Lazy Parity Update : A Technique to Improve Write I/O Performance of Disk Array 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

1 Introduction

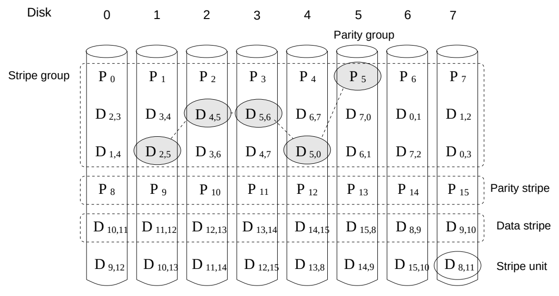
There have been a few RAID algorithms that can tolerate double disk failures by elaborately maintaining two distinctive parity information associated with user data [1, 3, 4]. This breed of algorithms is formally classified as RAID level 6. However, RAID level 6 has suffered from relatively low write I/O performance comparing with other RAID levels [5] at the expense of providing higher reliability by maintaining additional parity information. While RAID level 5 requires four disk accesses, RAID level 6 demands six disk accesses to process a small write I/O request. Thus, RAID level 6 might provide about 66% of write I/O performance of RAID level 5.

Let us briefly consider the following recent technology trends. Large-scale RAID systems containing a large number of disks become prevailing with the advent of SAN using Fiber Channel and Gigabit Ethernet. These systems are much more susceptible to multiple disk failures than a small-scale system, though. Disk capacity has been growing rapidly, resultingly a disk recovery time gets longer which proportionally increases the chance of a subsequent disk failure. Thus, we expect that RAID level 6 algorithms will be gaining its importance. Having said that, customers will be still reluctant to sacrifice write I/O performance by keeping additional parity information to increase reliability of a RAID system. In order to overcome this fundamental problem of RAID level 6, we aim at devising a technique to dramatically enhance a write I/O performance of RAID level 6 by delaying the process of updating one of two parity information, called the *Lazy Parity Update* technique. We apply this technique to a RAID level 6 algorithm RM2 and evaluate its performance in terms of reliability and I/O throughput.

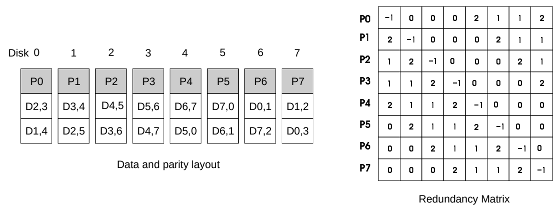
2 The Proposed Algorithm

RM2 [4] is used as our base algorithm among the existing RAID level 6 algorithms. After briefly describing the features of RM2, let us present a *Lazy Parity Update* technique to improve write I/O performance of RM2. Figure 1(a) presents data/parity placement in RM2. A *data stripe* represents a stripe containing only data stripe units and a *parity stripe* means a stripe containing only parity stripe units. A *parity group* is a set of data stripe units and a parity stripe unit where the parity stripe unit is calculated from the set of data 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. This data/parity placement of RM2 is mainly determined based on the *redundancy matrix* which maps each stripe unit to its corresponding two parity stripe units within a stripe group for 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 ($1 \leq k \leq M-1$) where M is a stripe group size. 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. P_i represents the parity stripe unit in the i-th parity group and $D_{i,j}$ means that the data stripe unit is involved in computing parity information of both the i-th and j-th parity groups. As an example in Figure 1(a), the the parity unit P_5 is related with the four data units which are $D_{2,5}$, $D_{4,5}$, $D_{5,6}$, and $D_{5,0}$. Conversely, the data unit $D_{2,5}$ is related with P_2 and P_5.

Let us introduce new nomenclature which will be used in the Lazy Parity Update technique. *Foreground Parity Group*(FPG) refers to as a subset of



(a) Data/parity distribution



(b) Data/parity layout and redundancy matrix

Figure 1: Architecture of the RM2 algorithm

$\mathcal{P} = \{P_i | 0 \leq i \leq N-1\}$ such that it can cover all data stripe units in a stripe group. Especially, the minimum set of FPG is denoted as FPG_{min}. Also, the number of parity groups in FPG_{min} is written as $|FPG_{min}|$. *Background Parity Group*(BPG) is defined as $\mathcal{P} - FPG$. In Figure 2, P_0, P_2, P_4, P_6 fall into FPG when $N = 8$. We have proved that $|FPG_{min}|$ is $\frac{N}{2}$ if $N =$ even and $\lceil \frac{N}{2} \rceil + M - 2$ if $N =$ odd for given N disks, where M is a stripe group size.

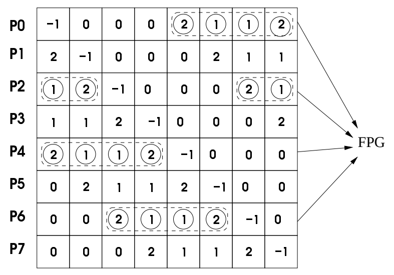


Figure 2: An illustrative example of configuring FPG and BPG with $N = 8$ and $M = 3$

Lazy Parity Update (LPU) Technique : The proposed algorithm makes an exploit of the LPU technique on the basis of the existing RM2. The key idea of the LPU scheme is to update only one of two parity groups associated with data in foreground while delaying the process of the other parity group until the system becomes idle. For this, the LPU scheme needs to initially determine FPG and BPG sub-sets for given N disks and then to process each write I/O request based on the FPG and BPG information. We have developed a scheme which determines the FPG_{min} for given N disks. A description of this scheme is omitted in this paper. We assume that a write-through buffer cache is used, first. As shown in Figure 3, given two parity groups related with data, if a parity group is included in FPG, then it should be updated before a completion message is returned to its host. If a parity group belongs to BPG, its parity group number is added to Background Parity Group List ($BPGL$), *i.e.*, postponing its process until a RAID system becomes idle.

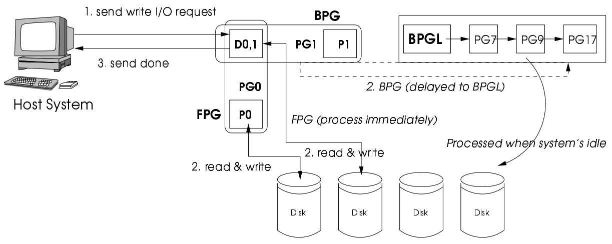


Figure 3: Key idea of the Lazy Parity Update technique

With a write-back buffer cache, the above behavior occurs when destaging process begins, resultingly destaging rate can be doubled theoretically. Once a disk failure occurs, however, two parity groups associated with data are regarded as being included in FPG. Thus, processing of parity groups is no longer delayed. In the presence of a faulty disk, a RAID system gets ready to tolerate an additional disk failure by processing all delayed parity groups in $BGPL$, not by reconstructing the whole associate parity groups in a faulty disk. This is why the proposed algorithm is much more reliable than RAID level 5.

3 Preliminary Performance Results

Let us provide a brief description of our experimental environment and preliminary results on reliability and I/O throughput of the proposed algorithm which enhances write I/O performance of RM2 with the LPU technique.

Current Experimental Environment : The proposed algorithm has been implemented on a real RAID system called PosRAID along with RAID level 5 and RM2. Hardware components of the PosRAID encompass Pentium III 800MHz, 256MB memory, two QLogic's QLA2200 FC cards, 32bit/33MHz PCI bus, and 8 FC disks. Note that the PosRAID does not support a hardware XOR. Software components of the PosRAID include VxWorks version 5.4 RTOS, a communication module in charge of communicating with hosts and internal disks via 1Gbps FC HBAs, a RAID engine module mapping a logical block address to a physical block address according to a RAID algorithm and managing a buffer cache, and a resource/configuration management module taking care of allocating, deallocating, and configuring all hardware and software resources within a system. For the measurement of I/O throughput, sizes of LUNs configured by different RAID algorithms are set to equal. In result, the number of disks used in RAID level 5, RM2, and the proposed algorithm are different, *i.e.*, while RAID level 5 uses three disks, RAID Level 6(RM2) and the proposed algorithm does four disks. Also, each disk capacity is 3GB and a stripe unit size is 64KB for all LUNs.

Reliability Analysis : We analyze reliability ($MTTF_{proposed}$) of the proposed algorithm by using the state transition diagram shown in Figure 4 and a stochastic transitional probability matrix scheme [2].

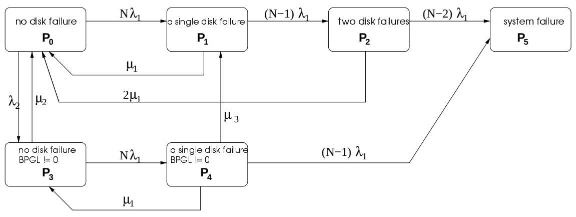
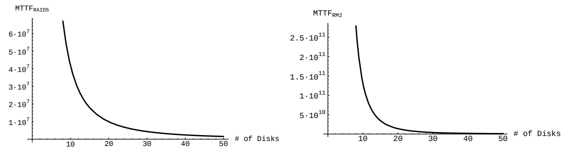


Figure 4: State transition diagram for the proposed algorithm

It can be expected that the reliability of the proposed algorithm will vary from that of RAID level 5 at the worst case to that of RAID level 6 at the best case, depending on increasing and decreasing rates of $BGPL$ (λ_2, μ_2) and processing rates of $BGPL$ in the presence of a disk failure (μ_3). By regenerating I/O requests in traced I/O requests of `cello` [6], we obtain $\frac{1}{\lambda_2} = 0.26$, $\frac{1}{\mu_2} = 0.009$, and $\frac{1}{\mu_3} = 0.74$. Figure 5 and 6 presents that the reliability of the proposed algorithm using those realistic values is very high, about 0.2 times of RAID level 6 (RM2) and 1000 times of RAID level 5.



(a) RAID level 5 (b) RM2

Figure 5: Reliability of RAID level 5 and RAID level 6 with the increase of the number of disks, where MTTF of a disk $\frac{1}{\lambda_1} = 30,000$ and MTTR of a disk $\frac{1}{\mu_1} = 24$ (unit : hour)

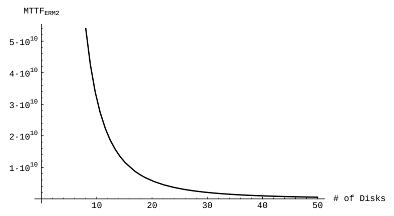
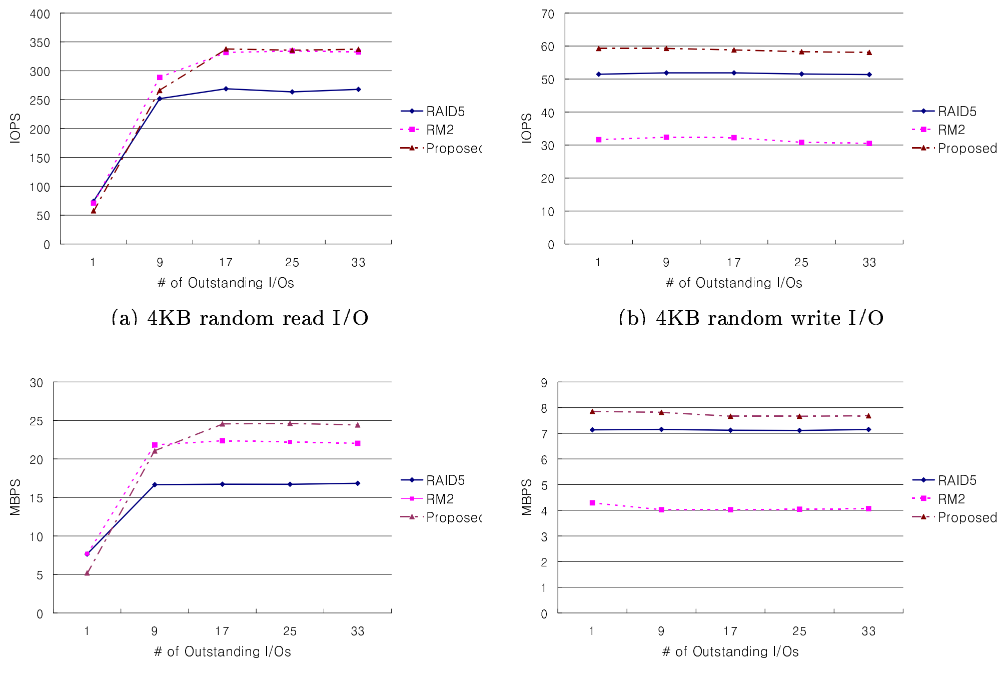


Figure 6: Reliability of the proposed algorithm with empirically obtained $\frac{1}{\lambda_2} = 0.26$, $\frac{1}{\mu_2} = 0.009$, and $\frac{1}{\mu_3} = 0.74$, where MTTF of a disk $\frac{1}{\lambda_1} = 30,000$ and MTTR of a disk $\frac{1}{\mu_1} = 24$ (unit : hour)

I/O Throughput Analysis : For I/O throughput measurement, a well-known I/O benchmark (IOMeter) issues 4KB and 128KB random I/O workloads to each LUN configured with a different RAID algorithm. The number of concurrent I/O processes so called queue depth increases every 180 seconds including 60-second warm-up time. Figure 7(a)–(b) show that read I/O performance of RM2 and the proposed algorithm is better than that of RAID level 5. This is due to the use of extra disk in both algorithms. As for write I/O performance, the proposed algorithm using the LPU technique improves write I/O performance of RAID level 6 (RM2) by 100%. It also outperforms write I/O performance of RAID level 5 due to the existence of an additional disk as in read I/O performance.

4 Status Quo and Future Work

Our on-going and near-term work puts an emphasis on performing the same experiment with differ-



(a) 4KB random read I/O

(b) 4KB random write I/O

(c) 128KB random read I/O

(d) 128KB random write I/O

Figure 7: I/O performance of RAID level 5, RAID level 6(RM2), and the proposed algorithm under no disk failures for given 4KB/128KB random read/write requests

ent configurations and devising a more sophisticated idle detection scheme.

- We are using other traced I/O requests to measure more accurate reliability of the proposed algorithm, various types of I/O workload patterns to obtain write I/O performance of different algorithms, and different configurations of LUNs including the odd number of disks, the fixed number of disks for all algorithms, and a larger number of disks.
- Detecting system idleness relies on a real-time scheduling facility of an underlying RTOS by giving a relatively lower priority to a task which processes delayed parity groups in *BPGL*. This idle detection mechanism should be enhanced.

Finally, our long-term future work includes the following items:

- The LPU technique will be applied to other RAID level 6 algorithms such as EVEN-ODD, DATUM, and P+Q.
- We will look into the problem of enhancing performance in the presence of a disk failure.

References

[1] G. Alvarez, *et. al.*, “Tolerating multiple failures in RAID architectures with optimal storage and uniform declustering,” *Proceedings of the 24th ISCA*, Jun. 1997.

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

[3] M. Blaum and *et. al.*, “EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures,” *IEEE Transactions on Computers*, vol. 44, no. 2, Feb. 1995.

[4] C. Park, “Efficient placement of parity and data to tolerate two disk failures in disk array systems,” *IEEE Transactions on Parallel and Distributed Systems,* Vol. 6, No. 11, pp. 1177–1184, November 1995.

[5] D. Patterson, G. Gibson, and R. Katz, “A case for redundant arrays of inexpensive disks(RAID),” *Proceedings of IEEE COMPCON*, pp. 112–117, Spring 1989.

[6] C. Ruemmler and J. Wilkes, “Unix disk access patters,” *Proceedings of Winter USENIX,* pp.405–420, Jan. 1993.