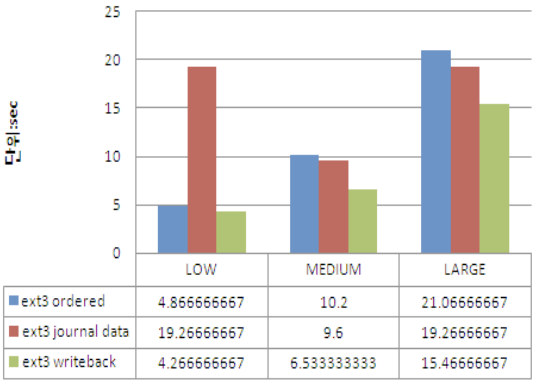
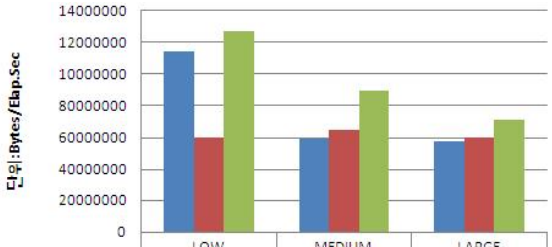


Fig. 4. Elapse time of transaction processing

Figure 5 shows data bandwidth which is the data transfer rate, when write operation is run. As is indicated in this figure, delayed write outperforms other methods in the overall workload. We may consider the effects of memory, and L2 caching is also viable at this experiment.



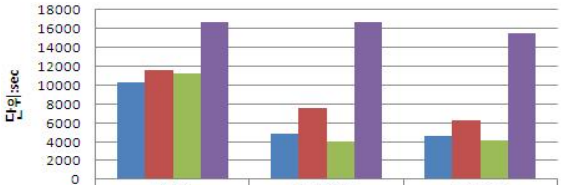
	LOW	MEDIUM	LARGE
ext3 ordered	114542470.4	59087319.47	57286665.87
ext3 journal data	60061297.87	64697965.33	60061297.87
ext3 writeback	127474684.8	89927557.87	71565673.6

Fig. 5. Data bandwidth of write operation

4.3 Comparison to file systems

Figure 6 through Figure 8 show the result of the I/O performance of the specific file systems which is measured in terms of the number of transactions per second, data bandwidth of read and write operations.

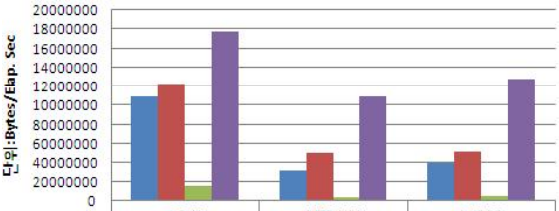
Figure 6 shows that spadfs file system outperforms other file systems. It is considered that spadfs adopts crash count for failure recovery, which helps to reduce overhead of journaling. ReiserFs has better performance than Ext3. XFS and ReiserFS outperform other file systems in all kinds of workload.



	LOW	MEDIUM	LARGE
ext3	10333.33333	4914.141333	4751.612
xfs	11666.66667	7630.952	6405.93
resiserfs	11333.33333	4127.818	4227.056667
spadfs	16666.67	16666.67	15555.55533

Fig. 6. The number of transaction per second

As shown in Figure 7, XFS has 1.1 times more data bandwidth than Ext3 and 7.7 times more data bandwidth than ReiserFS.

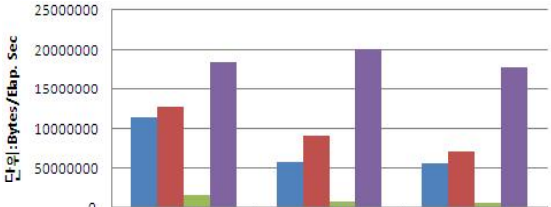


	LOW	MEDIUM	LARGE
ext3	110181721.6	32264647.73	41080712.53
xfs	122621595.2	50382677.07	52116539.47
resiserfs	15944863.47	4461984.7	5198398.1
spadfs	177712464	109305891.7	127730336

Fig. 7. Read data bandwidth

As shown in Figure 8, the I/O performance when writing data gives the pattern which is close to that of read operation.

In our tests, we observed XFS to be generally quite speedy. XFS tries to cache as much data in memory as possible, and only writes things out to disk when memory pressure dictates that it do so. In contrast, when ext3 (in "data=ordered" mode, the default) flushes data to the drive, depending on the I/O load, it can result in a lot of additional I/O access. The results show that XFS is the best file system to use for manipulating large files.



	LOW	MEDIUM	LARGE
ext3	114542470.4	59087319.47	57286665.87
xfs	127474684.8	92267467.2	72676021.87
resiserfs	16616160.8	8075819.933	7213069.5
spadfs	184745920	200175502.9	178118752

Fig. 8. Write data bandwidth

5. DESIGN CONSIDERATIONS OF DEDICATED FILE SYSTEMS FOR NGS

NGS system is designed for stand-alone storage system as well as storage pool based system. In the storage pool, network-attached storage devices provide ways to build large scalable file systems that distribute data and load across multiple network nodes. It further simplifies storage management and lets users take full advantage of the partitioning and heterogeneous storage virtualization capabilities of the integrated storage platform.

A file system for NGS is responsible for the structure and control of file storage. It is in fact an important component of any operating system. It is necessary for all the data to be stored in a persistent manner and, therefore, the most reliable medium for persistent storage has been the disk. In addition to being inexpensive, it has a large storage space and it can retain stored data even when the power is shut off.

Because of the increasing gap between processor speed and disk latency, file system performance is largely determined by its disk behavior. To design NGS in the perspective of I/O performance, we consider how to minimize the number of file operations performed by I/O benchmark tests. It is important to defer any I/O operations until the point that the application actually needs the data. The primary cache of file data, called the buffer cache, contains copies of recently accessed data blocks. To optimize performance, a file system attempts to overlap I/O requests with application computation.

By pre-fetching blocks from the disk before applications request them, the system tries to avoid forcing applications to stall while waiting for data from the storage. The caching area is managed by the existing OS kernel polices, to which we make little change for the sake of generality. By performance parameter determined, several small I/O

transfers are grouped into one large transfer. Due to RAM media characteristics there is basically no performance gap between sequential I/O and random I/O in the NGS. It is best to use the preferences system to capture only user preference and not data that can be inexpensively recomputed.

6. CONCLUSION

Many applications strongly require I/O performance guarantees. But relatively little successful research has been conducted on guaranteeing storage performance for mixed workloads. To design a file system for NGS system, we consider a file system which is running on Linux. We performed benchmark tests on practical file systems and then got the result of I/O performance. It may be possible to improve the NGS system in terms of file system, which handles the burst of I/O requests. This paper sketched the file system for NGS system that is composed of RAM based disks. In this paper the design guideline is presented. Our goal is to provide a shared storage system for virtualization with loosely coupled NGS systems. To improve I/O performance, it needs to apply the result of performance evolution to extend device driver.

ACKNOLWLEGEMENT

This work was supported by the IT R&D Program of MKE/KEIT [2008-S-037-02, NGS (Next Generation Storage) System Research & Development]

REFERENCES

[1] Agrawal, N., W. J. Bolosky, et al. “A five-year study of file-system metadata,” *ACM Trans. on Storage, vol. 3, no. 9,* 2007.
[2] Solid Data Systems, "Impact of Solid-state disk on high-transaction rate databases," *Solid data systems, Inc. White paper*, 2005.
[3] TMS, "Increase Application Performance with Solid State Disks", *TMS white paper*, 2008.
[4] Kleiman, Steven R. “Vnodes: Architecture for Multiple File System Types in Sun UNIX,” *Proc. the Summer 1986 USENIX Conference, Atlanta*, 1986.
[5] http://tldp.org/LDP/tlk/fs/filesystem.html
[6] Dominic Giampaolo, *Practical File System Design with the Be File System, Morgan Kaufmann Publishers*, 1999.
[7] Zhang Zhihui and Ghose Kanad, “yFS: A Journaling File System Design for Handling Large Data Sets with Reduced Seeking,” 2003, pp. 59-72.
[8] M. K. McKusick, W. N. Joy, S. J. Leffler, R. S. Fabry, “A Fast File System for UNIX,” *ACM Transactions on Computer Systems , vol. 2*, 1984, pp. 181-197.
[9] Ext3, http://en.wikipedia.org/wiki/Ext3
[10] Jeffrey Katcher, “PostMark: A New File System Benchmark,” *TR3022, Network Appliance Inc. Oct.*, 1997.
[11] A. Sweeney , “Scalability in the XFS File System,” *Proc. the Usenix 1996 Technical Conference*, 1996, pp. 1-14.
[12] Namesys, ReiserFS, http://en.wikipedia.org/wiki/ReiserFS
[13] M. Pato ka, Spadfs, http://artax.karlin.mff.cuni.cz/~mikulas/spad
[14] J. Katcher, “PostMark: A New File System Benchmark,” *Technical Report 3022, Network Appliance*, 1997.
[15] A. Brown and M. Seltzer, “Operating System Benchmarking in the Wake of Lmbench: A Case Study of the Performance of Netbsd on the Intel X86 Architecture,” *Proc. 1997 ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, June 1997, pp. 214-224.

YunHee Kang
He received the B.S., M.S in computer engineering from Dongguk university, Korea in 1989, 1991 respectively and also received Ph.D. in computer science from Korea university, Korea in 2002. Since 2000, he has been an assistant professor at the division of information and communication, Baekseok University, Korea. His research interests include performance evaluation in storage systems, grid computing, SOA and fault-tolerance in distributed systems.

JaeHa Kung
He is in the learning of Electrical Engineering at Korea University, Korea. His research interests are low-power and low-leakage circuits and their high performance design.

SeungKook Cheong
He received Ph.D. in electronics and information communication engineering from Hannam University, Daejon in 2004. Since 1985, he has been a principal researcher at Broadband Convergence Network Research Division, ETRI, Korea. His research interests include grid computing, utility computing and solid-state disk.