Enhancing Write I/O Performance of Disk Array RM2 Tolerating Double Disk Failures

Young Jin Nam, Dae-Woong Kim, Tae-Young Choe, Chanik Park
Department of Computer Science and Engineering/PIRL
Pohang University of Science and Technology
Kyungbuk, Republic of Korea
{yjnam,woong,choety,cipark}@postech.ac.kr

Abstract

With a large number of internal disks and the rapid growth of disk capacity, storage systems become more susceptible to double disk failures. Thus, the need for such reliable storage systems as RAID6 is expected to gain in importance. However, RAID6 architectures such as RM2, P+Q, EVEN-ODD, and DATUM traditionally suffer from a low write I/O performance caused by updating two distinctive parity data associated with user data. To overcome such a low write I/O performance, we propose an enhanced RM2 architecture which combines RM2, one of the well-known RAID6 architectures, with a Lazy Parity Update (LPU) technique. Extensive performance evaluations reveal that the write I/O performance of the proposed architecture is about two times higher than that of RM2 under various I/O workloads with little degradation in reliability.

1. Introduction

With the advent of Storage Area Networks, such as Fiber Channel and Gigabit Ethernet, large-scale storage systems which encompass a large number of disks are commonplace. Such systems are advantageous in terms of scalability and configurability, but they are more susceptible to double disk failures than small-scale systems [6]. Moreover, the rapid growth of disk capacity prolongs the disk recovery time in the event of disk failure. Eventually, this prolonged recovery time will raise the chances of subsequent disk failure during the reconstruction of user data and parity information stored in a faulty disk. In addition, the probability increases that, while reading data which was left unread for a long time, latent sector failures will occur [12]. To recognize these reliability issues associated with recent technology trends, we expect that the need for such reliable storage systems as RAID6 will gain in importance.

A few highly reliable RAID architectures, such as P+Q [6], EVEN-ODD [4], RM2 [8], and DATUM [2], can tolerate double disk failures by elaborately maintaining two distinctive parity data associated with user data. This breed of architectures is formally classified as RAID6 [6]. However, RAID6 suffers from a relatively low write I/O performance compared with other architectures, such as RAID1, 4, 5 [9] due to the maintenance of additional parity information. While RAID5 requires four disk accesses to process a write I/O request, RAID6 demands six disk accesses.

While a myriad of techniques to improve the write I/O performance of RAID5 can enhance that of RAID6 to some extent, none of them can serve as an ultimate solution to resolve the low write I/O performance of RAID6 because they still update two parity information per write I/O request. In [11], Savage and Wilkes proposed *A Frequently Redundant Array of Independent Disks* (AFRAID) which improves the write I/O performance of RAID5 by deferring the update of parity information until the storage system becomes idle. However, the problem of selecting parity groups for the delayed update in RAID5 is straightforward, but not trivial in RAID6. This paper proposes an enhanced RM2 architecture which improves the write I/O performance of RM2, a well-known RAID6 architecture, by employing the *Lazy Parity Update* (LPU). We provide a systematic scheme to determine the parity groups for the delayed update. It is shown that the proposed architecture can double the write I/O performance of RM2 while still providing high reliability under various I/O workloads.

2. The Proposed Architecture

We begin by explaining the nomenclature to be used for the description of the proposed architecture. A stripe unit refers to a group of consecutive disk blocks in a disk. Stripe units which contain user data and parity information are called a data stripe unit and a parity stripe unit, respec-

Proceedings of the International Conference on Parallel Processing (ICPP'02)
0-7695-1677-7/02 $17.00 © 2002 IEEE

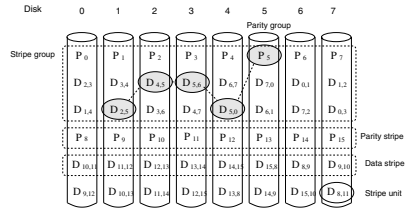
tively. A parity group is referred to as a set of data stripe units and an associated parity stripe unit where the parity stripe unit is calculated from the set of data stripe units. Data and parity stripe units are stored in different disks, so that faulty stripe units can be reconstructed from other non-faulty stripe units. Let us define $\mathcal{PG}$ as all parity groups within a RAID6 architecture. Next, the $\mathcal{PG}$ is divided into two disjoint parity groups defined as follows.

Definition 1 *Foreground Parity Group* (FPG) refers to the minimum set of parity groups in $\mathcal{PG}$ which can tolerate a single disk failure. *Background Parity Group* (BPG) is defined as $\mathcal{PG} - FPG$.

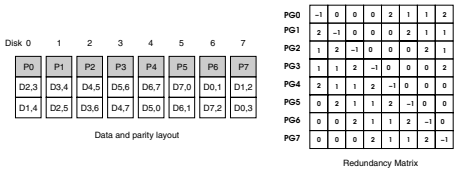
In addition, the *Background Parity Group List* ($BPGL$) refers to a list of delayed parity groups. $|BPGL|$ is the number of parity groups in $BPGL$. *Background Parity Group Task* ($BPGT$) is a background task which processes delayed parity groups in $BPGL$.

2.1. Overview of the RM2 Architecture

Figure 1(a) shows data and parity placements in RM2 [8] which is the base RAID6 architecture of our current work. A data stripe represents a stripe which contains only data stripe units, while a parity stripe contains only parity stripe units. A stripe group is defined as a set of data stripes and a parity stripe which covers all stripe units in a given parity group. Data and parity placements of RM2 are mainly determined based on a Redundancy Matrix, which maps each stripe unit to its corresponding two parity stripe units within a stripe group for the given N disks, as shown in Figure 1(b). A column and a row in the Redundancy Matrix correspond to a disk and a parity group, respectively. Entries in a column have a -1 and a pair of k's where $1 \leq k \leq M-1$ and M is the stripe group size. An efficient algorithm to determine the maximum M for the given N disks is given in [7]. Note that $M \geq 2$ because a single parity stripe always contains at least a single data stripe. Let us denote $RM_{i,j}$ as the i-th row and j-th column entry in the Redundancy Matrix. If $RM_{i,j} = -1$, then a parity stripe unit of disk j belongs to parity group i, if $RM_{i,j} = 0$, then it has no information, and if $RM_{i,j} = k$ for $1 \leq k \leq M-1$, then the k-th data stripe unit of disk j belongs to parity group i. PG_i represents the i-th parity group and P_i means the parity stripe unit in the i-th parity group. $D_{i,j}$ indicates that the data stripe unit is involved in computing P_i and P_j, implying the i-th parity group and the j-th parity group. As depicted in Figure 1(a), the parity unit P_5 is related to four data units which encompass $D_{2,5}$, $D_{4,5}$, $D_{5,6}$, and $D_{5,0}$. Conversely, the data unit $D_{2,5}$ is related to P_2 and P_5. More detailed information on RM2 architecture can be found in [8].



(a) Data/parity placement in a disk



(b) Data/parity layout and Redundancy Matrix

Figure 1. RM2 Architecture: (a) data and parity placements and (b) Redundancy Matrix

2.2. Enhancing Write I/O Performance of RM2

Lazy Parity Update (LPU) Technique: The LPU technique divides all parity groups of $\mathcal{PG}$ into FPG and BPG for the given N disks and serves a write I/O request according to the information from FPG and BPG. Let us

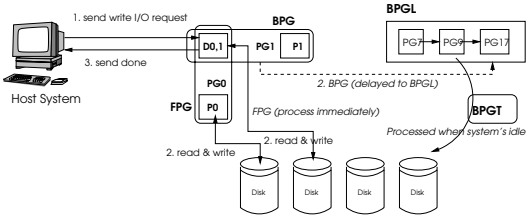


Figure 2. Operation of the Lazy Parity Update technique

denote two parity groups associated with a data block k as pg_k^i, where $i = 0, 1$. Under a normal condition with no disk failures, if pg_k^i belongs to FPG, it will be served before the notification of its completion is delivered to a host. If pg_k^i is included in BPG, its processing will be postponed by registering it into $BPGL$. The request will be processed when the storage system becomes idle. In case that a write-back buffer cache is employed, this behavior occurs when destaging begins. As a result, it can improve the through-

put in order to destage the delayed write I/O requests. In the presence of at least one faulty disk, however, two associated parity groups are regarded as being included in FPG. As a result, the processing of parity groups can be delayed no longer. Instead, the storage system is ready to tolerate an additional disk failure by processing all the delayed parity groups in $BPGL$ without a further increase in the $BPGL$ list. Algorithm 1 presents how to manipulate two parity information associated with a write I/O request by using the LPU technique.

Algorithm 1: Manipulating parity information with the LPU technique

```
input  : a write I/O request, r_k
begin
  for each PG_k^i of two associate parity groups do
    if # of faulty disks > 0 or PG_k^i ∈ FPG then
      update PG_k^i;
    else
      delay the updating of PG_k^i by putting it into BPGL;
    end
  end
end
```

Figure 3 depicts the state diagram of the proposed architecture which combines the LPU technique with RM2. Basically, six states exist, which are `NORM_safe`, `NORM_unsafe`, `DEG1_safe`, `DEG1_unsafe`, `DEG2`, and `FAIL`. Each state is determined based on the number of faulty disks and the status of $BPGL$. By delaying the

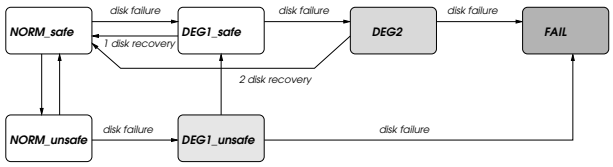
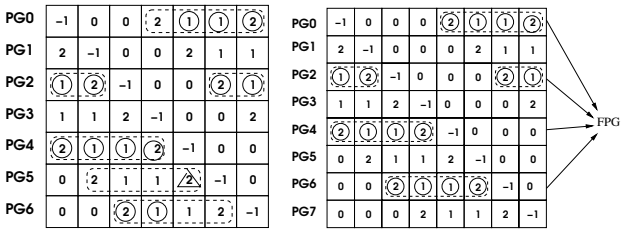


Figure 3. State transition diagram of the proposed architecture

updating of a parity group, `NORM_safe` is changed into `NORM_unsafe` if $BPGL = \emptyset$. When all delayed parity groups in $BPGL$ are processed, `NORM_unsafe` is changed to `NORM_safe`. If a disk failure occurs in `NORM_safe` and `NORM_unsafe`, then the current state is replaced with `DEG1_safe` and `DEG1_unsafe`, respectively. With an additional disk failure, `DEG1_safe` is moved to `DEG2`. A subsequent disk failure in either `DEG1_unsafe` or `DEG2` results in `FAIL`. When all faulty parity and data stripe units are recovered at `DEG1_safe` and `DEG2`, the current state is moved to `NORM_safe`.

Initialization of FPG with RM2: In order to make the LPU technique work with RM2, we need to appropriately initialize the two disjoint sets of parity groups – FPG and BPG for the given N disks. Figure 4 provides two examples of initializing parity groups into FPG and BPG with a Redundancy Matrix of a different size. Recall that a Redundancy Matrix plays a key role in manipulating the RM2 architecture. It maintains information on how to place data and parity stripe units within a stripe group. Since a data and/or parity layout within a stripe group is repeated over a storage system, as shown in Figure 1(a) and (b), we focus on a single stripe group rather than consider all parity groups in the storage system. When $N = 7$ and $M = 3$, as shown in Figure 4(a), PG_0, PG_2, PG_4, PG_5, and PG_6 belong to FPG. In case of $N = 8$ and $M = 3$, PG_0, PG_2, PG_4, and PG_6 fall into FPG. Two examples in Figure 4 reveal



(a) $N = 7, M = 3$ (b) $N = 8, M = 3$

Figure 4. Illustrative examples of configuring FPG and BPG with $N = 7, M = 3$ and $N = 8, M = 3$ within a single stripe group

an interesting tendency, where an $|FPG|$ of $N = 7$ is not smaller than $N = 8$, *i.e.*, the size of FPG is not directly proportional to the number of disks, N. Next, we will solve the problem of how to compute an $|FPG|$ and how to select the set of minimum parity groups of FPG. The following lemma and theorem compute the size of FPG for the given N disks and stripe group size of M.

Lemma 1 *Given RM2 with N disks and a stripe group size of M, $|FPG| \geq \lceil \frac{N}{2} \rceil S$, where S is the total number of stripe groups.*

Theorem 1 *Given RM2 with N disks and a stripe group size of M,*

$$|FPG| = \begin{cases} \frac{N}{2}S & \text{if } N = \text{even} \\ (\lceil \frac{N}{2} \rceil + M - 2)S & \text{if } N = \text{odd} \end{cases},$$

where S is the total number of stripe groups.

Algorithm 2 selects FPG for a single stripe group with the given N disks. Note that a data/parity layout within a stripe group is repeated in the other stripe groups. Thus, a parity group index can be regarded as a relative parity group index at each stripe group, not as an absolute parity group index.

For a stripe group, according to Algorithm 2, the number of FPG is $\frac{N}{2}$ with an even N and the number of FPG is $\lceil \frac{N}{2} \rceil + M - 2 = \lceil \frac{N}{2} \rceil + \lfloor \frac{M-2}{2} \rfloor + \lceil \frac{M-2}{2} \rceil$ with an odd N.

Algorithm 2: Selecting the minimum FPG within a stripe group

input : N disks, M stripe group size
output : FPG
begin
  /* each stripe group k has a set of parity strip units */
  /* a stripe group denoted by $\{PG_i | 0 \leq i \leq N-1\}$ */
  if N *is even* **then**
    $FPG = \{PG_{2i} | 0 \leq i < \frac{N}{2}\}$
  else
    $FPG = \{PG_{2i} | 0 \leq i < \lceil \frac{N}{2} \rceil\} \cup$
    $\{PG_{2j+1} | 0 \leq j < \lfloor \frac{M-2}{2} \rfloor\} \cup$
    $\{PG_{N-2-2k} | 0 \leq k < \lceil \frac{M-2}{2} \rceil\}$
  end
end

Theorem 2 *For the given N disks and stripe group size of M, Algorithm 2 provides a valid FPG.*

The proofs of the lemma and the theorems can be found in [7].

2.3. Other Issues

When implementing the proposed architecture on an actual RAID system, some issues arise. First, we need to detect the system idle period in order to process the delayed parity groups in $BPGL$. In our current design, the proposed architecture begins to serve the delayed parity groups immediately when no outstanding I/O requests are incoming from hosts. Second, we must determine how quickly the delayed parity groups in $BPGL$ are to be processed in the event of a single disk failure. As a result, the system moves into a safe state where an additional disk failure can be tolerated. Our current design configures the priority of $BPGT$ as equal to a background task which reconstructs lost data from a faulty disk. Setting the priority of $BPGT$ as that of the normal I/O process will increase the service time of each I/O request until the processing of all delayed parity groups is completed.

2.4. Features of the Proposed Architecture

First, the proposed architecture can considerably improve the write I/O performance of RM2 because it updates only the parity groups to prevent data loss from a single disk failure under no disk failure. However, the write I/O performance of the proposed architecture is identical to that of RM2 in the presence of disk failure.

Second, even if the proposed architecture cannot guarantee the strict reliability requirement of tolerating double disk failure at any given time as RM2, it is expected to still provide extremely high reliability compared with RAID5. Let us consider the event of a disk failure when a storage system is in either `DEG1_safe` or `DEG1_unsafe` depending on the $BPGL$ status. We will focus merely on the `DEG1_unsafe` state as `DEG1_safe` can tolerate an additional disk failure. At `DEG1_unsafe`, a recovery path exists from `DEG1_unsafe` to `DEG1_safe` in order to reach a state where an additional disk failure can be tolerated in the face of a single disk failure. This path needs to process only the delayed parity groups in the $BPGL$, rather than recover all the parity groups in the storage system, as with RAID5. Consequently, it can be expected that the reliability of the proposed architecture is much higher than that of RAID5, considering the number of parity groups to be processed.

3. Performance Evaluations

This section evaluates the performance of the proposed architecture, which combines the LPU technique with the RM2 architecture.

3.1. Experimental Environment

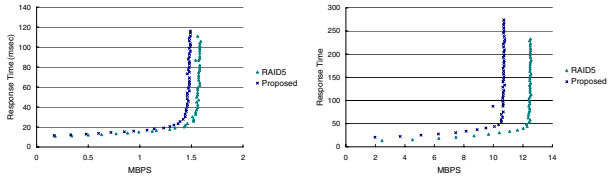
The proposed architecture is implemented on a real RAID system called PosRAID along with RAID5 and RM2. Hardware components of the PosRAID encompass Pentium IV 1.3GHz, 256MB memory, two QLogic's QLA2200 Fibre Channel HBAs, 32bit/33MHz PCI bus, and six 7200rpm 9GB Seagate Barracuda ST39175FC disks. Note that PosRAID does not support a hardware XOR module. Software components of PosRAID include VxWorks version 5.4 RTOS, a communication module which communicates with hosts and internal disks via Fibre Channel host bus adapters, a RAID engine module which maps a logical block address to a physical block address based on a given RAID architecture and manages a buffer cache, and a resource and/or configuration management module which allocates, deallocates, and configures all hardware and software resources within the storage system. Table 1 shows the configurations of three different architectures under performance evaluations. Note that the number of disks used in both architectures are equal. As a result, the size of RAID5 is larger than RM2 and the proposed architecture which is based on the RM2 architecture. Also, the redundancy rate of RM2 and the proposed architecture is 100 percent because M is set to 2 with $N = 6$.

Table 1. Configurations of three different architectures

Architecture	RAID5	RM2	Proposed
# of disks (N)	6	6	6
Stripe unit size	32KB	32KB	32KB
Buffer cache size	32MB	32MB	32MB
Write cache policy	write back	write back	write back
Parity group size	6	3	3
Stripe group size (M)	–	2	2
Total size	45GB	27GB	27GB
Redundancy ratio	20%	100%	100%

3.2. I/O Performance Measurement

This section measures the I/O performance of the proposed architecture, RAID5, and RM2. Figure 5(a)–(b) show the read I/O performance of RAID5 and the proposed architecture for 4KB and 64KB read I/O workloads with an increase in I/O processes. The read I/O performance of RM2 is the same as that of our proposed architecture. Each I/O has a thinking time which is exponentially distributed with a 10-msec mean which then generates I/O requests and waits until the issued I/O request completes. It then repeats this process of I/O generation. The start block addresses of I/O requests are uniformly distributed in the first 10GB disk space, ranging from black 0 to block 20,971,520. In the

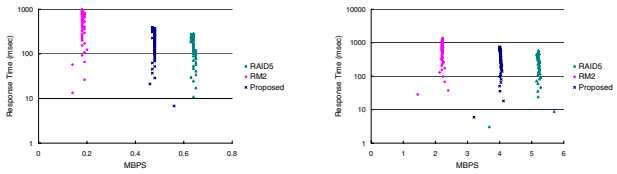


(a) 4KB random read I/O (b) 64KB random read I/O

Figure 5. Read I/O performance of RAID5 and the proposed architecture with an I/O workload where $N = 6$ and no disk failure exists: (a) 4KB random reads and (b) 64KB random reads

results, the read I/O performance of the proposed architecture is about 93.3 percent and 85.5 percent of the read I/O performance of RAID5 under a 4KB random read I/O workload and a 64KB random read I/O workload, respectively. This performance gap occurs due to the different redundancy rate of each architecture. Table 1 shows that RAID5 requires 120% of $\frac{10GB}{6}$ (2GB) space from each disk in order to provide the 10GB disk space, whereas the proposed architecture needs 200% of $\frac{10GB}{6}$ (3.3GB) space from each disk. This implies that the maximum distance of a disk head movement by the proposed architecture is longer than that of RAID5.

Figure 6(a)–(b) gives the write I/O performance of RAID5, RM2, and the proposed architecture for 4KB and 64KB I/O workloads with an increase in I/O processes. First, we can see that the write I/O performance of the proposed architecture is two times higher than RM2, as expected. However, with the increase of an I/O request size from 4KB to 64KB, the performance gap between the two architectures is slightly reduced because the relative ratio of required disk accesses between the two architectures decreases with the larger I/O request size. However, the write I/O performance of the proposed architecture is still lower than that of RAID5 for the same reason noted above.



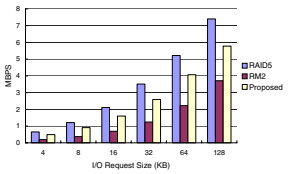
(a) 4KB random write I/O (b) 64KB random write I/O

Figure 6. The write I/O performance of RAID5, RM2, and the proposed architecture with an I/O workload where $N = 6$ and no disk failure exists: (a) 4KB random writes and (b) 64KB random writes

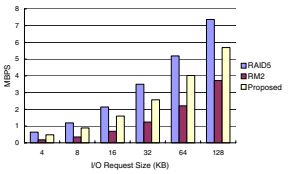
Figure 7(a)–(d) reveals the write I/O performance and corresponding response times of RAID5, RM2, and the proposed architecture as a write I/O request size varies. As previously shown, the proposed architecture outperforms RM2 by 100% with different write I/O request size under medium and heavy I/O workloads. As mentioned, differences of write I/O performances between the proposed architecture and RM2 are reduced as the I/O request sizes increase.

3.3. Reliability Measurement

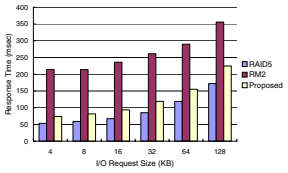
The reliabilities of RAID5, RM2, and the proposed architecture are analyzed in terms of *mean time to data loss* (MTTDL). Next, a realistic MTTDL value of the proposed architecture is computed by obtaining realistic values that correspond to theoretical parameters used in the mathematical analysis.



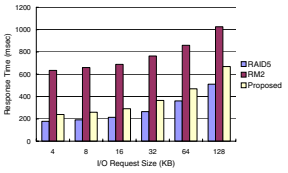
(a) Write I/O performance under medium I/O workload



(b) Write I/O performance under heavy I/O workload



(c) Response time under medium I/O workload



(d) Response time under heavy I/O workload

Figure 7. The write I/O performance and corresponding response times of RAID5, RM2, and the proposed architecture as a function of an I/O request size, where $N = 6$ with no disk failure: the medium I/O workload has 11 I/O processes and the heavy I/O workload has 31 I/O processes

Theoretical Analysis: The analysis assumes that both the failure rate and the repair rate of each disk follow an exponential distribution. The MTTDL of each architecture is computed by using the fundamental matrix M defined as $M = [I - Q]^{-1}$, where Q is a truncated stochastic transitional probability matrix [3]. Then, we can obtain the MTTDL of RAID5 and RM2 as follows. Note that Equation (1) is the same as the reliability equation of RAID5 given in [9].

$$MTTDL_{RAID5} = \frac{(2N-1)\lambda_1 + \mu_1}{N(N-1)\lambda_1^2} \quad (1)$$

$$MTTDL_{RM2} = \frac{\lambda_1^2 - 2\lambda_1\mu_1 + \mu_1^2}{N\lambda_1^3} + \frac{\lambda_1^2 - \lambda_1\mu_1 - 2\mu_1^2}{(N-1)\lambda_1^3} + \frac{\lambda_1^2 + 3\lambda_1\mu_1 + \mu_1^2}{(N-2)\lambda_1^3} \quad (2)$$

Figure 8 depicts a Markov diagram of the proposed architecture based on the state transition diagram in Figure 3. Finally, the $MTTDL_{proposed}$ of the proposed architecture is obtained as follows:

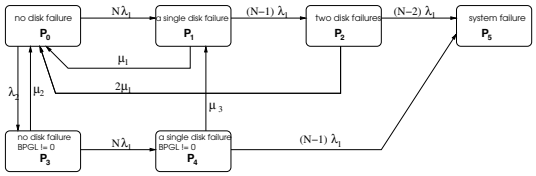
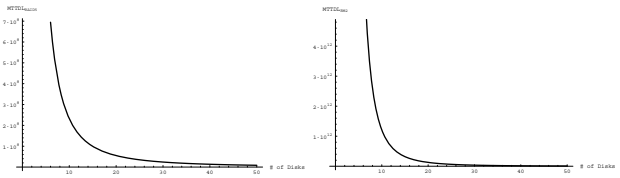


Figure 8. Markov diagram of the proposed architecture

$$MTTDL_{proposed} = \frac{\alpha}{N(N-1)(N-2)\lambda_1^3} + \frac{\beta}{\gamma}\frac{1}{N(N-1)(N-2)\lambda_1^3}, \quad (3)$$

where $\alpha = (2\lambda_1^2 - 6N\lambda_1^2 + 3N^2\lambda_1^2 - 4\lambda_1\mu_1 + 5N\lambda_1\mu_1 + 2\mu_1^2)$, $\beta = (2N\lambda_1^4\lambda_2 - 5N^2\lambda_1^4\lambda_2 - N^4\lambda_1^4\lambda_2 + 8\lambda_1^3\lambda_2\mu_1 - 26N\lambda_1^3\lambda_2\mu_1 + 26N^2\lambda_1^3\lambda_2\mu_1 - 8N^3\lambda_1^3\lambda_2\mu_1 - 20\lambda_1^2\lambda_2\mu_1^2 + 40N\lambda_1^2\lambda_2\mu_1^2 - 19N^2\lambda_1^2\lambda_2\mu_1^2 + 16\lambda_1\lambda_2\mu_1^3 - 16N\lambda_1\lambda_2\mu_1^3 - 4\lambda_2\mu_1^4)$, and $\gamma = (2N\lambda_1^3 - 3N^2\lambda_1^3 + N^3\lambda_1^3 + 2\lambda_1^2\lambda_2 - 3N\lambda_1^2\lambda_2 + N^2\lambda_1^2\lambda_2 - 4\lambda_1\lambda_2\mu_1 + 3N\lambda_1\lambda_2\mu_1 + 2\lambda_2\mu_1^2 + 2\lambda_1^2\mu_2 - 3N\lambda_1^2\mu_2 + N^2\lambda_1^2\mu_2 - 2N\lambda_1^2\mu_3 + N^2\lambda_1^2\mu_3 - 2\lambda_1\lambda_2\mu_3 + N\lambda_1\lambda_2\mu_3 - 2\lambda_1\mu_2\mu_3 + N\lambda_1\mu_2\mu_3)$.

Comparisons of Reliabilities: Figure 9 shows the reliability of RAID5 and RM2 architectures, where a MTTF of a single disk ($MTTF_{disk}$) is set to 1,000,000 hours [1] and a MTTR of a single disk ($MTTR_{disk}$) to reconstruct all faulty parity and data blocks is set to 48 hours [11]. That is, $\lambda_1 = 1/1,000,000$ and $\mu_1 = 1/48$. We can see that the reliability of RM2 is higher than that of RAID5 by 10,417 times with 6 disks and by 869 times with 50 disks.

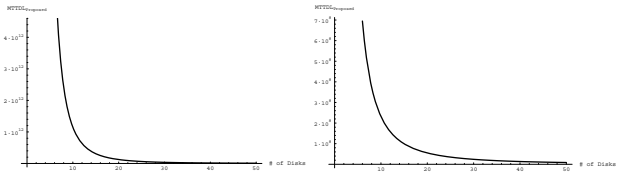


(a) RAID5 (b) RM2

Figure 9. Reliability comparison between RAID5 and RM2

Depending on the increasing and decreasing rates of the delayed parity groups in $BPGL$ (λ_2, μ_2) and the processing rates of delayed parity groups in $BPGL$ in the presence of disk failure (μ_3), the reliability of the proposed architecture

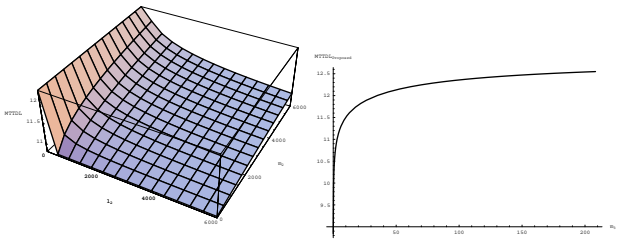
ranges from that of RAID5 as the worst case to that of RM2 as the best case. Figure 10 shows two extreme cases. While the first case of $\lambda_2 = 10^4\mu_2$ and $\mu_3 \simeq \mu_1$ has the same reliability as RAID5, the second case of $\lambda_2 = 10^{-4}\mu_2$ has the same reliability as RM2.



(a) $\lambda_2 \gg \mu_2, \mu_3 \simeq \mu_1$ (b) $\lambda_2 \ll \mu_2$

Figure 10. Two extreme cases of reliability of the proposed architecture

In what follows, we will investigate the effects of the reliability of the proposed architecture as a function of λ_2, μ_2, and μ_3. First, Figure 11(a) presents the effects of λ_2 and μ_2 with a fixed μ_3 on the reliability of the proposed architecture with $N = 6$, where μ_3 is set to $100\mu_1$. If λ_2 is higher than μ_2, then the reliability decreases because the probability of staying at $NORM_{unsafe}$ correspondingly increases. Given a fixed λ_2, the reliability is enhanced with an increase of μ_2. If $\lambda_2 \simeq 0$, then μ_3 does not affect the



(a) Effects of λ_2 and μ_2 (b) Effects of μ_3

Figure 11. Reliability of the proposed architecture with $\lambda_1 = \frac{1}{1{,}000{,}000}$, $\mu_1 = \frac{1}{48}$

reliability of the proposed architecture. In Figure 11(b), λ_2 and μ_2 are set to $\frac{1}{0.01}$ and 100, respectively, *i.e.*, μ_2 is set to much lower than λ_2. As a result, the system is expected to remain at $NORM_{unsafe}$ for most of the time with no disk failure. Next, the value of μ_3 varies from $\frac{1}{48}$ to $\frac{1}{0.0048}$. Figure 11(b) shows that the reliability of the proposed architecture is equal to that of RAID5 when μ_3 is small. Conversely, if μ_3 becomes larger, *i.e.,* ≥ 100, then its reliability becomes equal to that of RM2. Note that $\mu_3 = \frac{1}{0.001}$ (hour) means that the expected time to process all delayed parity groups in $BPGL$ will be approximately 4 seconds.

Measurement of a Realistic $MTTDL_{proposed}$ with a Traced I/O Workload: To empirically obtain realistic values for λ_2, μ_2, and μ_3 which were used in the previous analysis, we employ a traced I/O workload of the `cello` system [10]. A Linux system regenerates all I/O requests which were issued to eight disks within the `cello` system on 05/04/92. Note that a different disk in the `cello` system is mapped to a different region of the proposed architecture. When measured, the system remained at $NORM_{safe}$ for 83,176 seconds and $NORM_{unsafe}$ for 3,224 seconds during the 84,400-second observation period. Recall that the RAID system being tested exploits the delayed write scheme with a 64MB write back cache. Thus, the event of inserting the delayed parity groups into $BPGL$ occurs in a bursty manner when dirty data in the write back cache are flushed into physical disks. First, by computing all inter-arrival times between two subsequent times when a parity group is delayed into an empty $BPGL$, we can obtain a realistic value of λ_2, *i.e.*, $\frac{1}{\lambda_2} = 268.01$ seconds, where the maximum inter-arrival time is 3560.78 seconds and the minimum inter-arrival time is 1.11 seconds. Second, by averaging out all elapsed times to process the delayed parity groups in $BPGL$ when $BPGL \neq \emptyset$, it can be calculated that $\frac{1}{\mu_2} = 14.45$ seconds. Third, the maximum value of $|BPGL|$ at $NORM_{unsafe}$ is observed as 934. Recall that the priority of the $BPGT$ is the same as that of the background process which rebuilds faulty data and parity blocks. Then, by deriving a relative relationship between μ_1 and μ_3, we can compute the worst case time to process all delayed parity groups in $BPGL$ in the presence of a disk failure. Since the size of a single disk is 9GB and $M = 2$, reconstructing an entire faulty disk requires rebuilding $147,456$ $(= \frac{9GB}{32KB \cdot 2})$ parity groups, which takes $\frac{1}{\mu_1} = 48$ hours. Thus, it can be seen that $\frac{1}{\mu_3}$ is $\frac{1}{\mu_1}\frac{934}{147,456} = 0.30$ hours, implying that processing the delayed parity groups in $BPGL$ is about 160 times faster than reconstructing all of the parity groups in a faulty disk.

Finally, the obtained realistic values for $\frac{1}{\lambda_2}$, $\frac{1}{\mu_2}$, and $\frac{1}{\mu_3}$ by regenerating traced I/O requests of the `cello` system are 0.074, 0.004, 0.30 hours, respectively, where MTTF of a disk $\frac{1}{\lambda_1} = 1,000,000$ hours and MTTR of a disk $\frac{1}{\mu_1} = 48$ hours. Next, Figure 12 shows that the reliability of the proposed architecture when those realistic values are applied to Equation (4). It is revealed that the reliability of the proposed architecture is extremely high, *i.e.*, about 298–436 times that of RAID5 and 0.04–0.34 times that of RM2.

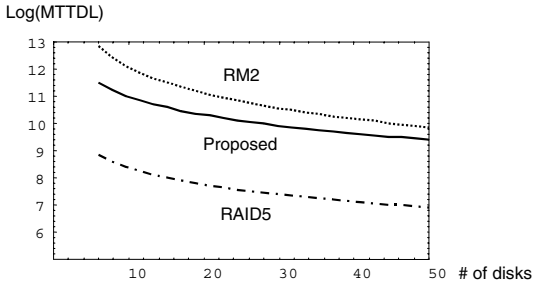


Figure 12. Comparing the reliability of RAID5 and RM2 with the reliability of the proposed architecture configured with the empirically obtained values of $\frac{1}{\lambda_2} = 0.074$, $\frac{1}{\mu_2} = 0.004$, and $\frac{1}{\mu_3} = 0.30$

4. Concluding Remarks

In order to overcome the low write I/O performance of RM2, a well-known RAID6 architecture, we employed a *Lazy Parity Update* (LPU) technique which loosens the strict reliability requirement of tolerating double disk failures at any given time. We implemented the proposed architecture on top of an actual RAID system and then thoroughly evaluated its I/O performance and reliability. I/O throughput measurements revealed that the proposed architecture improves the write I/O performance of RM2 by more than two times. Reliability measurements by theoretical analysis and regeneration of a traced real I/O workload showed that the proposed architecture continuously provides extremely high reliability, *i.e.*, about 298–436 times the reliability of RAID5 and 0.04–0.34 times the reliability of RM2.

In future work, we will conduct more experiments on different testing environments with an odd number of disks which demands a relatively larger FPG compared with an even number of disks, a larger number of disks, and different types of traced I/O workloads. In addition, we plan to devise a more sophisticated idle detection mechanism based on previous work [5, 11] and apply the LPU technique to other RAID6 architectures, such as P+Q, EVEN-ODD, and DATUM.

Acknowledgments

The authors would like to thank the Ministry of Education of Korea for its financial support through its BK21 program.

References

[1] St39175fc product manual. Web document. URL: *http://www.seagate.com/support/disc/fc/st39175fc.html*.

[2] G. Alvarez and *et. al.* Tolerating multiple failures in raid architectures with optimal storage and uniform declustering. In *Proceedings of the 24th ISCA*, June 1997.

[3] R. Billinton and R. Allan. *Reliability Evaluation of Engineering System: Concepts and Techniques*. Pitman Advanced Publishing Program, Boston, 1992.

[4] M. Blaum and *et. al.* Evenodd: An efficient scheme for tolerating double disk failures in raid architectures. *IEEE Transactions on Computers*, 44(2), February 1995.

[5] R. Golding and *et. al.* Idleness is not sloth. In *Proceedings of Winter USENIX*, pages 201–212, January 1995.

[6] P. Massiglia. *The RAID Book*. RAID Advisory Board, 6 edition, 1997.

[7] Y. Nam and *et. al.* Enhancing write i/o performance of disk array rm2 tolerating double disk failures. Technical Report CSE-SSL-2002-02, POSTECH, Pohang, Kyungbuk, Republic of Korea, January 2002.

[8] C. Park. Efficient placement of parity and data to tolerate two disk failures in disk array systems. *IEEE Transactions on Parallel and Distributed Systems*, pages 76–86, November 1995.

[9] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks(raid). In *Proceedings of IEEE COMPCON*, pages 112–117, Spring 1989.

[10] C. Ruemmler and J. Wilkes. Unix disk access patters. In *Proceedings of Winter USENIX*, pages 405–420, January 1993.

[11] S. Savage and J. Wilkes. Afraid - a frequently redundant array of independent disks. In *Proceedings of USENIX Technical Conference*, January 1996.

[12] A. Thomasian. Performance analysis of raid5 disk arrays. *Tutorial Material, SIGMETRICS/PERFORMANCE*, June 1998.