Scalable Computing: Practice and Experience
Volume 10, Number 1, pp. 87–98. http://www.scpe.org
ISSN 1895-1767
© 2009 SCPE

VIRTUAL LARGE-SCALE DISK BASE ON PC GRID[*]

ERIANTO CHAI, KATSUYOSHI MATSUMOTO, MINORU UEHARA AND HIDEKI MORI[†]

Abstract. With the recent flood of data, one of the major issues is the storage thereof. Although commodity HDDs are now very cheap, appliance storage systems are still relatively expensive. As a result, we developed the VLSD (Virtual Large-Scale Disk) toolkit to assist in the construction of large-scale storage using only cheap commodity hardware and software. As an experiment in using the VLSD toolkit, storage was created for an educational environment. This paper presents an evaluation of the performance of the storage by performing a GCC build on the trial production storage system both during trouble-free operation and failure.

Key words: virtual large-scale disk; grid; open source; raid; fault tolerance

1. Introduction. Large-scale storage has become a necessity as we are flooded with information. Large-scale storage is needed for research in areas involving large amounts of data, such as the Human Genome Project or calculation engineering. In addition, a large amount of data is used privately for video and music storage, etc. Virtual machines such as VMware and Microsoft Virtual PC 2004 allow students to learn how to maintain Linux servers safely and are very useful in higher education, but they have large disk space requirements. VMware recommends 8GB as the HDD capacity for running Linux and 16GB for Windows. In our experiments, we found that running Windows actually requires 24GB HDD capacity. This means that 40-120TB HDD capacity is needed to support 5000 students (Figure 1.1).

To service this requirement, we considered an appliance file server, with 70TB capacity. The quote for such a file server was 250 million yen (about 2 million dollars). However, we already have 500 PCs. If we added 180GB HDDs to each PC at a cost of 150 dollars per HDD, the 70TB distributed file server could be realized at a cost of just 75 thousand dollars. This represents a cost ratio of 1:33, making the cost saving more important than any other factor such as performance, dependability or maintainability. Thus we proposed a distributed storage system and have developed a toolkit that can assist in constructing such a system. We call this the VLSD (Virtual Large-Scale Disk) toolkit. [3], [4].

When we set about constructing a large-scale storage system, we tried mounting several isolated disks using a distributed file system such as SMB/CIFS or NFS. There are, however, two fundamental problems with this file system based approach. The first problem is not being able to utilize the full storage capacity. For example, when we construct 70TB of storage using 180GB HDDs, the maximum size of any file must be less than 180GB. The total size of all files in a directory must also be less than 180GB. In addition, for some OSs it is impossible to create a file greater than 4GB in such a physical file system. The second problem is linked to the Linux OS platform. Almost all distributed file systems are based on the NFS protocol, which is not suited to Windows. Windows clients have a larger capacity for HDDs than a Linux server. Thus a file system based on a distributed storage system such as NFS can neither stripe physical storage nor concatenate disks. We therefore used a disk based approach, in which a virtual disk was created in a distributed physical file system.

VLSD can concatenate multiple virtual disks, thus increasing the storage capacity. VLSD also supports typical RAID classes such as RAID0, 1, 4, 5, and 6. It can construct multiple hierarchies of RAID combining any RAID class with any other RAID class.

VLSD is independent of platform because it is written in 100% pure Java. Using a VLSD, we constructed a trial 70TB storage system consisting of 484 PCs, with each PC providing 180GB free disk space. This system realizes an acceptable MTTF using RAID66, which is 2 layered RAID6.

In an educational environment storage system, a great number of small files tend to be generated. This corresponds to the Small Read and Small Write on the VLSD system. The performance of the storage built using a VLSD was evaluated both during normal operation and at times of failure. During disk failure, performance drops for the Small Read as illustrated by the graph in Fig. 3.9. This paper therefore, evaluates performance by performing a GCC build on the trial storage system during times of both trouble-free service and failure.

This paper is organized as follows. Section 2 summarizes related works. Section 3 describes the VLSD, while Section 4 explains the build and evaluation of GCC. Finally, we present our conclusions.

[*]This research was assisted by department laboratory grant: "Research(195000066) of high trust by the PC grid and highly effectivedecentralized virtual storage".

[†]Department of Information and Computer Sciences, Toyo University, 2100 Kujirai, Saitama, 3508585, Japan

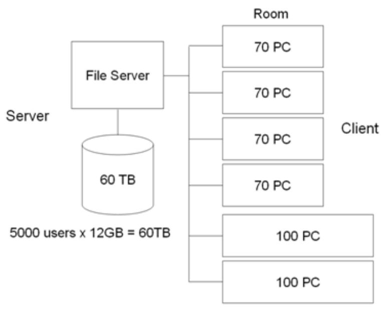


FIG. 1.1. University PC classroom

2. Related Work.

2.1. RAID. RAID (Redundant Arrays of Inexpensive Disks) [1] is used to raise the reliability of large-scale storage. It is a technique that increases the performance and fault-tolerance of data storage. These goals can be combined and are achieved by distributing data and creating data redundancy for recovering data on two or more hard disk drives. RAID levels are designated by integers from 0 to 6 each denoting a different structure of data redundancy and distribution.

RAID0 is striping without redundancy and thus capacity is increased with the addition of drives. RAID1 implements mirroring. Capacity therefore does not increase with the addition of a second drive. In a RAID2 array, a disk is made to distribute each bit and data errors are corrected by ECC. Since there is little improvement compared with a parity based system, RAID2 is hardly used. RAID3 uses parity to correct errors and carries out striping per bit or per byte. Usually, parity is saved on a separate disk. RAID3 and RAID4 differ only in the point at which striping is performed. Both record parity on a separate disk, which can become a bottleneck. RAID5 distributes where parity is saved. Since a bottleneck does not occur, performance is high. RAID5 is however, susceptible to mistakes where an operator removes the drive under drive, causing two drives to fail simultaneously. RAID6 is an extension of RAID5 and can sustain a double failure by copying parity data twice in two separate ways. Under RAID6 it is possible to recover without losing data even if two disks fail at the same time.

RAID arrays are classified into software (SW) and hardware (HW) systems. HW RAID is predominant, as SW RAID is considered to have lower performance than HW RAID and adds load to the CPU. However, the latest CPUs have higher performance than a HW RAID controller. CPU control of SW RAID is effective in machines such as file servers that can concentrate on the RAID operation. Moreover, SW RAID is the only option for implementing RAID across a network.

2.2. RAID6. There are two types of parity in RAID6, namely P parity and Q parity. [5] P parity is generated by XORing the block data of all the disks, while the Q parity used by RAID6 is more complicated to calculate. Generation of Q parity is by a Galois Field (GF) operation. Although a GF is called a limited object in algebra, it is also called a Galois object or region in computer-related fields. A GF is a set of values containing a finite number of elements. The GF of a 2^8 piece element is denoted as GF (2^8), and contains an integer element from 0 to $2^8 - 1$.

There are two types of parity in RAID6 (P parity and Q parity) [5]. P parity is generated from XORing the block data of all the disks. Q parity used by RAID6 is more complicated to calculate than P parity. Generation of Q parity is by Galois Field (GF) operation. Although a GF is called a limited object in algebra, it is also called a Galois object or region in computer-related fields. A GF is a set of values containing a finite number of elements. The GF of a 2^8 piece element is denoted as GF (2^8), and there is an element from an integer 0 to 2^{8-1}.

Four cases can be considered when two disks fail in RAID6: P and data drive failure, Q and data drive failure, P and Q drive failure, and failure of two data drives.

2.3. NBD. NBD [6] is a virtual disk system that can be compiled into the Linux kernel. The block device that this remote server offers can be handled as a local drive by clients. An arbitrary file system can be constructed in a high-order layer because it inputs and outputs at the block level. ISCSI etc. offers similar block level I/O. Because NBD has been adopted by Linux as standard it is far simpler to install compared with the alternatives. Moreover, mounting the protocol is simple and easy. The proposed system, however, does not depend on NBD. It should be noted that NBD is merely one of the available choices.

2.4. Samba. Samba [7] is used to access the file system on Linux, UNIX and other similar operating systems from Windows clients. Samba is free software that can be used to mount network shares created using Microsoft Corporations NetBIOS protocol. File and print services can also be provided to Windows clients by UNIX-like operating systems such as Linux, Solaris, and FreeBSD using Samba. The first version was developed by Andrew Tridgell, and it was opened to the public under the GPL in 1992. In this research, Samba is used to transmit requests from Windows clients to NBD.

2.5. NFS. NFS [8] enables directories and files to be shared with other machines via a network. Users and programs can access files on remote systems using NFS in the same way as they would a local file. NFS is a de facto standard distributed file system in UNIX. In Linux NFS is supported as standard. Moreover, Windows can also employ NFS using the Service for UNIX (SFU). However, operations that use SFU are more difficult than operations that use Samba. Therefore, NFS is not used by the client OS to connect to the network in this research. However, it is used to transmit the requests from Linux clients to NBD.

2.6. XFS. XFS [9], widely recognized as a highly efficient file system, offers quick restoration from system crashes and support for very large disks. XFS was the first journaling file system for Linux, and has built a strong track record in production environments because the second half of 1994. XFS makes it possible to control a huge file system using a 64-bit file system or allocation group. Handling large files presents problems both in securing the writing/reading area, and in the time taken to retrieve an empty area on the disk. In XFS, this is handled by a delay allocation and B+-Tree. When the resources on an XFS disk run short they can be extended; the logical volume can be expanded and the file system can easily be resized, without unmounting. XFS is also efficient, in that it does not generate unnecessary disk activity.

3. VLSD. VLSD is a toolkit for constructing large-scale storage systems. It is written in pure Java and can therefore be used on any platform on which Java runs, such as Windows, Linux, etc. VLSD can even be used if there is no native NBD server in the OS, as the toolkit includes software for RAID and NBD. VLSD contains typical RAID classes and NBD client/server software, thus allowing the combination of any RAID class and NBD devices with one another.

3.1. Class of VLSD. From a software perspective, VLSD consists of the following classes.

- **NBDServer**: The NBD server is called by the client, and offers empty capacity to the storage system as a virtual disk file. The OS of the client can be either Windows or Linux. Since the NBD server is mounted using Java, it is platform independent. Moreover, two or more disks may be added to the array by a single client. FAT32 may also be used despite its 4GB maximum file size restriction. Because a virtual disk created using 120GB drives cannot be used as a single file, a virtual disk is created by bundling two or more files and combining them with RAID0 or JBOD as described later.
- **DiskServer**: Is an interface to the disk server.
- **DiskServerImpl**: Is an implementation of the disk server, and a remote disk using RMI.
- **Disk**: Is an interface to a virtual disk.
- **AbstractDisk**: Is a class of an abstract virtual disk that defines the constants and methods that are used in low order classes.
- **DiskArray**: RAID1 is mounted by the base class of the disk wrapper that consists of two or more disks.
- **RAID**: The base class for RAID. Mounting of RAID1 is inherited from DiskArray.
- **SingleDisk**: Is the base class of the disk wrapper which consists of a single disk.
- **PagedDisk**:Is a wrapper for an arbitrary disk accessed via disk paging. For the wrapped disk, read/write is per page only. The fraction on each page is disregarded.
- **VariableDisk**: Is a disk with changeable capacity that does not direct resources beforehand, but can have resources secured dynamically if necessary.
- **RemoteDisk**: Is a remote disk, where the disk server is accessed.
- **RAID0**: RAID0 is used to increase capacity. It differs from JBOD, described below, in that striping is performed. Though the performance is good, adding additional disks only adds capacity equal to that of the smallest drive in the array. For example, striping drives of 100GB, 120GB, and 160GB results in 300GB only 100GB x 3. It is better to use JBOD when aiming purely to increase capacity, although

RAID0 can be expected to improve performance. In some file systems, the super block that manages i-nodes concentrates disk activity in a specific area. In such a case, striping has the effect of distributing this load. RAID0 performs per byte striping.

- **RAID5**: Parity is distributed and stored on each disk. The disk where parity is stored is different for each block.
- **RAID6**:The RAID6 class allows the generation of the GF table, block unit striping, and distributed parity to be performed.
- **JBOD**: This class does not provide redundancy like RAID0 and is used to increase capacity only. Because striping is not performed, capacity is simply summed. For example, the total capacity is 380GB if 100GB, 120GB and 160GB drives are connected. JBOD, though, lacks the load-balancing effect of RAID0 as described above. Since caching effectively creates a certain amount of scale, RAID0 performance may be superior.

Table 3.1 gives the required time for each class to do small read and small write operations, while Table 3.2 gives the required time to do large read and large write operations. The experiment is conducted on one machine. The block size of one disk stripe is 8KB and we executed 105 loops for small read, 103 loops for small write, 104 loops for large read and 102 loops for large write using random positions on the disk. As shown in Table 3.1, JBOD is faster than RAID0 on one machine. However, RAID0 can be faster than JBOD when executing on two or more machines.

Table 3.1
Required Time of each class on Small Read and Small Write

Class	Small Read (s)	Small Write
FileDisk	0.97	0.86
VariableDisk	1.09	0.94
JBOD (2 disks)	1.44	1.10
PagedDisk	2.03	1.22
SingleDisk	1.00	0.91
RAID0 (2 disks)	2.08	1.03
RAID1 (2 disks)	1.00	1.42
RAID5 (3 disks)	2.13	1.42
RAID6 (4 disks)	2.14	1.67

Table 3.2
Required Time of each class on Small Read and Small Write

Class	Large Read (s)	Large Write
FileDisk	-	-
VariableDisk	-	-
JBOD (2 disks)	-	-
PagedDisk	-	-
SingleDisk	-	-
RAID0 (2 disks)	0.52	0.17
RAID1 (2 disks)	0.28	0.17
RAID5 (3 disks)	0.52	0.36
RAID6 (4 disks)	0.50	0.59

3.2. Trial Production of the Large-Scale Storage Using VLSD. In this section we discuss the design of a 70TB distributed storage system in the environment described in Section 1. This system comprises 484 client PCs each of which provides 180GB free disk space. The total free space is 85TB. However, as a client PC may be inadvertently shut down, this makes the distributed storage system no more reliable than a conventional file server. We overcame this problem by using hierarchical RAID, RAID66, which is two layered RAID6 (Fig. 3.1).

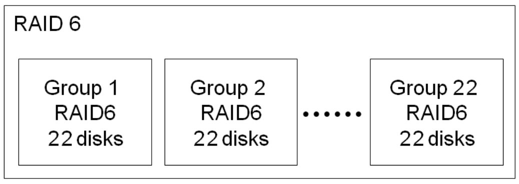


Fig. 3.1. *Composition of RAID66*

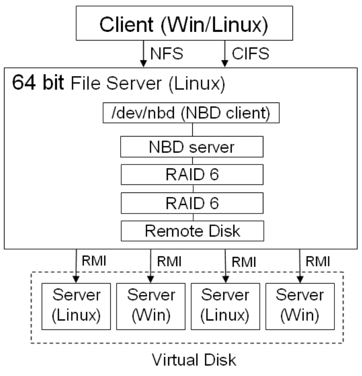


Fig. 3.2. *System Overview*

The trial system includes a 64-bit file server and the disk server as shown in Fig. 3.2. A virtual disk that supports OSs such as Linux and Windows for the disk servers provides read/write access to the disk via the Java RMI. The file server connects to the prepared disk and constructs RAID66. RAID66 is RAID6 on two levels. The NBD Server waits for access by an NBD Client. After the NBD Client is started, it is formatted with XFS. Windows clients access the file server through Samba, while NFS is used for Linux clients.

Since the NBD protocol lacks security, applying it on a network is dangerous. However, because NBD is used in our system for inter-process communication only, it can be applied without risk. Actual communication between client/server is realized by a protocol based on RMI with the relevant security considerations implemented.

Disk servers (of which there are 484) are started with the following command.

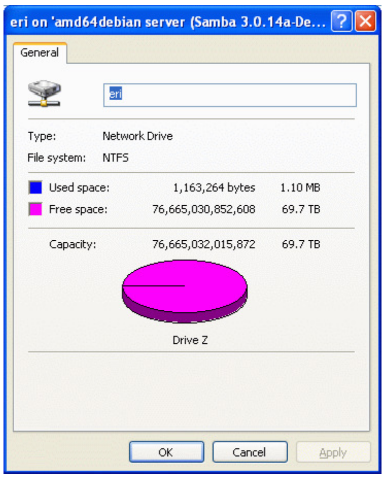
```
# java DiskServerImpl //localhost/pcn/DiskServerImpl imagen*
```

Here, imagen is a virtual disk of 180GB. It is executed on the file server side as follows.

```
# java vlsd.server.RAID66 9000
# nbd-client localhost 9000 /dev/nb0
# mkfs.xfs /dev/nbd0
# mount /dev/nbd0 /home/eri
# java vlsd.server.RAID66 9000
# nbd-client localhost 9000 /dev/nb0
```

Once execution has terminated successfully, storage can be used via a Windows mapped network drive. Figure 3.3 shows the disk drive properties of the shared disk and confirms the capacity of 70TB. Although the capacity of 484 x 180GB disks is 87TB because two per group parity is needed by RAID6, for a RAID6 array which consists of N nodes, only (N-2) / N disks can store data. Since RAID66 becomes two classes only 82% of the whole capacity can be used (=20/22*20/22). Therefore, the actual capacity is 70TB. In addition, a variable capacity virtual disk has been used in this trial. Data was not necessarily actually saved to all 70TB. Many client file systems converted only some of the available disk area during formatting.

3.3. Comparison of fixed and a variable length disks. As an experiment, a fixed-length disk and a variable-length disk were formatted by XFS, and the format times compared. The disks were virtual and saved as a file. Unlike in a fixed disk, only the part actually required in the variable-length disk was written to an image file.



eri

Type:	Network Drive	
File system:	NTFS	
🟦 Used space:	1,163,264 bytes	1.10 MB
🟪 Free space:	76,665,030,852,608	69.7 TB
Capacity:	76,665,032,015,872	69.7 TB

Drive Z

Fig. 3.3. *Disk drive properties of a RAID66 array*

Table 3.3
Format of virtual disk

Variable-Length Disk	
Capacity of Virtual Disk	Capacity nessary for format
1GB	10.45MB
10GB	10.66MB
100GB	50.78MB
1TB	129.41MB
10TB	129.39MB
100TB	132.07MB
1PB	165.45MB
10PB	498.58MB
100PB	3829.71MB

The time to format a variable-length disk is faster as depicted in Fig. 3.4. We postulate that the reason is because seek times are quicker with a variable-length disk. While formatting the variable-length disk using XFS, the capacity required for the format was recorded. As shown in Table 3.3 about 10MB is necessary when the disk is formatted to 1GB, and roughly a hundred MB is required for a 1TB disk. Actual physical disk space of 500MB is necessary for 10PB, and this increases to 3.8GB to format a 100PB disk.

3.4. Evaluation of RAID under Normal Operation. We compared the performance of our system with the theoretical performance presented in the paper by Chen et al. (1994). In this paper, typical RAID arrays are evaluated by their throughput per dollar relative to RAID0. Small read/write means read/write access to a block. The series RAIDNe gives the experimental values, while the values in the series RAIDNt are the theoretical values given in the paper by Chen et al. The throughput looks good, but this is simply because RAID0 is not so good and therefore makes comparisons seem highly effective. In our experiments, JBOD actually outperforms RAID0.

As shown in Fig. 3.5, the experimental values for RAID5 and RAID6 are almost equal to their respective theoretical values. This means that cost performance is also almost equal. Since for a Small read the data can be collected by access to a single disk in RAID5 and RAID6, performance is equal to RAID0.

In Fig. 3.6 the experimental values for RAID5 and RAID6 are about four times faster than the respective theoretical values. This is perhaps explained by the fact that RAID0 is slow. In RAID6, calculation of parity becomes complicated in comparison to RAID5 because there are two parity data records, and processing times become correspondingly larger. As a result, as shown in Fig. 3.6, RAID6 has lower throughput than RAID5. And unlike RAID0, in a Small Write RAID5 and RAID6 both require four accesses, namely read-out of data, read-out of parity, writing data, and writing parity.

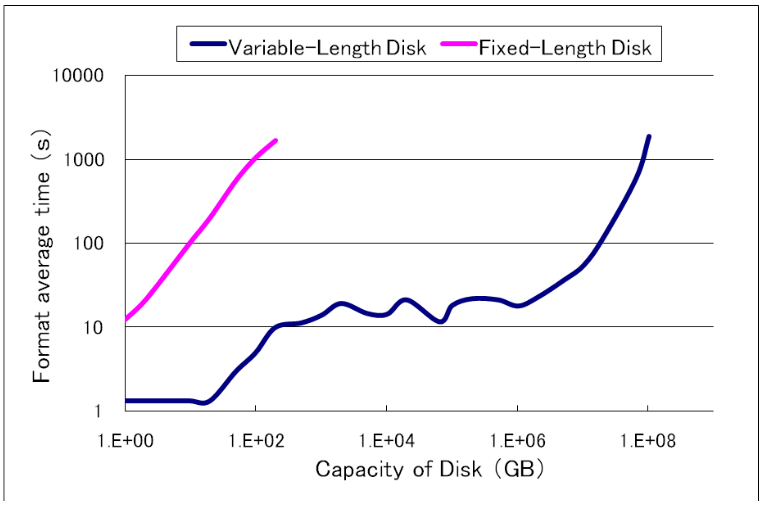


FIG. 3.4. *Comparison of a fixed and a wariable-length disk*

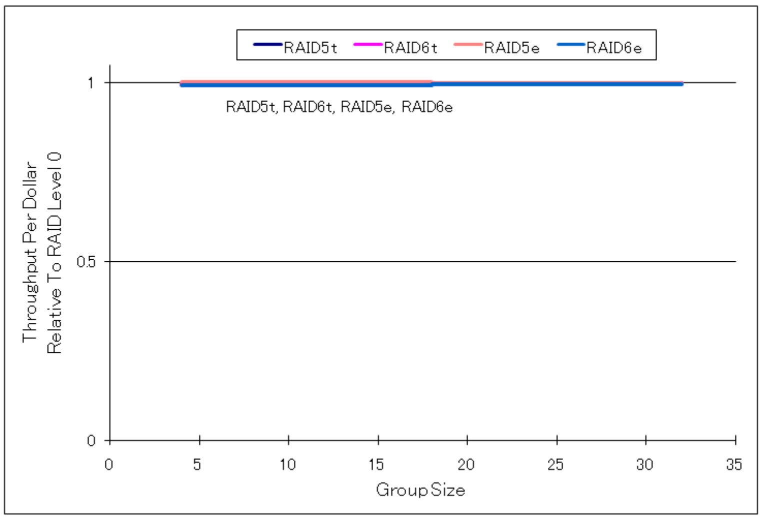


FIG. 3.5. *Small Read*

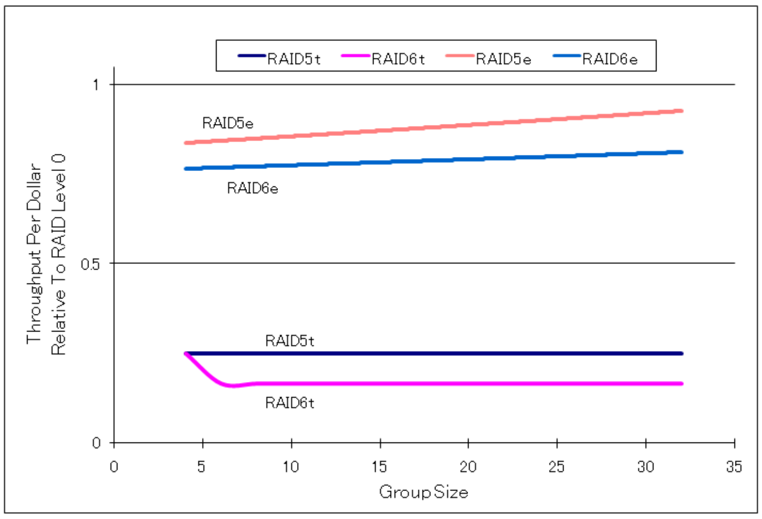


FIG. 3.6. *Small Write*

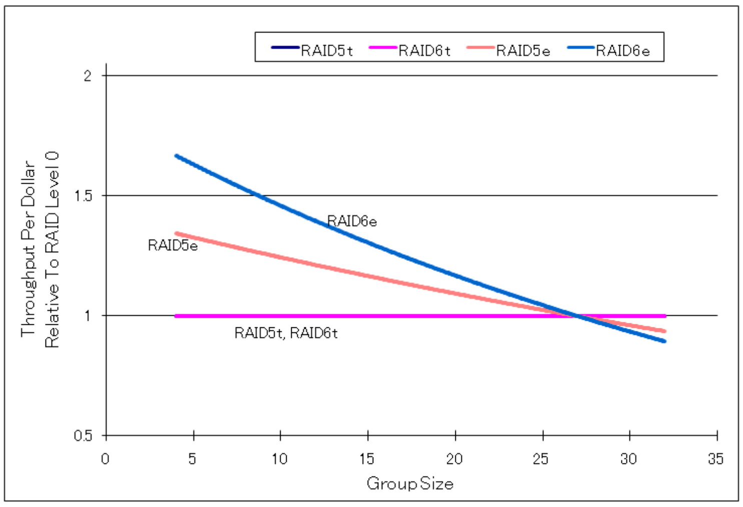


FIG. 3.7. *Large Read*

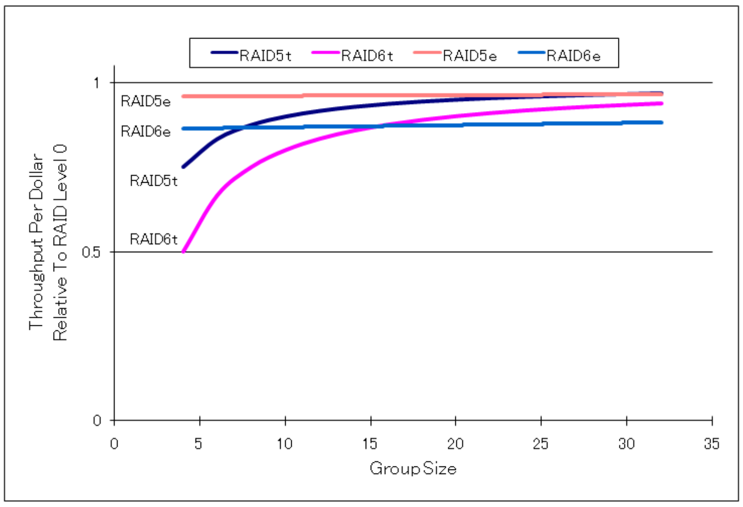


FIG. 3.8. *Large Write*

In the experimental values (Fig. 3.7) for Large Reads, although cost performance is very high for a small group size, it becomes lower for a large group. In a Large Read, because the reading domain of RAID6 is smaller than that of RAID5, cost performance is high. It is possible that the cost performance drops with a large group size because of an increase in some calculation processing.

The experimental values are almost constant in Fig. 3.8. For a large group size, the theoretical values exceed the experimental values for RAID6.

3.5. Evaluation of Small Read and Small Write of RAID6 under Drive Failure. Figures 3.9 to 3.12 depict graphs by throughput per dollar of typical RAID arrays using RAID6 in the case of a single disk failure (RAID6e1), and 2 disk failures (RAID6e2). The number of disk accesses for RAID6e1 and RAID6e2 is the same; the difference is that RAID6e2 has more parity processing operations.

In Small Read (Fig. 3.9), when one disk fails, cost performance falls to 25%, and when two disks fail, it falls to 10%. If the group becomes larger, the cost performance of RAID6e1 and RAID6e2 decreases. It is believed that in Small Read there is little disk access and almost all processing time is operation processing.

As shown in Fig. 3.10, in Small Write, the throughput of RAID6e1 is between 80% and 60%, while the throughput of RAID6e2 is about 70% to 40%. Because of the volume of disk access, the throughput of Small Write is higher than Small Read.

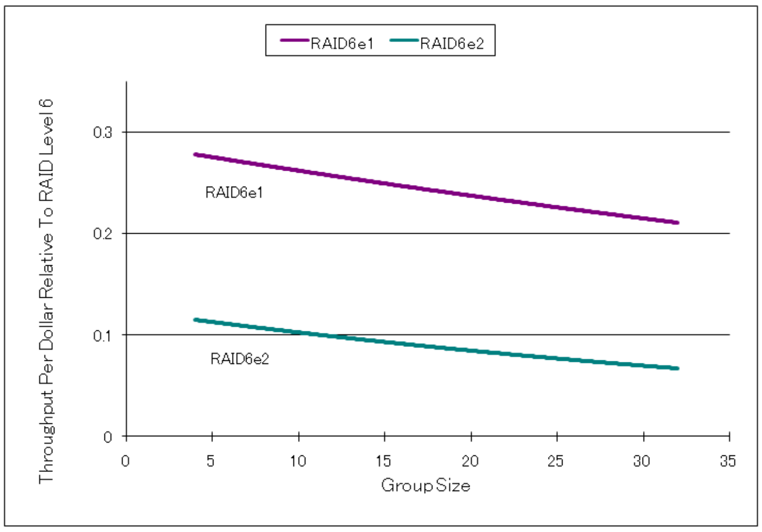


FIG. 3.9. *Small Read*

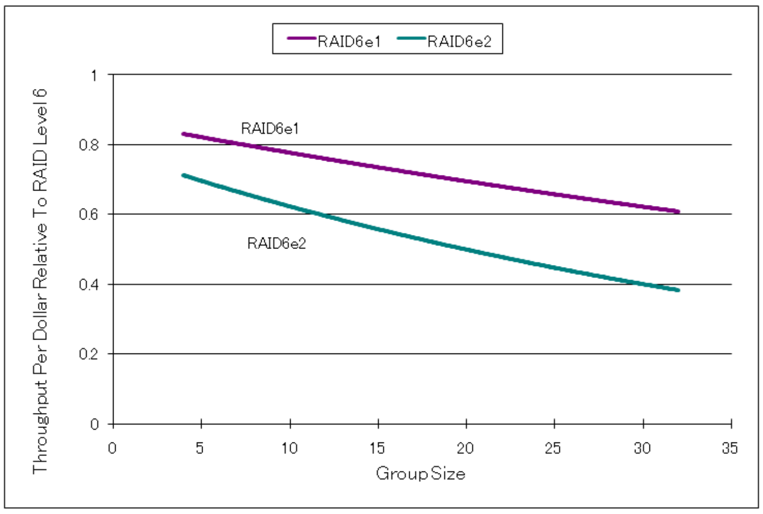


FIG. 3.10. *Large Write*

The difference between RAID6e1 and RAID6e2 is four fold in Large Read (Fig. 3.11). There is a throughput difference between RAID6e1 and RAID6e2 in both Large and Small Read. Meanwhile, because the disk writing time is far slower than the operation processing time in Small Write and Large Write (Fig. 3.12), there is no difference in throughput for Small Read and Large Read which perform disk reading only.

3.6. Reliability of hierarchy RAID on VLSD. The MTTF of the entire hierarchical RAID is calculated using the measured MTTF (mean time to failure) and MTTR (mean time to repair). Hierarchical RAID consists of a combination of RAID1, 5 and 6.

In Table 3.4, the total number of disks is 484. Layers 1 and 2 each contain 22 disks. RAID3 and 4 are omitted because they are the same as RAID5. Moreover, in RAID1 it was assumed that the contents were copied N times.

RAID1 should not be used, because the capacity efficiency decreases significantly. To achieve the largest capacity and highest MTTF, RAID66 is the best.

4. Building GCC. Many small files, such as text documents or source code of programs, are created in an educational environment. The size of most files is 1KB or less. Therefore, it is necessary to verify storage constructed with the VLSD in this environment.

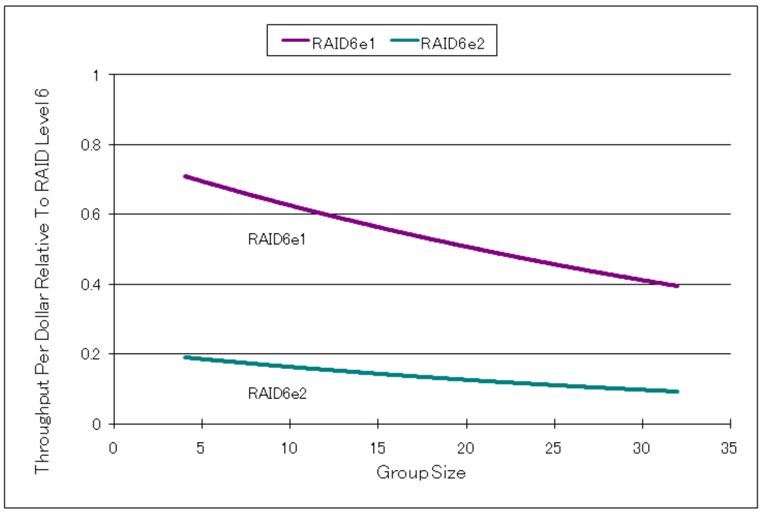


Fig. 3.11. *Large Read*

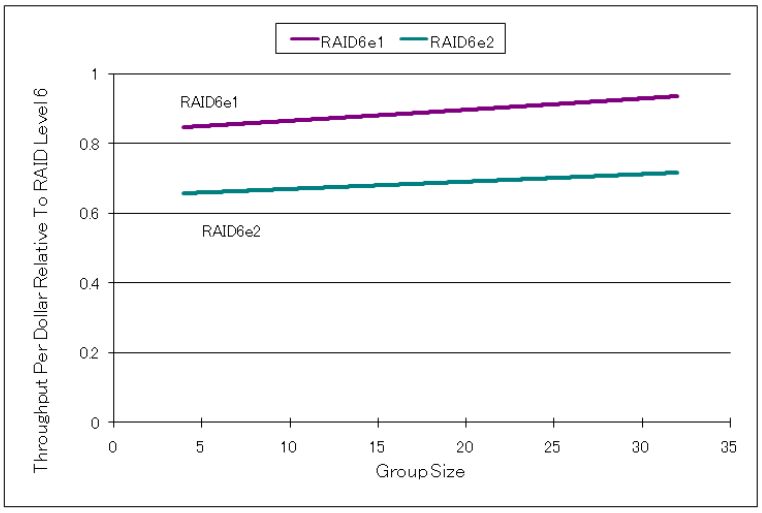


Fig. 3.12. *Large Write*

The performance of the storage built using VLSD was evaluated both during normal operation and in instances of failure. When a disk fails, performance drops for the Small Read as shown by the graph in Fig. 3.9.

A situation in which many small files are generated by the GCC build is actually created for this experiment, and is required to verify how the VLSD is affected. As shown in Fig. 4.1, before conducting the GCC build experiment, the experimental system must be constructed. Three Operating Systems, FreeBSD, Ubuntu Server, and Windows XP are installed on one machine for the experiment. For data communications, NFS is used between FreeBSD and Ubuntu, while NBD and VLSD are used between Ubuntu and Windows.

Below is the operating system specification.

- 1^{st} OS
  Windows XP Professional 64 bit
  CPU: AMD Athlone(tm) 64 X2 Dual Core 3800+
  RAM: 4GB
- 2^{nd} OS
  Ubuntu Server 7.10 64 bit using VMWare on 1st OS
  CPU: Same as 1st OS CPU
  RAM: 1GB

TABLE 3.4
MTTF and capacity of two level RAID

RAID level	MTTF[h]	Capacity Efficiency[%]
RAID11	4.25×10^{2463}	0.2
RAID15	1.7×10^{219}	4.3
RAID16	2.2×10^{328}	4.1
RAID51	4.19×10^{182}	4.3
RAID55	5×10^{15}	91.1
RAID56	1.1×10^{23}	86.8
RAID61	4.78×10^{272}	4.1
RAID65	7.8×10^{23}	86.8
RAID66	2.2×10^{35}	82.6

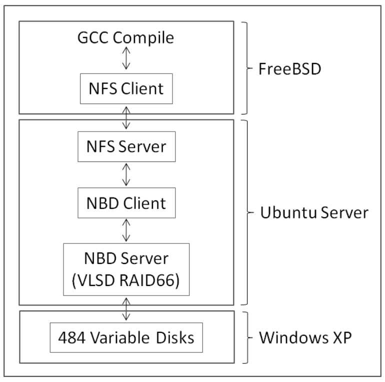


FIG. 4.1. *Experiment of GCC Build System*

- 3^{rd} OS
  FreeBSD 7.0 64 bit using VMWare on 1st OS
  CPU: Same as 1st OS CPU
  RAM: 1GB

To evaluate the performance of the VLSD, we setup three cases.

- In the first case (local disk), GCC is built on a local disk of FreeBSD.
- In the second case (NFS), GCC is built on a remote disk of the Ubuntu NFS server (local disk of Ubuntu).
- In the last case (VLSD), GCC is built on the remote disk of the Windows XP VLSD data grid (local disk of Windows).

As shown in Table 4.1, when performing the GCC build, the delay increases to 11% using NFS, while a 19% overhead was recorded when using VLSD. Consequently, the overhead for NFS is similar to the overhead for VLSD. It is thought that the network might be the cause of this delay.

When using RAID66, the storage system can still operate with up to nine failed disks depending on the fault position of the disk. This is the reason that up to eight disk faults were evaluated in this experiment. It is believed that the differences in the GCC build times with trouble-free operation, one disk failure, four disk failures and eight disk failures are not large because of the file caches in both the file system and operating system.

5. Conclusion.

VLSD is a toolkit used to build mass storage using available PCs. Using VLSD a large-scale virtual disk of 70TB was constructed with RAID66 and its performance was evaluated. Although the results differed from the theoretical values, this variation can be considered to be no more than standard errors introduced by the empirical implementation. We also evaluated each disk class, namely FileDisk, VariableDisk, JBOD, PagedDisk, SingleDisk, RAID0, RAID1, RAID5 and RAID6.

To simulate a typical storage system in an educational environment that may contain many small files, we built GCC on various storage systems and evaluated the performance of each. The results show that the

Table 4.1
Required Time for GCC Build

Path	Compiled Time	Overhead
Local Disk	2:05:17	0%
NFS	2:18:43	11%
VLSD vith no disk fault	2:28:32	19%
VLSD vith 1 disk fault	2:29:02	19%
VLSD vith 4 disk fault	2:31:16	21%
VLSD vith 8 disk fault	2:32:16	22%

overhead for VLSD is the same as the overhead for NFS. This means that the storage system is usable even with many small files.

In this study, the estimation used only a single machine. However, the real system consists of around 500 machines. Performance still needs to be investigated when a network of two or more machines is used.

REFERENCES

[1] D. A. Patterson and R. L. Katz, *A Case for Redundant Arrays of Inexpensive Disks (RAID)*, ACM SIGMOD, 1988, pp. 109–116

[2] P. M. Chen, E. K. Lee, G. A. Gibson, H. Katz and D. A. Patterson, *RAID: High-Performance, Reliable Secondary Strage*, ACM Computing Surveys, Vol. 26, No.2, Jun. 1994, pp. 145–185

[3] E. Chai, M. Uehara, H. Mori and N. Sato, *Virtual Large-Scale Disk System for PC-Room*, LNCS 4658, Network-Based Information Systems, Sept. 2007, pp. 476–485

[4] E. Chai, M. Uehara and H. Mori, *Evaluating Performance and Fault Tolerance in a Virtual Large-Scale Disk*, In Proceedings of 22nd International Conference on Advaced Information Networking and Applications, Mar. 2008, pp. 926–933

[5] *Intelligent RAID6 Theory Overview and Information*, `http://download.intel.com/design/storage/papers/30812202.pdf`

[6] W. Verhelst,*Network Block Device*, `http://nbd.sourceforge.net`

[7] *Samba*, `http://www.samba.org`

[8] S. Shepler, et. al., *NFS version 4 Protocol,* `http://www.ietf.org/rfc/rfc3010.txt`

[9] *XFS*, `http://oss.sgi.com/projects/xfs/`

Edited by: Fatos Xhafa, Leonard Barolli
Received: September 30, 2008
Accepted: December 15, 2008