Dynamic parity logging disk arrays for engineering database systems

K.H. Yeung
T.S. Yum

Indexing terms: Disk Arrays, Database systems, Performance Evaluation, Dynamic parity logging

Abstract: RAID (redundant arrays of inexpensive disks) has gained much attention in the recent development of fast I/O systems. Of the five levels, the traditional mirrored disk array still provides the highest I/O rate for small 'write' transfers. This is because the mirrored disk array has no small 'write' problem which is found in other levels of RAID. The authors propose a novel RAID architecture for fast engineering database systems, called dynamic parity logging (DPL) disk array. DPL disk array has no small 'write' problem and can provide much higher 'write' throughput than other RAID architectures. DPL disk array also has journalling capability, which means that some older design versions are kept for future references. A queueing model for DPL disk array is built. Analytical results, supported by simulation, show that the DPL disk array can provide the highest 'write' throughput when compared to RAID levels 1, 4, and 5.

1 Introduction

As processor speed continues to increase, the I/O performance of a computer system was recognised to be more and more crucial to the overall system performance. Redundant arrays of inexpensive disk (RAID) systems were proposed in the late 1980s to improve the I/O performance, and are now commonly used in operating systems such as Windows NT™. Five levels of RAID have been defined when RAID was first introduced [1]. RAID level 5, one of the best performing levels, can yield very high throughput for large data transfers. However, the throughput reduces significantly if the proportion of 'write' transfers increases for such systems. This is because 'write' transfers require the extra steps of reading back the old data and the old parity, and the writing in of the new parity. This is commonly called the *small 'write' problem* [2]. Moreover, each 'write' transfer requires the simultaneous access of two or more disks and thus has a much higher probability of blocking by busy disks than a 'read' transfer. The average waiting time for all the required disks to be free in a data transfer, called the *blocking time*, is therefore longer for 'write' transfers. Owing to these two problems, the traditional mirrored disk array still provides the faster response than RAID level 5 for database applications. Extensive research efforts have been made on improving the 'write' performance of RAID for conventional database systems. A survey on these can be found in [3].

Unlike conventional database systems, similar research on RAID for specific types of database systems is relatively limited. In this paper, we focus on engineering database systems (EDS) and discuss the design of a fast disk array architecture for these systems. A new architecture called dynamic parity logging (DPL) disk arrays is proposed. As discussed in [4–7], EDS has the unique ways of processing data and the unique data storage requirements when compared to conventional database systems. These unique characteristics of EDS are carefully studied in the design of DPL disk array. Without sacrificing the high storage utilisation (as found in RAID level 5), DPL disk array aims at solving the small 'write' problem and reducing the blocking time for 'write' transfers. DPL disk array also has journalling capability, which means that some *older* design versions are kept for future references.

© IEE, 1997
IEE Proceedings online no. 19971466
Paper first received 30th July 1996 and in final revised form 15th April 1997

K.H. Yeung is with the Department of Electronic Engineering, City University of Hong Kong, Yau Yat Chuen, Hong Kong
T.S. Yum is with the Department of Information Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong

2 DPL disk array architecture

2.1 Overview of DPL disk array

A DPL disk array has a total of $N + 2$ disks (Fig. 1). Disks 1 to N are for data storage and are called the *data disks* while disks $N + 1$ and $N + 2$ are mirrored disks and are called the *parity disks*. This pair of parity disks is used to store the parity blocks of *parity sets*. A parity set is defined to be a collection of blocks for which the parity sum of these blocks is always maintained to be '1' for odd parity and '0' for even parity. In conventional RAID architectures, parity sets are static and are usually formed by the blocks which are on the same block row. However, parity sets in a DPL disk array are *dynamically* assigned when block updates are performed.

The working principle of DPL disk array for 'write' transfers is illustrated in Fig. 1. In our discussions, we shall denote block j in disk i as $B(i, j)$, and its contents $b(i, j)$. Fig. 1 shows *five* successive updates on *four* different blocks $B(3, 79)$, $B(6, 93)$, $B(8, 28)$ and $B(6, 25)$. Suppose the contents of these blocks have not been

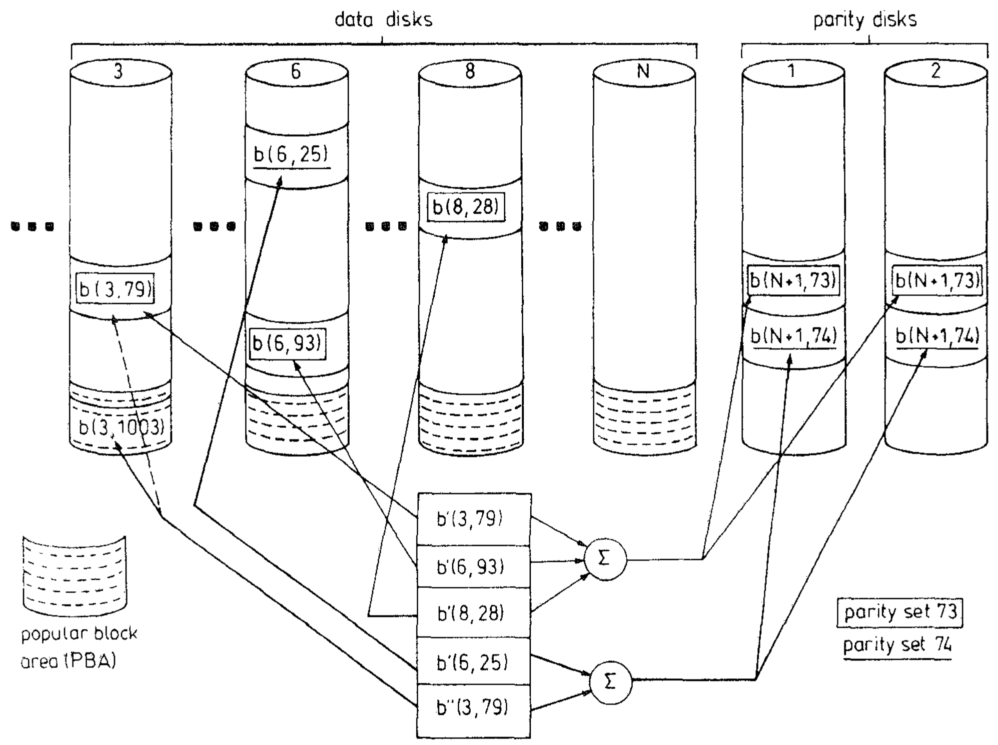

Fig. 1 *Example showing the working principle of DPL disc array for 'write' transfers*

a

entry 1	entry 2		entry N	entry N+1
block identifier of the first member block	block identifier of the second member block		block identifier of the N th member block	parity sum S

b

B(3,79)	B(6,93)	B(8,28)	empty		S=b(N+1,73)=b'(3,79)⊕b'(6,93)⊕b'(8,28)

Fig. 2 *Structure of parity blocks showing the example of Fig. 1*
(*a*) Parity block format
(*b*) Parity block of parity set 73

B(6,25)	B(3,1003)	empty		S=b(N+1,74)=b'(6,25)Ob''(3,79)

Fig. 3 *Structure of parity blocks showing the example given in Fig. 1*
(*c*) Parity block of parity set 74

modified since the last system backup and the new contents for the five updates are identified as $b'(3, 79)$, $b'(6, 93)$, $b'(8, 28)$, $b'(6, 25)$ and $b''(3, 79)$ respectively as shown in the Figure. Noting that $B(6, 93)$ and $B(6, 25)$ are located in the same disk, and $B(3, 79)$ is modified twice. Since second copies of the original block contents $b(3, 79)$, $b(6, 93)$, $b(8, 28)$ and $b(6, 25)$ have been stored in the tertiary storage, they can be restored if necessary. These blocks can therefore be overridden by new contents. The new contents however should be protected by additional parity. Therefore, the mod-2 sum of the new contents of the first three updates, i.e. $b'(3, 79) \otimes b'(6, 93) \otimes b'(8, 28)$, is computed and saved to the parity disks as a parity block. Fig. 2 shows the structure of a parity block. It consists of a block header which holds the block identifiers of its member blocks, and a parity sum of its member blocks. The parity block for the first three updates of our example is shown in Fig. 2*b*. A parity set consisting of three data blocks and one parity block is thus formed. This parity set is identified as parity set 73 in Fig. 1. These blocks can be assigned to the same parity set because they are located on different data disks. If one of these data disks fails, the lost blocks in the failed disk can be recovered from the remaining blocks of the parity set. This parity set cannot include the next updated block $B(6, 25)$ because the parity set already has a member block $B(6, 93)$ in disk 6. We can, however, form the next parity set 74 starting from this block update. In our example, parity set 74 consisting of $B(6, 93)$, and $B(3, 1003)$, and their parity sum $b'(6, 25) \otimes b''(3, 79)$ is saved to the parity disks (Fig. 3). Note that block $B(3, 79)$ was updated before. Its current contents $b'(3, 79)$ therefore cannot be overridden. Instead, we store $b''(3, 79)$ to a reserved area of the disk called the popular block area (PBA) at location $B(3, 1003)$. (Blocks which are updated more than once after backup are called the popular blocks). As shown in Fig. 1 each data disk has a PBA for storing new updates of popular blocks in that disk. When a popular block is written to the PBA, a block header pointing to its original location on the disk is also written. This pointer is used for restoring the content of the block when system backup is performed.

If there is only one parity disk, all parity information is lost when the parity disk fails and a very time consuming backup of all data disks is required. This can be avoided by using a pair of mirrored disks to store the parity information. Table 1 summarises the properties of DPL disk array. These properties are mostly self evident from the previous discussions.

Table 1: Properties of DPL disk array

1. These is no small 'write' problem
2. The blocking time for 'write' transfers is much smaller than that of other RAID architectures
3. The parity disks contain a journal of block updates
4. Data will not be lost in single disk failures
5. The log volume will be significantly smaller than the updated volume
6. The parity disks work sequentially under all conditions
7. No data is lost when the I/O controller fails
8. DPL Disk Array requires a brief period of data restoration in case of disk failures

2.2 DPL disk array operation

Fig. 4 shows the organisation of DPL disk array. Requests for data transfers are sent from host to the I/O controller for I/O operations. The I/O controller consists of seven parts: an I/O queue, a local memory, an I/O processor, a DMA controller, a block location table, N data disk counters, and a parity disk counter.

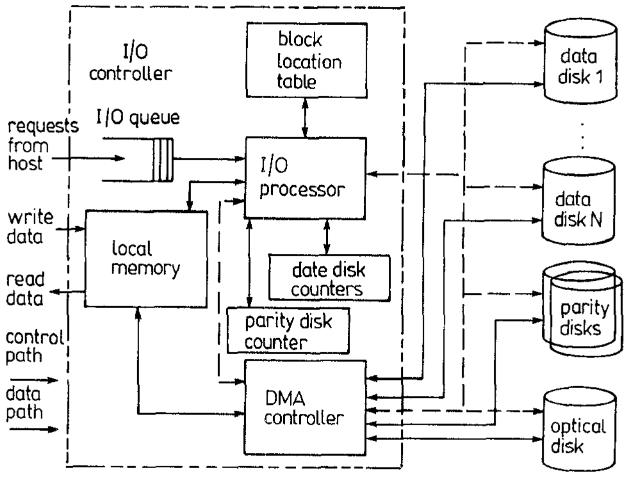

Fig. 4 *Organisation of DPL disk array*

The I/O queue is used for storing I/O requests and the requests are served in a FCFS manner. The data associated with a request (i.e. the new data for a block) is stored in the local memory when the request is placed on the queue. The local memory is also used for buffering the data sent to or read from each of the disks and for storing all temporary data for processing. The DMA controller functions as a multiplexer/demultiplexer and handles simultaneous data transfers to various disks. The I/O processor is the heart of the I/O controller. It is responsible for distributing works to the disks for data transfers. It also performs necessary data processing works such as parity computation. The block location table is used for locating data blocks in the disk array. It is a $K_D \times N$ table, where K_D is the number of data blocks per disk. If there are K blocks in a disk, the size of the PBA is equal to $K_P = K - K_D$ blocks. Each data block has its own entry in the block location table, and each entry consists of a status bit and a block address. For block $B(_{i,j})$ the contents of the status bit and the block address are denoted as $s(B(_{i,j}))$ and $a(B(_{i,j}))$ respectively. The status bit $s(B(_{i,j})) = 1$ indicates that the block has been updated since last system backup and $s(B(_{i,j})) = 0$ indicates otherwise. The block address points to the current location of the block in the disk array. In particular, $a(B(_{i,j})) = 0$ indicates that $b(_{i,j})$ is the most recently updated contents of $B(_{i,j})$ and $a(B(_{i,j}))$ points to a location in the PBA indicates that $B(_{i,j})$ has been updated more than once. One thing worth noting is the size of the block location table. For a typical disk having 2^{19} blocks (which corresponds to a 1GB drive with a block size of 2KB), each entry in the table require 19 + 1 bits. The size of the block location table for a $N = 25$ disk array system is $20 \times 25 \times 2^{16}$ bytes, or 32MB. By using today's flash RAM technology, it is practical to implement it on nonvolatile memory units [8]. It is also possible to use hashing techniques to reduce the size of this table. Its effect on performance, however, is a topic for further research. To facilitate the management of PBA, the I/O controller maintains a pointer for each data disk pointing to the next available block in the PBA. We denote the pointer value for data disk i by c_i, where $K_D < c_i \leq K$. Each time the I/O processor attempts to write a popular block to PBA, the corresponding data disk pointer is read to give the appropriate location for the block. That pointer value is then incremented by one to point to the next available block in the PBA. A similar pointer c_p is maintained for the parity disks which points to the location for the next parity block writing.

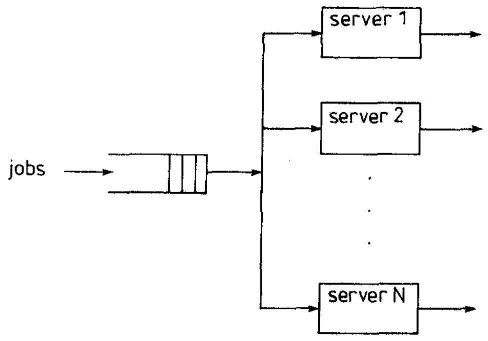

Fig. 5 *Queueing models for DPL disk array*

3 Performance of DPL disk array

Fig. 5 shows the queueing model for a DPL disk array. Requests for data transfers are sent from host and become *jobs* for the servers. Job arrivals are assumed to be a Poisson process with rate λ, and are assumed to target uniformly on all disks. The probability that a job targets on a particular block of a disk is also assumed to be the same for all blocks. A job is of the 'write' type with probability α and of the 'read' type with the remaining probability. As we have mentioned before, one parity block is written to the parity disks only when the system finishes the construction of a parity set. When a parity set is still in construction, the parity sum of the parity set is kept in the nonvolatile memory. Therefore, we only need to consider the data disk operation for each 'write' job because the access time of the nonvolatile memory is much faster than that of the data disks. For mathematical convenience we assume that the service time for a job is exponentially distributed with service rates μ_w and μ_r for 'write' jobs and 'read' jobs, respectively. We also assume that the I/O controller is fast enough so that it does not affect the system performance.

3.1 Mean system backup time

Suppose the system finishes backing up the data and switches to normal mode at time 0. All blocks are marked as unchanged at that time. Since all blocks are requested with the same probability, the rate of 'write' arrivals targeting on a particular block γ is given by:

$$\gamma = \frac{\text{Rate of "write" arrivals to the disk array}}{\text{Total number of blocks in the disk array}} = \frac{\alpha\lambda}{NK_D} \quad (1)$$

Let p_k be the probability that there are k 'write' requests in $(0, t)$ targeting on a particular block. With the Poisson arrival assumption, we have

$$p_k = \frac{(\gamma t)^k}{k!} e^{-\gamma t} \quad (2)$$

Consider block i in disk j. Let $X_i(t)$ be the number of block i images written to PBA of disk j in $(0, t)$. For $k + 1$ arrivals k images are written to PBA, we have

$$p[X_i(t) = k] = \begin{cases} p_0 + p_1 & \text{for } 0 \le k \le 1 \\ p_{k+1} & \text{for } k \ge 2 \end{cases} \quad (3)$$

The expected value of $X_i(t)$ is given by:

$$E[X_i(t)] = \gamma t - (1 - p_0) \quad (4)$$

The variance of $X_i(t)$ is given by:

$$VAR[X_i(t)] = \sum_{k=0}^{\infty} P[X_i = k]k^2 - E[X_i]^2 = [\gamma t - (1 - p_0)]^2 \quad (5)$$

Next, let random variable $X(t)$ denotes the total number of blocks written to PBA of disk j in $(0, t)$, or

$$X(t) = \sum_{i=1}^{K_D} X_i(t) \quad (6)$$

As the X_is are independent and identically distributed random variables, by central limit theorem the distribution of $X(t)$ can be well approximated by a Gaussian distribution with mean $m = K_D[\gamma t - (1 - p_0)]$ and variance $\sigma^2 = K_D[\gamma t - (1 - p_0)]^2$ if K_D is not too small. Therefore, the probability $r(t)$ that the PBA of disk j does not overflow in $(0, t)$ is given by:

$$r(t) = P[X(t) \le K_p] = \int_{-\infty}^{K_p} \frac{e^{-(x-m)^2/(2\sigma^2)}}{\sqrt{2\pi}\sigma} dx \quad (7)$$

Finally, the probability that the PBAs of all N disks do not overflow in $(0, t)$ is simply given by $r^N(t)$. Note that system backup should be performed if PBA of any one of the disks overflows. If we let random variable T be the time which the system has to perform system backup, the mean system backup time T_B is given by:

$$T_B = \int_0^{\infty} (1 - P[T \le t])dt = \int_0^{\infty} (1 - (1 - r^N(t)))dt = \int_0^{\infty} r^N(t)dt \quad (8)$$

As an example, consider a DPL disk array with $N = 10$ and $K_D = 50{,}000$. Fig. 6 plots the mean system backup time T_B (in days) against the 'write' arrival rate $\alpha\lambda$ for three different values of K_p. Simulation was also performed (not shown in the Figure) and it verified the analytical results. From Fig. 6, we observed that T_B is inversely proportional to $\alpha\lambda$ for all values of K_p shown. We also observed that T_B is increased by about 45% when the size of the PBA is doubled for all 'write' arrival rates shown. If T_B and the maximum number of blocks being written between successive system backups are given, K_p can be determined from Fig. 6. Take for an example, if $T_B = 5$ (i.e. weekly backup is performed for five working days per week) and the maximum number of block updates per day is 50,000 blocks, the minimum value of K_p should be about $0.1K_D$ or 5,000 blocks. If the maximum number of block updates per day grows to about 70,000 blocks, K_p should be at least $0.2K_D$ or 10,000 blocks. For systems with very high update rates, says 350,000 blocks per day (or on the average 70% of the blocks will be updated once per day), daily backup should be performed for a K_p value of about $0.2K_D$.

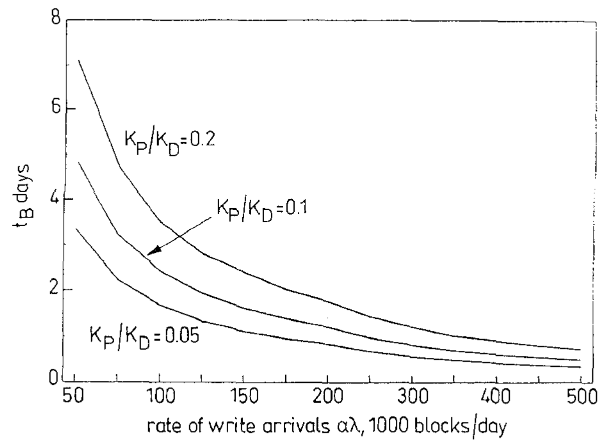

Fig. 6 *Mean system backup time against the rate of 'write' arrivals*

3.2 Utilisation of the parity disks

Consider a particular parity set A_s and let $|A_s|$ be the number of elements in A_s. Then

$$P[|A_s| = m] = P\begin{bmatrix}\text{The first}\\ \text{block is}\\ \text{new to } A_s\end{bmatrix} P\begin{bmatrix}\text{The second}\\ \text{block is}\\ \text{new to } A_s\end{bmatrix} \cdots P\begin{bmatrix}\text{The } m^{\text{th}}\\ \text{block is}\\ \text{new to } A_s\end{bmatrix} P\begin{bmatrix}\text{The } (m+1)^{\text{th}}\\ \text{block is not}\\ \text{new to } A_s\end{bmatrix}$$
$$= 1 \times \frac{N-1}{N} \times \frac{N-2}{N} \times \ldots \times \frac{N-(m-1)}{N} \times \frac{m}{N} = \frac{m(n-1)!}{N^m(N-m)!} \quad (9)$$

The expected size L_s of parity set A_s is therefore

$$L_s = \sum_{m=1}^{N} mP[|A_s| = m] = \sum_{m=1}^{N} \frac{m^2(N-1)!}{N^m(N-m)!} \quad (10)$$

Since only one parity block is generated for each parity set, the average number of parity blocks generated in T_B, denoted as B_p, is therefore

$$B_p = \frac{\alpha\lambda T_B}{L_s} \quad (11)$$

For a small disk array of four data disks, we find that the log volume is about 45% of the update volume ($1/L_s \approx 0.45$). For a large disk array of 24 data disks, the log volume reduces to about 17% of the update volume ($1/L_s \approx 0.17$).

3.3 Average delay

As discussed before, DPL disk array has no small 'write' problem. This is because blocks can be directly written to the disks without reading back the old parity and old data. Therefore, it is reasonable to assume that the same average delay is experienced by the 'write' jobs and 'read' jobs. With this assumption, we can use the queueing model for RAID level 5 when $\alpha = 0$ to study the performance of DPL disk array. A detailed analysis of RAID level 5 using exponential servers is reported in [3]. By using the same analysis (with different numbers of disks, i.e. $N + 1$ for RAID level 5 and N for DPL disk array), we obtain the average delay for DPL disk array.

Fig. 7 shows the average job delays for DPL disk array and RAID level 5 when each architecture has four data disks. We assume in this example that $\mu_w = \mu_r = 50$ jobs/sec for DPL disk array. For RAID level 5, the service rates for 'write' jobs and 'read' jobs are assumed to be 30 jobs/sec and 50 jobs/sec, respectively. Since 'write' operations in RAID level 5 need to read back old parity and data, the corresponding service rate is lower due to extra disk rotation. We find from Fig. 7 that DPL disk array outperforms RAID level 5 for most values of α. This is because there is no small 'write' problem in DPL disk array and the 'write' jobs for DPL disk array only require the access of one disk. When $\alpha = 1$, DPL disk array provides maximum throughput which is 2.5 times higher than that of RAID level 5. When all the jobs are of the 'read' type, we observe that RAID level 5 performs slightly better than DPL disk array because data is distributed into five disks for RAID level 5 and is only distributed into four disks for DPL disk array. The difference in performance for $\alpha = 0$ will be smaller for larger Ns for obvious reasons.

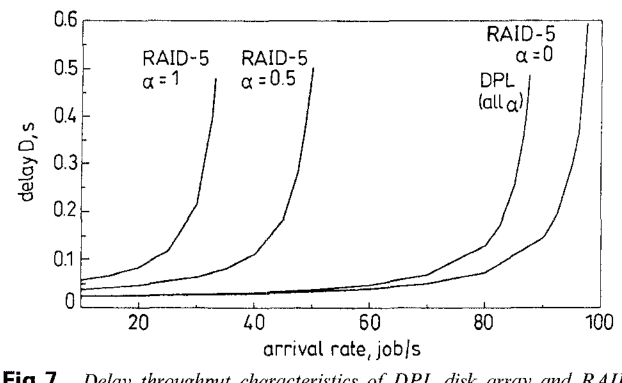

Fig. 7 *Delay throughput characteristics of DPL disk array and RAID level 5*

3.4 Throughput performance using a precise disk model

In our previous analysis disk service time is assumed to be exponentially distributed. This is usually not true for practical disk drives. To better understand the performance of DPL disk array and other RAID architectures in practice, we perform throughput simulation on different RAID architectures by using a precise disk model. In our simulation, disks are not assumed to be rotationally synchronised and their simulation parameters are summarised in Table 2. Each disk access involves a seek time, a latency and a data transfer time. We use the seek profile reported in [9], and the latency is assumed to be uniformly distributed. Data transfer time for one sector is equal to the disk revolution time divided by the number of sectors per track as given in Table 2. With that, the mean service time for all jobs is computed to be 20ms. As stated in [10], this kind of disk modelling provides more than 94% accuracy when ignoring the disk caching effect. Since disk caching has little impact on 'write' performance (which we are most interested in), we can thus assume that the system has no disk caching mechanism.

Table 2: Disc parameters used in simulation

Cylinders per disk	1024	Revolution time	13.3ms
Tracks per cylinder	14	Single cylinder seek time	2ms
Sectors per track	48	Average seek time	13ms
Bytes per sector	512	Max. data transfer rate	1.7MB/s
Block size	2KB		

Fig. 8 shows the maximum throughput per disk when all four architectures use the same total number of 24 disks. When $\alpha = 0$, we observe that RAID level 1 provides twice the throughput when compared to other three architectures. The trade-off for this good performance of RAID level 1 is the reduction of storage capacity by half. When the proportion of 'write' jobs increases, we find that the throughput of DPL disk array remains constant while that for the other three architectures drop significantly. When $\alpha \geq 0.4$, DPL disk array provides the highest maximum throughput per disk. The reason is obvious that for each 'write' operation, all the other three architectures require two disk updates while DPL disk array requires only one. From these results we can conclude that DPL disk array provides the best 'write' performance. We have also performed another simulation and the results show that for DPL disk array, at most 10% drop in throughput is observed for all values of α and $\beta \leq 0.4$. The performance of DPL disk array is therefore quite insensitive to β.

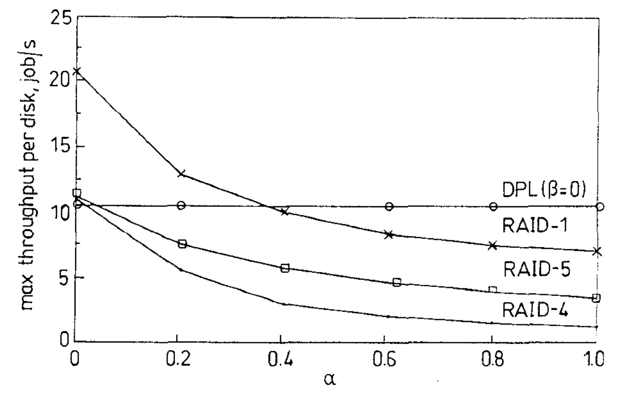

Fig. 8 *Maximum throughput comparison for various RAID architectures*

4 Conclusions

We propose in this paper a new RAID architecture called dynamic parity logging disk array for fast EDS.

DPL disk array solves the small 'write' problem found in most RAID levels and can significantly reduce the blocking time for 'write' transfers. It also has the journalling capability which is very desirable for EDS. Analytical results on DPL disk array shows that it can provide much faster 'write' response than RAID level 5. Simulation using a precise disk model also showed that DPL disk array provides the highest 'write' throughput when compared to RAID levels 1, 4, and 5.

5 Acknowledgment

We gratefully acknowledge the many helpful suggestions of three reviewers which leads to a substantial improvement of the readability of this paper. We also gratefully acknowlege the support of Hong Kong Research Grants Council through Grant No. RGC9040213.

6 References

1 PATTERSON, D., GIBSON, G., and KATZ, R.: 'A case for redundent arrays of inexpensive disks (RAID)'. Proceedings of the ACM SIGMOD conference, 1988, pp. 109–116
2 GIBSON, G.A.: 'Redundant disk arrays: reliable, parallel secondary storage' (MIT Press, 1992)
3 YEUNG, K.H.: 'High performance disk array architectures'. PhD dissertation, Chinese University of Hong Kong, 1995
4 KATZ, R.H.: 'Information management for engineering design' (Springer-Verlag, 1985)
5 GARDARIN, G., and GELENBE, E.: 'New applications of data bases' (Academic Press, 1984)
6 ENCARNACAO, J.L., and LOCKEMANN, P.C.: 'Engineering databases' (Springer-Verlag, 1990)
7 WANG, P.C.C.: 'Advances in engineering data handling' (Kluwer Academic Publishers, 1984)
8 GEPPERT, L.: 'Technology 1995; solid state', *IEEE Spectr. (USA)*, January 1995, **32**, (1), pp. 35–39
9 LEE, E.K., and KATZ, R.H.: 'The performance of parity placements in disc arrays', *IEEE Trans. Comput. (USA)*, June 1993, **42**, (6), pp. 651–664
10 RUEMMLER, C., and WILKES, J.: 'An introduction to disc drive modeling', *IEEE Comp. Mag. (USA)*, March 1994, **27**, (3)