Multi-Partition RAID: A New Method for Improving Performance of Disk Arrays under Failure

WEN-JIIN TSAI AND SUH-YIN LEE

Institute of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, 300, Taiwan, ROC
Email: wjtsai@info1.csie.nctu.edu.tw

Disk arrays have been proposed as a way of improving I/O performance by using parallelism among multiple disks. This paper focuses, however, on improving the performance of disk array systems in the presence of disk failures, which are significant for applications where continuous operation is of concern. Although several approaches have been explored, the goals of achieving high performance and storage efficiency often conflict. In this paper, we propose a new variation of RAID organization, multi-partition RAID (mP-RAID), to improve storage efficiency and reduce performance degradation when disk failures occur. The idea is to recognize that frequently demanded data dominate the degree of the performance degradation when disk failures occur. mP-RAID subdivides a disk array into several partitions associated with different block organizations. Based upon data popularity, we assign data to appropriate partitions so that high performance and better storage efficiency can be achieved simultaneously.

Received August 4, 1995; revised May 23, 1997

1. INTRODUCTION

Disk arrays have been proposed as a way of improving I/O performance by using parallelism among multiple disks [1, 2]. For disk array systems, the performance when all disks are operational, in for example, a fault-free state, and the techniques for increasing the storage efficiency have been well studied [4–7]. However, the performance of disk arrays in the presence of failed disks, in for example, a failure state, should also receive attention so that lost data can be recovered without taking the system off-line. This is especially important for applications in which continuous operation is of concern [8]. The failure state of disk arrays can be further subdivided into degradation mode and recovery mode. The degradation mode is the condition when a failed disk in a RAID has not been replaced. In this mode, an on-the-fly reconstruction takes place for every access that requires data from the failed disk. The recovery mode is the condition when the recovering process is working to reconstruct the entire contents of a failed disk and store them on a replacement disk. Several strategies for reconstructing the data on failed disks were analysed in [9–11]. We focus this paper, however, on the performance of disk array systems in failure states, especially those in degradation modes that are significant for continuous-operation applications.

Fault-tolerance in storage subsystems is generally achieved either by replication or by parity-encoding [1, 2] for error correction. Patterson [12] presented five ways of organizing data on disks, popularized as Redundant Arrays of Inexpensive Disks (RAID). Several approaches for improving performance in failure state have also been explored. For replication-based systems, Copeland and Keller [13] presented a scheme called interleaved declustering, which balances the reconstruction load over all surviving disks so that performance is improved. There is also a variant called chained declustering that increases the data reliability of disk arrays [14]. Since replication-based systems consume a lot of storage capacity for redundancy, several researchers have focused on parity-based disk arrays. Unfortunately, standard parity-based systems suffer severe performance degradation in failure state due to increased reconstruction overhead [10]. Muntz and Lui [15] solved this problem by employing a declustered-parity RAID, which reduces the increased load on surviving disks, while increasing storage redundancy. For disk array systems, the goals of achieving high performance and ensuring storage efficiency often conflict in the failure state.

In order to balance performance and storage efficiency, HP AutoRAID systems [16] exploit a two-level storage hierarchy implemented inside a single disk-array controller. In the upper level of this storage hierarchy, RAID-1 (mirroring) is used to provide excellent performance for write-active data, and in the lower level, RAID-5 is used to provide the most cost-efficient storage for write-inactive or read-only data, at somewhat lower performance. To maintain the optimum balance, the system automatically

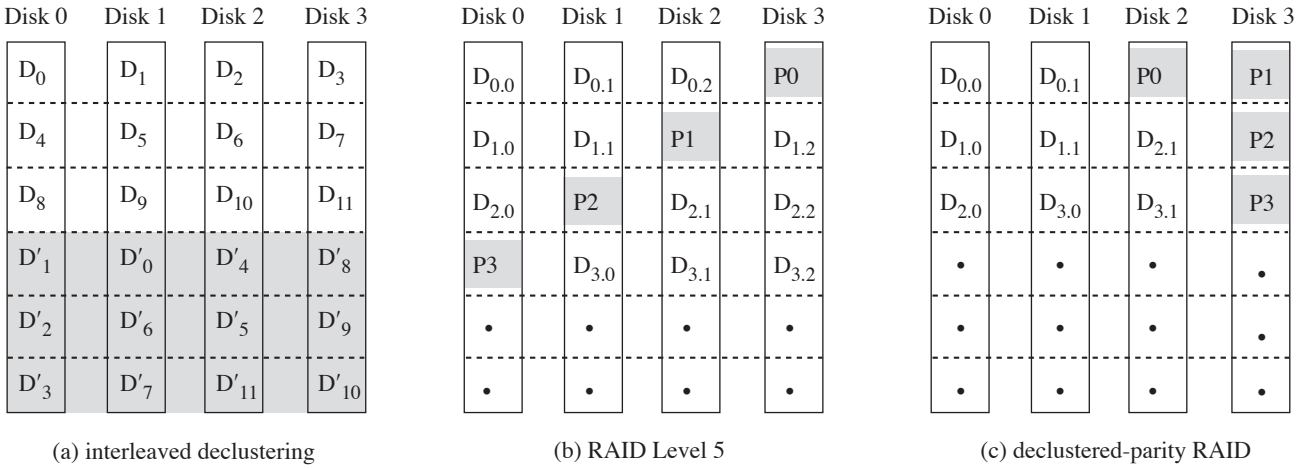


FIGURE 1. Various block organizations of disk arrays.

migrates data between these two levels as access patterns change.

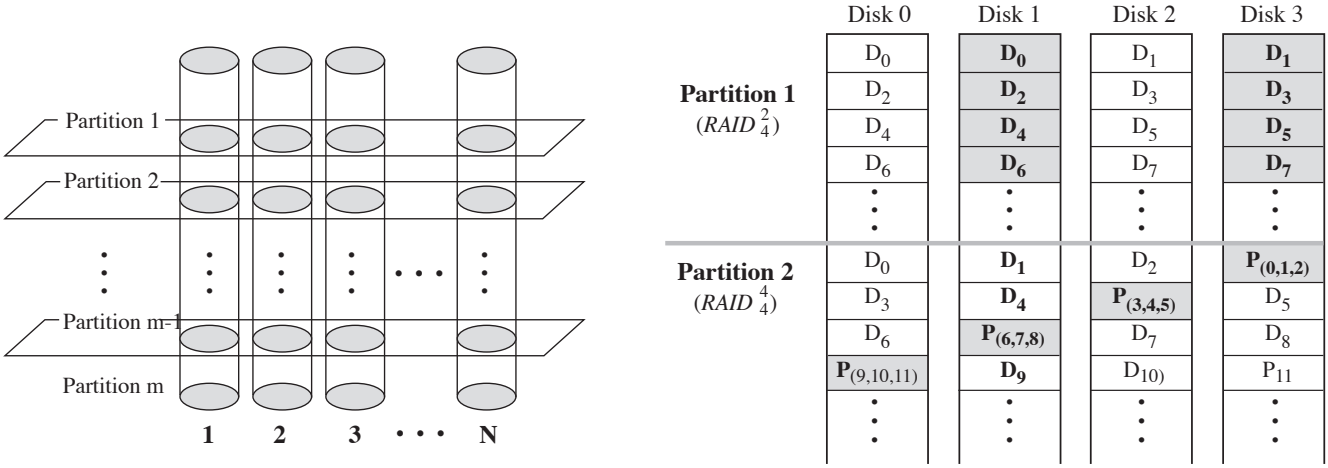
This paper extends the above idea by mixing declustered-parity RAIDs of different declustering ratio in a single disk-array controller to improve the performance in the failure state, where the declustering ratio indicates the fraction of the surviving disks that must be accessed during the reconstruction of a failed disk [15]. The idea is to recognize that the data reconstruction performance is dependent on the declustering ratio and that frequently demanded data dominate the degree of the performance degradation (due to reconstruction of the defective data). We subdivide a disk array into several partitions with different declustering ratio and then assign data to appropriate partitions based on data popularity. We call this multi-partition RAID, denoted by mP-RAID, inside which the storage hierarchy is not restricted to just two levels, and the block organizations used are not only restricted to RAID-1 and RAID-5. Note that RAID-1 and RAID-5 are just two cases of declustered RAIDs with extreme declustering ratios (the minimum and the maximum, respectively). By combining various declustering ratios within a single disk array, mP-RAID offers more configuration options, thus its performance/cost ratio for a wide variety of data-access patterns can more easily be optimized. Two approaches for configuring mP-RAID are also proposed in this paper, which are data-independent and popularity-based schemes. Our simulations show that mP-RAID does significantly increase storage efficiency and reduce performance degradation in the failure state.

The rest of this paper is organized as follows. Section 2 presents a closer look at the impact of disk failures on the performance of various RAIDs, including mirroring, RAID-5 and declustered RAID. Section 3 describes the system architecture and block organization of the proposed mP-RAID. Data-placement and partition-arrangement policies for optimizing mP-RAID's performance are also presented in this section. Section 4 describes two schemes for configuring mP-RAID by choosing appropriate parameters, the number of partitions, partition size, and declustering ratio. Analytical and simulation-based studies for mP-RAIDs are presented in Sections 5 and 6, respectively. Finally, conclusions are given in Section 7.

2. RELATED WORKS

Fault-tolerance in disk arrays is generally achieved either by replication or parity-encoding for error correction. Replication-based systems protect data by storing one or more duplicate copies of all data on separate disks (mirrored disks). To improve the performance in the failure state, interleaved declustering [13] allocates only half of each disk for primary data; the other half contains partial non-primary copies from each primary data on all other disks as depicted in Figure 1a, where D_i denotes the primary data and D_i' the non-primary copy. Note that, with interleaved declustering, the workload associated with reconstructing a failed disk will be distributed across all surviving disks in the array. Assuming that there is one disk failure in Figure 1a, for example, each surviving disk is imposed by only one-third of the load in the failed disk.

Parity-based systems incorporate redundancy by maintaining error-correction code or parity code computed over subsets of the data. The set of data units, over which a parity unit is computed, plus the parity unit itself, is called a parity group. When any single disk fails, its data can be reconstructed by reading data in the corresponding parity group, including the parity unit, and by computing the cumulative exclusive-or of this data. Figure 1b shows the block organization of parity-based redundancy, called RAID Level 5, where $D_{i.0}$, $D_{i.1}$ and $D_{i.2}$ denote the data blocks in parity group i protected by parity P_i. Unfortunately, disk failures cause large performance degradations in RAID Level 5 due to an increase in the loads imposed on all the surviving disks for every reconstruction of data in the failed disk. For example in Figure 1b, when disk 1 fails, the defective data block $D_{0.1}$ must be reconstructed by accessing $D_{0.0}$, $D_{0.2}$



(a) System architecture of mP-RAID

(b) Block organization in $2P\text{-}RAID_4^{(2,4)}$

FIGURE 2. The proposed multi-partition RAID.

and P_0. Thus, all surviving disks, 0, 2 and 3 are involved. Reconstruction of $D_{1.1}$ and $D_{3.0}$ is also carried out in a similar manner. Assuming that all the disks are equally loaded, a disk failure can double the load on each of the surviving disks. Note that this is only for the case in which the disk array is completely loaded with small reads (one block is read at a time) before the disk fails. When there are write requests or request sizes are not limited to one block at a time, the impact of a disk failure on the workload of a RAID-5 disk array is not as simple as just doubling that of the surviving disks as we have described. A better understanding of this impact can be found in [17].

Muntz and Lui [15] resolved this problem by introducing declustered-parity RAID, which regards a disk array as several RAIDs with smaller parity-group size. Due to these small parity-group sizes, the reconstruction overheads can be reduced. Figure 1c gives an example of declustered-parity RAID with an array size of four disks and a parity-group size of three. When disk 1 fails, the defective block $D_{0.1}$ can be reconstructed by accessing $D_{0.0}$ and P_0, i.e. only two (instead of three) disks are involved in each reconstruction. Assuming that all the disks in Figure 1c are equally loaded, a failed disk increases the workload on each of the surviving disks by only two-thirds. The penalty is a lower storage efficiency.

In the following presentation, we shall generally refer to $RAID_N^G$ $(N, G > 0)$ as a declustered-parity RAID with an array size of N disks and a parity-group size of G, except that $G = 2$ (i.e. $RAID_N^2$) denotes a replication-based disk array that can be RAID Level 1 (mirroring) or interleaved declustering as described above.

3. THE MULTI-PARTITION RAID

The defective data unit in a $RAID_N^G$ must be reconstructed by accessing the associated $G - 1$ surviving units in the array. That is, in a $RAID_N^N$ (RAID Level 5), all the surviving disks will be involved in each reconstruction (due to $G - 1 = N - 1$). The increased workload, which may significantly degrade performance, of course, can be reduced by choosing a smaller G. For example, in a $RAID_N^2$ (replication-based), the defective data can simply be reconstructed by reading its non-primary copy from one of the surviving disks $(G - 1 = 1)$. However, a $RAID_N^G$ with a small G, while yielding higher failure-state performance, suffers from increased storage consumption for redundant data. In $RAID_N^2$, 50% of the storage capacity is used for redundancy, but in $RAID_N^N$, only $1/N$ is required. This implies that the goals of achieving high storage efficiency and high failure-state performance often conflict. For a $RAID_N^G$, the trade-off between performance and storage efficiency is governed by the parity-group size G. Our hypothesis is that mP-RAID offers a better way to optimize these factors.

3.1. System architecture

In mP-RAID, we divide a disk array into several partitions associated with different parity-group sizes as depicted in Figure 2a. Let $mP\text{-}RAID_N^{(G_1,G_2,\ldots,G_m)}$ denote a multi-partition disk array, meaning that an array of N disks is a collection of m partitions with parity-group sizes equal to $G_1, G_2, \ldots,$ and G_m, respectively. The $mP\text{-}RAID_N^{(G_1,G_2,\ldots,G_m)}$ is, in fact, a collection of $RAID_N^{G_1}$, $RAID_N^{G_2}, \ldots, RAID_N^{G_m}$, except that they share the common N disks. For an $mP\text{-}RAID_N^{(G_1,G_2,\ldots,G_m)}$, the block organization in a partition with parity-group size equal to G_i will be arranged like that in a $RAID_N^{G_i}$. Figure 2b shows the block organization of a disk array, $2P\text{-}RAID_4^{(2,4)}$, which comprises two partitions, a $RAID_4^2$ (replication-based) partition at an upper address space and a $RAID_4^4$ (RAID Level 5) partition at a lower address space. Assuming that disk 2 fails, reading of the defective data units in partition $RAID_4^2$ (e.g. partition 1: D_1) can easily be achieved by accessing another disk (i.e. disk 3). However, reading of the defective data units in partition

$RAID_4^4$ (e.g. partition 2: D_2) must involve reconstruction of the data by reading from all the surviving disks (i.e. disks 0, 1 and 3).

Disk-array systems must have a function that maps data blocks to physical disk locations and identifies the parity layouts. Block mappings in mP-RAID are essentially different from those used in conventional disk arrays because different block organizations can exist simultaneously in mP-RAID (for partitions with different parity-group sizes). Fortunately, there are several algorithms for block mapping in a declustered-parity RAID of any parity-group size. Holland and Gibson [8] proposed a method, called block design, which maintains a block design table including every possible mapping of parity-group members to disks such that the parity layout on the disks can easily be identified using a simple mapping function. They also proposed an alternative called balanced incomplete block design for reducing the table size. The use of balanced incomplete block designs for constructing a declustered array were also proposed by Ng and Mattson [17] at almost the same time. In addition to block-design-based methods, Merchant and Yu [18] presented a fast algorithm for distributing parity groups over disks. Their algorithm is based on almost-random permutations generated by shuffling the identity permutation. All these algorithms have been shown to meet several criteria for a good parity layout: single failure correction, distributed parity, distributed reconstruction, efficient mapping, for example.

3.2. Data placement policy in mP-RAID

As we have mentioned, a $RAID_N^G$ with a small G, while achieving good performance, consumes more storage for redundancy. A $RAID_N^G$ with a large G has better storage efficiency, but incurs the penalty of significant performance degradation in the failure state. For a $RAID_N^G$, the trade-off between performance and storage efficiency is often governed by the parity-group size G. The idea behind mP-RAID is to make compromises among different Gs. Note that there is an exception that a large G can achieve better write performance if the writes are sufficiently large. In this case, the trade-off between performance and storage efficiency never exists because a large G is always the best solution. Thus, we focus this study on applications that are not large-write intensive.

In degradation mode, defective data blocks with higher access frequencies often cause more severe performance degradation due to higher reconstruction overheads. A $RAID_N^G$ of smaller G is preferred for these data blocks to improve performance. However, for data blocks with lower access frequencies, a $RAID_N^G$ of larger G may be preferred for storage efficiency. With consideration of data popularity in mP-RAID, we assign data to appropriate partitions and take advantage of different Gs to achieve higher performance and greater storage efficiency. Moreover, for systems in which every file is accessed as a whole each time, data units can be set in terms of files rather than data blocks because data blocks belonging to the same file must have the same access frequencies. Let $workload(f_i)$ denote the popularity of file f_i, defined by $workload(f_i) = size(f_i) \times access\ frequency(f_i)$. We assign files with higher workloads to partitions with smaller Gs (say hot partitions), and files with lower workloads to partitions with larger Gs (say cold partitions).

However, since data popularity may change, the data stored on mP-RAID must be adjusted accordingly. Hot data that becomes less popular must be moved to a cold partition; and data, possibly newly added data, which becomes more popular must be moved to a hot one. That is, the system must periodically migrate data between partitions to accommodate changes in data-access patterns. Data migration causes overhead due to data movement and redundancy reconstruction. Fortunately, data popularity in real systems often change slowly over time [19, 20]. In movie systems, for example, the migration period can be several weeks or months. The migration operations can thus be performed at times when the system can tolerate the overhead.

3.3. Partition arrangement

The performance of magnetic disks can be improved by placing frequently accessed data near the middle tracks of the disks to reduce the overhead that disk head movement entails. In this section, we use an organ-pipe arrangement for locating partitions in mP-RAID to achieve this goal. With this arrangement, the partition with the smallest G, which will contain the data with highest workload, is located on the middle tracks of the disk. Successively, the next two partitions with the next larger Gs are located adjacently on alternative sides. This process is continued with the remaining partitions of the mP-RAID. The ranks of Gs for M partitions placed on disks by organ-pipe arrangement can then be represented as the vector $(M-1, M-3, \ldots, 5, 3, 1, 0, 2, 4, 6, \ldots, M-2, M)$. Assuming that partitions are sorted in an ascending order of G, the mapping between partitions and disk locations is illustrated in Figure 3. Moreover, within each partition, we can place the more frequently accessed data on the side near the middle tracks to further improve the performance as indicated by the directions of the arrows shown in Figure 3. For systems where the data access probabilities are known in advance, and are identically and independently distributed, the organ-pipe arrangement which exhibits the minimal average access cost has been shown to be optimal for physical layouts on magnetic disks [21, 22].

3.4. Comparison with multiple independent disk arrays

An alternative to mP-RAID is to partition vertically an array of disks into several independent RAIDs (without sharing the same disks), each associated with different parity-group size G as depicted in Figure 4. Here we give the reasons why we use mP-RAID instead of multiple independent RAIDs as our platform:

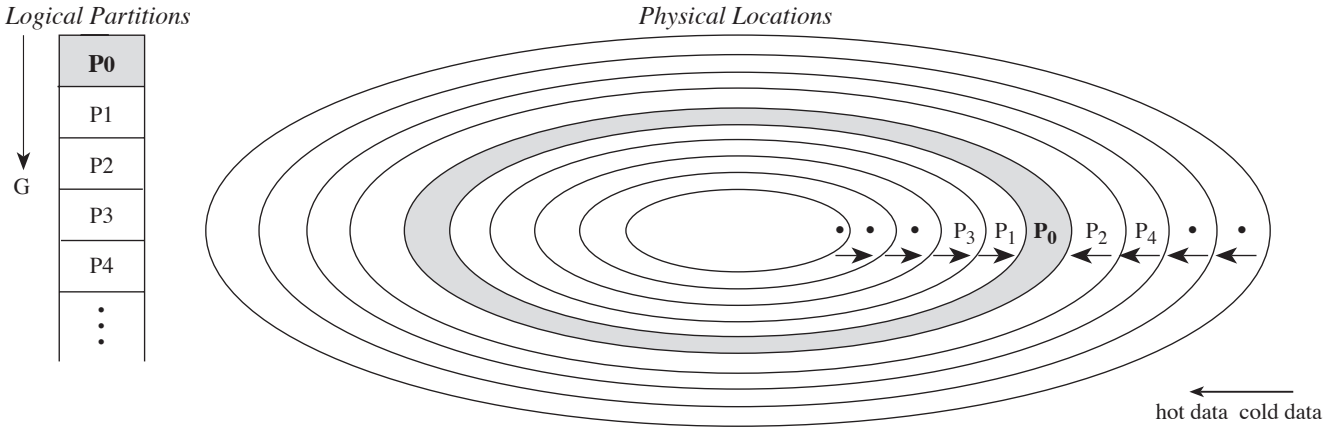


FIGURE 3. Partition arrangement by using the organ-pipe arrangement.

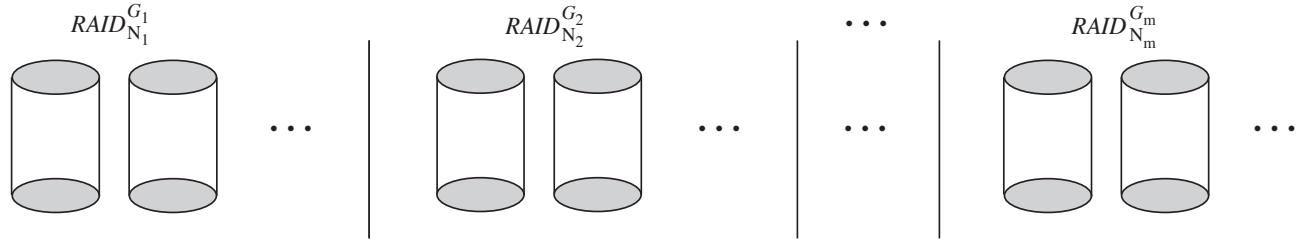


FIGURE 4. Multiple independent RAIDs.

1. Load balancing in fault-free state. In mP-RAID, since all the partitions (no matter what G is used) reside across all the disks, both high- and low-demand data are uniformly spread among disks. With multiple independent RAIDs, however, unbalanced loads result, with heavier loads imposed on the disks associated with RAIDs of small G.
2. Seek-time optimization. In mP-RAID, due to the partition arrangement, the high-demand data can be arranged near the middle tracks of all disks to reduce disk-head movements. With multiple independent RAIDs, such optimization may be less successful because high-demand data is concentrated on only one set of disks.
3. Load balancing in failure state. In mP-RAID, a failed disk causes all the partitions to be equally damaged and, therefore, all the surviving disks must share involvement in partial reconstruction. With multiple independent RAIDs, only the RAID containing the failed disk is involved in reconstruction, which may create a hot spot that significantly degrades system performance.
4. Deterministic performance degradation upon failure. With multiple independent RAIDs, a failed RAID with a larger G will result in less performance degradation, while a failed RAID with a smaller G will lead to larger degradation. Since we have no way of knowing which RAID will fail, the performance degradation during failure is unpredictable. In mP-RAID, any failed disk must contain both high- and low-demand data and, therefore, the increased load for reconstruction is deterministic and predictable.

However, there are some shortcomings that mP-RAID might suffer in comparison with multiple independent RAIDs.

1. With mP-RAID, it becomes impossible to grade existing disks according to usage, with high-demand data on smaller, faster, but presumably more expensive disks (e.g. SCSI disks) and low-demand data on larger, slower, but presumably cheaper disks (e.g. IDE disks). With multiple independent RAIDs, however, it is easy to do so.
2. Using mP-RAID, it is possible for distinct partitions to reside across different numbers of disks so that all the disks in the array need not have equal capacity (and need not be identical). However, this may add block-mapping complexity and may lose some advantages that mP-RAID has as described above. In order to take advantage fully of mP-RAID, all disks must be identical, so when the disks need to be upgraded, all disks must be replaced. With multiple independent RAIDs, however, only subsets of disks need to be replaced.

4. TWO MP-RAID CONFIGURATION STRATEGIES

Given a set of disks, there are several configurations for mP-RAID characterized by number of partitions, partition size and G-vector. We refer to the sequence of parity-group size used in mP-RAID as the G-vector. For example, the G-vector of $mP\text{-}RAID_N^{(G_1,G_2,\ldots,G_m)}$ is $(G_1, G_2, \ldots, G_m)$. Different configurations may lead to different performance and storage efficiency. In this section, we describe two schemes that might be employed in configuring an mP-RAID.

4.1. Data-independent scheme

With the data-independent scheme, the number of partitions, the partition size and the G-vector of an mP-RAID are determined by the system designer only, and are independent of data popularity. For example, we can simply use a two-partition RAID, $2P\text{-}RAID_N^{(2,N)}$, with equally sized partitions, and then assign the more popular data to the hot partition (i.e. the $RAID_N^2$ partition) and the less popular data to the cold partition (i.e. the $RAID_N^N$ partition). Using such a scheme, the system designer only needs to make sure that the multi-partition RAID is capable of accommodating all the data that will be stored on the array. Thus, for an $mP\text{-}RAID_N^{(G_1,G_2,\ldots,G_m)}$ configuration, the system designer must ensure that the following inequality is satisfied:

$$\sum_{i=1}^{m}[C_i \times (1 - 1/G_i)] \geq \text{total data size}, \qquad (1)$$

where $C_1, C_2, \ldots, C_m$ are partition sizes of m partitions and $\sum_{i=1}^{m} C_i$ is equal to the total capacity of the entire disk array. To satisfy the data-placement policy, we store data in an mP-RAID with a data-independent configuration based on the following steps:

1. Files are sorted in descending order of workload and are distributed across disks of the mP-RAID in a round-robin fashion.
2. Within each disk, the partition with smallest G, say P_1, is allocated for file placement first. When the available capacity of partition P_1 is exhausted, the next partition is then allocated, so that the partitions with the smallest G are allocated first. This process is continued until all the files are located.

4.2. Popularity-based scheme

The popularity-based scheme is based on the assumption that data popularity is known *a priori*. Given a set of files that will be stored on the disk array, the mP-RAID configuration is determined by clustering these files according to their workloads so that files with similar popularity are assigned to the same set. Set clustering can be accomplished using the following process. Assume that the files are pre-sorted in descending order of workload, such that $workload(f_i) > workload(f_j)$ for $i < j$. We start with f_1, forming a single clustered set τ_1 and setting the minimum workload of τ_1, say min_{τ_1}, to $workload(f_1)$ and the working set to τ_1. Then we check to see whether the next file f_2 belongs to τ_1. If $|workload(f_2) - min_{\tau_1}| < \Phi$ then f_2 is merged into τ_1 and min_{τ_1} is modified to $workload(f_2)$. Otherwise, f_2 forms a new set τ_2 with min_{τ_2} being equal to $workload(f_2)$ and the working set being set to τ_2. The Φ above is used as a threshold for clustering files among different sets. In this way, we check to see whether each file, f_i, belongs to the working set or must form a new set. The process is continued with all the remaining files. The pseudo-code for our clustering algorithm is given in the Appendix. Using the popularity-based scheme, we configure the mP-RAID with the number of partitions being equal to the number of clustered file sets, and the partition capacity being proportional to the size of file sets. Let $size(\tau_i)$ denote the total size of the data in set τ_i. We use the following equation along with Equation (1) to determine partition size.

$$\begin{aligned} C_1(1 - 1/G_1) \;:\; C_2(1 - 1/G_2) : \ldots : C_m(1 - 1/G_m) \\ = size(\tau_1) : size(\tau_2) : \ldots : size(\tau_m), \end{aligned} \qquad (2)$$

where C_i denotes the size of partition i and $(G_1, G_2, \ldots, G_m)$ the G-vector. Note that in Equation (2), the partition size is determined by the size of the clustered file set along with the G-vector. For an array of N disks and m partitions, $(N-1)^m$ possible G-vectors may result from the potential combinations of parity-group sizes (say Gs), where $(N-1)$ is the number of candidates for G, ranging from 2 to N. Among the $(N-1)^m$ different G-vectors, the one that leads to the highest effective throughput and storage efficiency is selected. Both effective throughput and storage efficiency of a disk array are defined in Section 5. In summary, we select the popularity-based configuration as follows:

1. Compute the workload of each file and sort files in a descending order of workload.
2. Cluster files into sets using a clustering algorithm and determine the number of partitions.
3. Generate C_m^{N-1} possible G-vectors, representing the possible assignments of m partitions to $N-1$ objects. For each G-vector, determine the partition size by using Equation (2) together with Equation (1).
4. Among the C_m^{N-1} configurations, select the best one based on the evaluation model.

To satisfy this data placement policy, we store data in an mP-RAID based on the following two steps.

1. All the files are sorted in a descending order of workloads and are distributed across all the disks in a round-robin fashion.
2. For each disk, the files in the same clustered file set are located in the same partition. The clustered file sets with higher workloads are assigned to partitions with smaller G and those with lower workloads are assigned to partitions with larger G.

TABLE 1. The effective throughput of various RAIDs in degradation mode [$k = 1$ for $G_i > 2$ (parity-based partition) and $k = 0$ for $G_i = 2$ (replication-based partition)].

RAID types	Small $read_f$	Small $write_f$	Large $read_f$	Large $write_f$
$RAID_N^G$	$\frac{1}{G-1}$	$\frac{1}{(G-1)\times k+1}$	$\frac{G-1}{G-1}=1$	$\frac{G-1}{G-1}=1$
$mP\text{-}RAID_N^{G_1,\ldots,G_m}$	$\sum_{i=1}^{m}\frac{WR_i}{G_i-1}$	$\sum_{i=1}^{m}\frac{WR_i}{(G_i-1)\times k+1}$	$\sum_{i=1}^{m}WR_i=1$	$\sum_{i=1}^{m}WR_i=1$

5. PERFORMANCE DEGRADATION ANALYSIS

The idea behind mP-RAID is to maintain high performance during a failure state, especially when the system is affected by a failed disk that has not yet been replaced, i.e. is in degradation mode. To show how effective this is, an analytical model similar to that used by Patterson *et al.* [12] for a fault-free state is presented here. We first summarize the notation used.

W_i: the workload on partition i in terms of number of block units per second, which is defined by $W_i = \sum workload(file_j), \forall file_j \in partition_i$. WR_i: the workload ratio of partition i, which is defined by the ratio of W_i to total workload. C_i: the capacity of partition i, in terms of the number of block units. CR_i: the capacity ratio of partition i, which is defined by the ratio of C_i to the total capacity.

5.1. The analytical model

5.1.1. *Effective throughput*

Let the term logical request refer to a user request after it has been mapped to the disk array. A user request that involves all data in a stripe (i.e. one that involves G logical requests) is called a large request. A user request that requires data on only one disk (i.e. one that consists of one logical request) is called a small request. Let the term physical request denote physical disk accesses invoked by logical requests. Due to redundancy in the disk array, a logical request often involves multiple physical requests. For example, a small write that consists of one logical write, will invoke two physical reads and two physical writes. Effective throughput, a measure of RAID performance, is defined by the ratio of logical requests to physical requests. In this paper, we focus on the effective throughput in degradation mode. Let a (large/small) $request_f$ refer to a user request that involves data located on the failed disk. In degradation mode, a (large/small) $request_f$ often involves several physical requests on surviving disks due to reconstruction of defective data. Table 1 shows the effective throughputs for $RAID_N^G$ in degradation mode. Since a small $write_f$ on a $RAID_N^G$ with $G = 2$ only needs to access the non-primary copy, no reconstruction is required, so for that case the parameter k in Table 1 is equal to 0. The effective throughput for an mP-RAID in degradation mode can also be easily calculated. For example, a $2P\text{-}RAID_N^{(2,N)}$ with p per cent of its workload falling in the $RAID_N^2$ partition, and q per cent in the $RAID_N^N$ partition (where $p+q=1$), will have an effective throughput for small $read_f$ equal to $p\times 1+q\times\frac{1}{N-1}$, and one for small $write_f$ equal to $p\times 1+q\times\frac{1}{N}$. Table 1 also shows the effective throughput of an $mP\text{-}RAID_N^{(G_1,G_2,\ldots,G_m)}$, where WR_i is the workload ratio for partition i.

Note that the above equations for 'large' reads and writes are only for the case when a request of size equal to a stripe also happens to be aligned with a stripe boundary. In fact, only a small fraction of stripe-size requests exactly fit this definition of 'large request'. For $RAID_N^G$, let us consider a large read that accesses $G-1$ blocks across two stripes: e blocks ($e < G-1$) in one stripe that contains a failed disk and the other $G-1-e$ blocks in the other stripe. The effective throughput for this request is given by $\frac{1}{2}\left(\frac{e}{G-1}+1\right)$, which, being less than 1, is less than the effective throughput of aligned large reads given in Table 1. That is, large reads that are not perfectly aligned will have lower effective throughputs. Similar results also hold for $mP\text{-}RAID_N^{(G_1,G_2,\ldots,G_m)}$ and for large writes.

5.1.2. *Storage efficiency*

The storage efficiency, a measure of redundancy capacity cost, is defined as the effective (user) data capacity divided by the total disk capacity. For a disk array, $RAID_N^G$, the storage efficiency equals $1-1/G$ because a fraction $1/G$ of the capacity is used for redundancy. For an mP-RAID, $mP\text{-}RAID_N^{(G_1,G_2,\ldots,G_m)}$, the storage efficiency is then given by $\sum_{i=1}^{m}(CR_i\times(1-1/G_i))$, where CR_i is the capacity ratio for partition i. For example, a $2P\text{-}RAID_N^{(2,N)}$, for which p per cent of capacity is used for the $RAID_N^2$ partition and q per cent for the $RAID_N^N$ partition (where $p+q=1$), has a storage efficiency equal to $p\times\frac{1}{2}+q\times\frac{N-1}{N}$.

5.2. Analytical evaluation

To simplify evaluation of the analytical model, we assumed that data popularity was known *a priori* and remained unchanged over some period, although we have described how to eliminate such a constraint by migrating data between partitions. Five different levels of demand skew, 90–10, 80–20, 70–30, 60–40 and 50–50 (uniform distribution) were used in our model, where 'X–Y' indicates X% of the disk accesses are to Y% of the data in the storage subsystem.

Muntz and Lui [15] have defined the ratio $(G-1)/(N-1)$ as α, where N is the number of disks and G is the parity-group size for a disk array. This parameter α, which is called the declustering ratio, indicates the fraction of each surviving disk that must be accessed during the

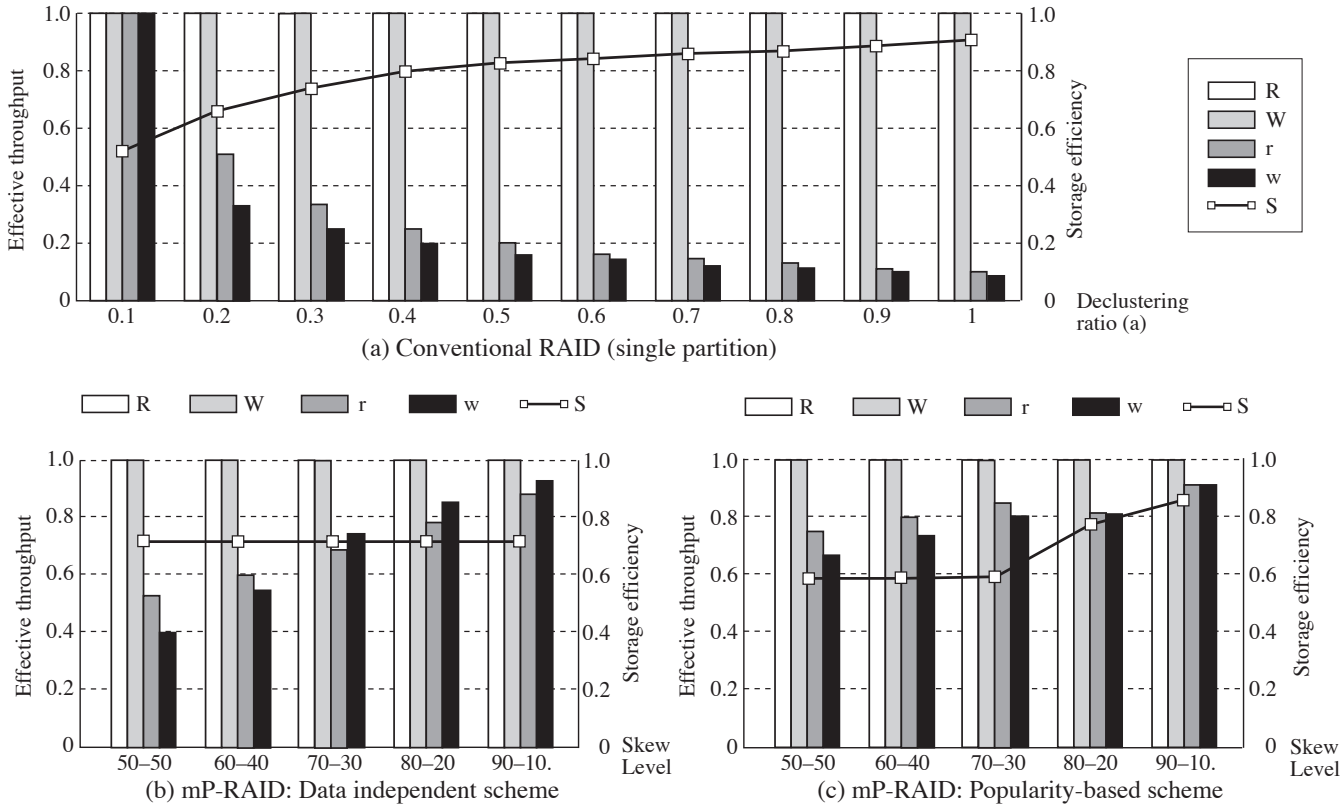


FIGURE 5. Comparison between various RAID organizations.

reconstruction of a failed disk. In the following, the analysis of storage efficiency and effective throughput of RAIDs in degradation mode was parametrized by α. For the mP-RAID, we performed analysis on both data-independent and popularity-based configurations. In the data-independent configuration, we used a $2P\text{-}RAID_{11}^{(2,11)}$ with equal partition sizes as our platform. Since the capacity ratio of partition $RAID_{11}^{2}$ to $RAID_{11}^{11}$ is around 1:2 (due to $1-1/2$:1-1/11$\approx$1:2), we assigned one-third of the data (with high workload) to partition $RAID_{11}^{2}$ and the other two-thirds of the data to partition $RAID_{11}^{11}$ for each of the five demand-skew levels. In the popularity-based configuration, we clustered the data according to the demand-skew levels. For example, on the 90–10 level, 10% of the data, which comprises 90% of the workload, was clustered in one set and 90% of the data, which comprises 10% of the workload, was clustered in the other set. Again, $G = 2$ for the hot partition and $G = 11$ for the cold partition.

Figure 5 shows a comparison between various RAID organizations in terms of effective throughput and storage efficiency in degradation mode, where labels **R**, **W**, **r**, **w** and **S** denote large $read_f$, large $write_f$, small $read_f$, small $write_f$ and storage efficiency, respectively. Figure 5a shows the effective throughput and the storage efficiency of the conventional RAID as a function of α, while Figure 5b and c show those of the mP-RAID for the two distinct configurations, data-independent and popularity-based, parametrized by five demand-skew levels. We used 11 disks for the analysis so that $\alpha = 0.1$ for a replication-based RAID organization (where $G = 2$) and $\alpha = 1$ for RAID Level 5 (where $G = 11$).

Figure 5a shows that while the effective throughputs of **r** and **w** improved as α decreased, the storage efficiency **S** degraded significantly. In the case of $\alpha = 0.1$, the storage efficiency **S** is only 0.5, meaning that half of the total storage capacity was used for redundancy, and the results are essentially independent of demand skew. However, in Figure 5b and c, the effective throughputs of **r** and **w** improved as demand-skew level increased, without degrading the storage efficiency **S**. For the data-independent mP-RAID shown in Figure 5b, the storage efficiency **S** was fixed at 0.7 and the effective throughputs of **r** and **w** varied from 0.4 to 0.93 for different demand-skew levels. In contrast, for the conventional RAID shown in Figure 5a the effective throughputs of **r** and **w** were only 0.5 and 0.37 for storage efficiency $\mathbf{S} = 0.7$ (i.e. when $\alpha = 0.2$). The popularity-based mP-RAID shown in Figure 5c also had a significant performance advantage over the conventional RAID because storage efficiency **S** and effective throughputs **r** and **w** approximated 0.9 at skew-level 90–10. The results in Figure 5 show that mP-RAID achieved a better performance(throughput)/cost(storage) ratio than a conventional RAID.

TABLE 2. Simulation parameters. G = parity-group size, CR = the ratio of partition capacity to total capacity.

(a) Disk parameters	
Cylinders per disk	2667
Tracks per cylinder	21
Sectors per track	99
Bytes per sector	512
Revolution time (ms)	11.1
Seek time (ms)	11 (avg), 1.7 (min), 22.5 (max)

(b) RAID parameters	
Number of disks	10
Stripe unit	One track (49.5 kbyte)
Parity layout	Block design based
Parity overhead	17%
Parity updating	Read–modify–write

(c) Partition parameters

RAIDs	Partition 1	Partition 2	Partition 3	Parity overhead(%)
SP	$G = 6, CR = 1.0$			$1/6 = 16.67$
2P	$G = 2, CR = 0.07$	$G = 7, CR = 0.93$		$1/2 \times 0.07 + 1/7 \times 0.93 = 16.78$
2POP	$G = 7, CR = 0.93$	$G = 2, CR = 0.07$		$1/7 \times 0.93 + 1/2 \times 0.07 = 16.78$
3P	$G = 2, CR = 0.02$	$G = 3, CR = 0.10$	$G = 7, CR = 0.88$	$1/2 \times 0.02 + 1/3 \times 0.10 + 1/7 \times 0.88 = 16.9$
3POP	$G = 3, CR = 0.10$	$G = 2, CR = 0.02$	$G = 7, CR = 0.88$	$1/3 \times 0.10 + 1/2 \times 0.02 + 1/7 \times 0.88 = 16.9$

6. SIMULATION

This section presents a simulation-based comparison of five different RAIDs: a single-partition RAID (SP), a two-partition RAID (2P), a two-partition RAID with organ-pipe (2POP), a three-partition RAID (3P) and a three-partition RAID with organ-pipe (3POP). Throughput (Mbyte/s) is used as our main performance metric.

6.1. The simulation model

6.1.1. The workload model

In order to model workloads according to demand skew, we used a Zipf-like probability distribution function, $Z(b_i)$ [23], as follows:

$$Z(b_I) = c/i^{(1-\theta)},$$

where

$$c = 1 \Big/ \sum_{i=1}^{B} (1/i^{(1-\theta)}) \qquad 1 \le i \le B,$$

with B being the number of blocks accessed, and θ being derived from $\theta = \log(\text{fraction of data accesses})/\log(\text{fraction of data blocks})$. For example, the θ value for the '80–20' degree of demand skew (in which 80% of the data accesses were to 20% of the blocks in the storage system) was $\theta = \log 0.8/\log 0.2 = 0.1386$. As θ varies from 1 to 0, the probabilities vary from a uniform distribution to the pure Zipf distribution. Our simulations used five θ values, 0.0458, 0.1386, 0.2962, 0.5575 and 1.0, representing the demand-skew levels, 90–10, 80–20, 70–30, 60–40 and 50–50 (uniform distribution). In addition, the workloads in our simulations were also characterized according to request size, request type (read/write) and read/write ratio.

6.1.2. The disk model

Our simulations were based on the use of Seagate Elite3 ST-43400N 3.5" SCSI disk drives. The relevant parameters are summarized in Table 2a. We modelled the seek profile as $seekTime(x) = a(x-1)^{1/2} + b(x-1) + c$, where x was the seek distance in cylinders, and a, b and c were constants chosen to satisfy the single-cylinder seek time, max-stroke seek time and average seek time. For the disk we modelled, $a = 231$, $b = 0.003$ and $c = 1.7$. The square root term in the above equation models the constant acceleration/deceleration period of the disk head and the linear term models the period after maximum disk head velocity is reached. Chen and Lee [24] have shown that such a model closely approximates the seek profile of an actual disk.

6.1.3. The RAID model

The block design, proposed by Holland and Gibson [8], was used for locating parity stripes on SP as well as on all the partitions in 2P, 2POP, 3P and 3POP, because it meets several criteria for a good parity layout as we have described. The only exception was that we implemented the partitions with $G = 2$ as replication-based partitions. Table 2b summarizes the relevant parameters of the disk arrays we modelled in this study. Moreover, for fair comparison, all the RAIDs were simulated with equal amounts of parity overhead. In this study, the parity overhead was set at 17% of total storage;

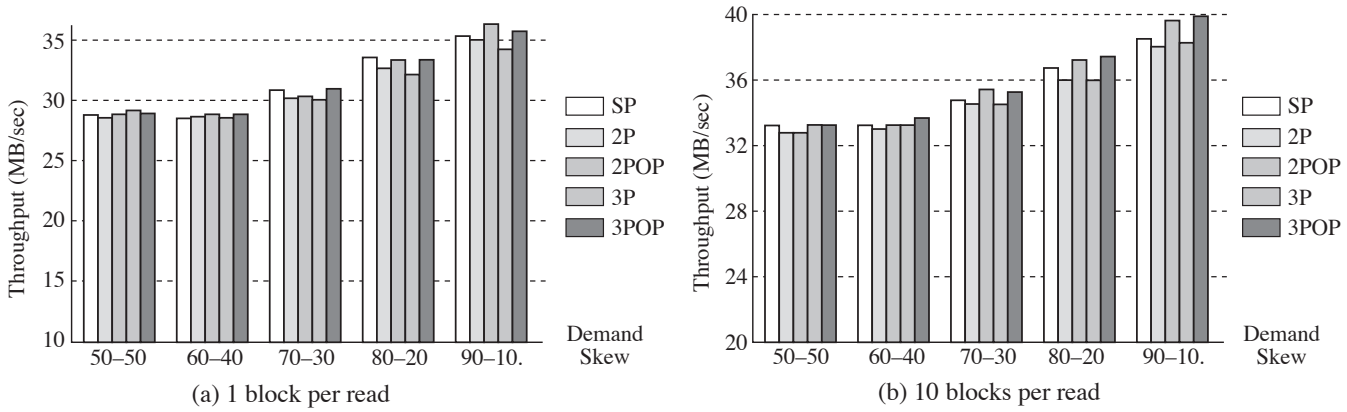


FIGURE 6. Read performance (fault-free state).

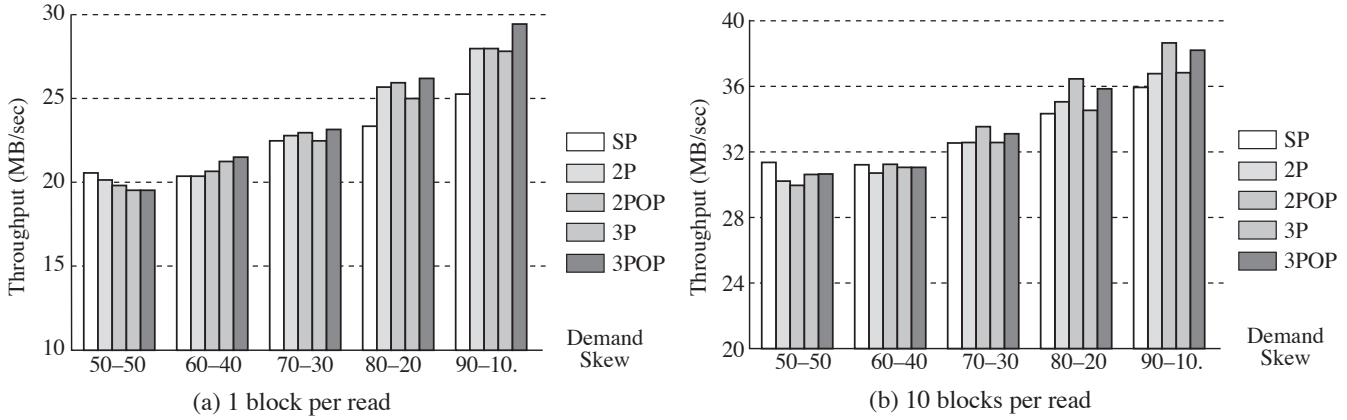


FIGURE 7. Read performance (degradation mode).

the parameters for all the RAIDs are given in Table 2c.

6.1.4. The RAID simulator

The raidSim software developed at UC Berkeley is an event-driven simulator for modelling disk arrays. It consists of a workload module for generating synthetic I/O requests, an array module for implementing a variety of RAID levels, and a disk module for modelling disk behaviour. In this study, we extended the workload module and the array module in raidSim with no modification to the disk module. The workload module was modified to accommodate workloads with demand skews based on the Zipf-like distribution function, and the array module was modified to support block-design-based parity layouts and handling partition information.

6.2. Simulation results and discussion

For each workload, both the fault-free state and degradation mode were simulated. Figures 6 and 7 show the throughputs of the five RAIDs for the read workload (100% reads), Figures 8 and 9 for the write workload (100% writes) and Figure 10 for a mixed workload (50% reads and 50% writes).

6.2.1. Read workloads

We observed that mP-RAID yields minimal difference in read performance relative to SP in the fault-free state because all the disk arrays achieved almost the same throughputs, as shown in Figure 6. However, Figure 7 shows that, in the degradation mode, the four mP-RAIDs (2P, 2POP, 3P and 3POP) had significant performance advantages over SP for small reads at high demand-skew levels because the four mP-RAIDs incorporated partitions with smaller Gs for placement of frequently accessed data. For these partitions, reads from the failed disk imposed lower overheads on surviving disks and hence led to higher throughputs. Moreover, although Table 1 implies no performance gains for degraded mP-RAIDs for large reads, Figure 7b shows a small improvement in performance for mP-RAIDs relative to SP at high skew levels. The reason

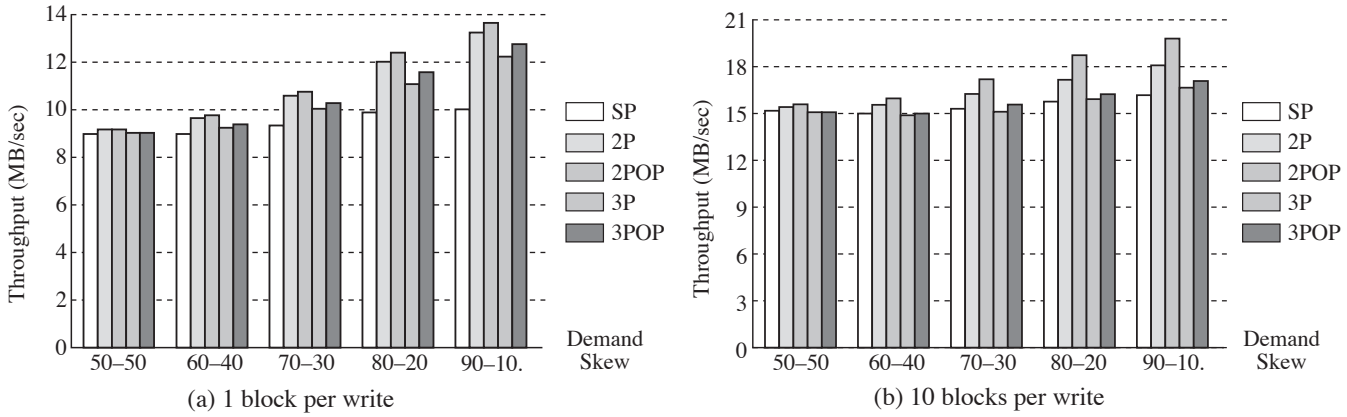


FIGURE 8. Write performance (fault-free state).

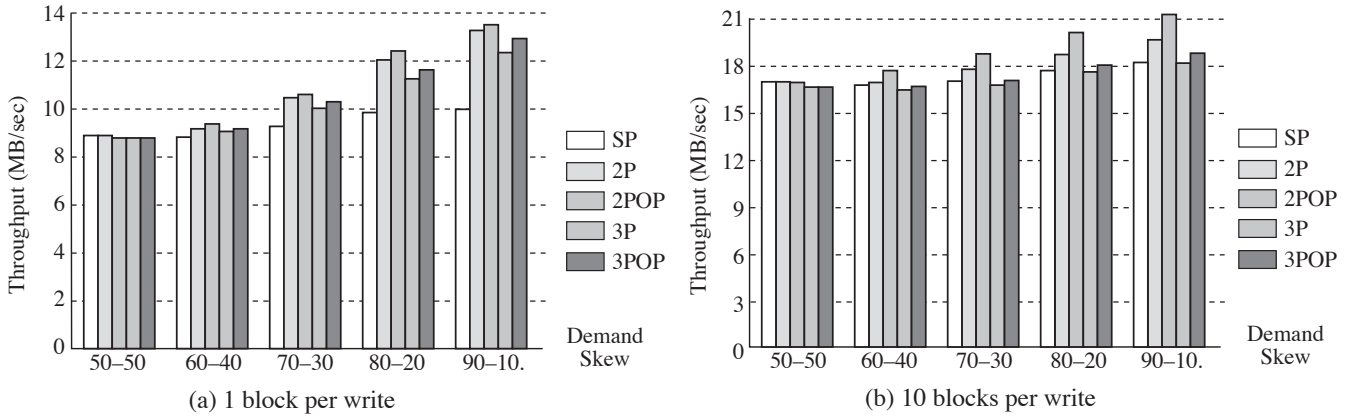


FIGURE 9. Write performance (degradation mode).

is that large reads will be broken into several smaller reads when the requested blocks reside across multiple stripes. These results demonstrate the superiority of mP-RAIDs for read workloads in the degradation mode.

6.2.2. Write workloads

It is observed that the write throughputs shown in Figure 8 were uniformly lower than the read throughputs shown in Figure 6 because writes entail extra overheads to maintain redundancy. A fault-free small write to parity-based partitions must execute four, instead of one, separate disk accesses for read–modify–write parity updating. This so-called small write problem often degrades the performance of RAIDs dramatically. However, Figure 8a shows that mP-RAIDs (2P, 2POP, 3P and 3POP) obtained better fault-free small write throughputs at high demand-skew levels as compared with SP because there was a replication-based partition on each of the four mP-RAIDs. For these partitions, a small write only needs to execute two, rather than four, disk accesses for redundancy updating, and hence, the better performance. This also explains why 2P-RAIDs (i.e. 2P and 2POP) performed better than 3P-RAIDs (i.e. 3P and 3POP) over most parameter ranges, since the replication-based partitions in 2P-RAIDs were much larger than those in 3P-RAIDs, as shown in Table 2c. Figure 9 shows that the four mP-RAIDs also had better degradation-mode performance than SP for reasons similar to those given for read workloads. Note that although several techniques have been proposed for solving the small-write problem in the fault-free state [25, 6], the results in Figure 9 indicate that mP-RAID not only relieves the small-write problem in the fault-free state, but also significantly improves write performance in degradation mode.

6.2.3. Mixed workloads

Figure 10 shows the throughputs of disk arrays for 50% reads and 50% writes. The request sizes were randomly set from 1 to 10 blocks. Large sequential requests were not used because the sequentiality is much less meaningful when more than one request is generated to the same array

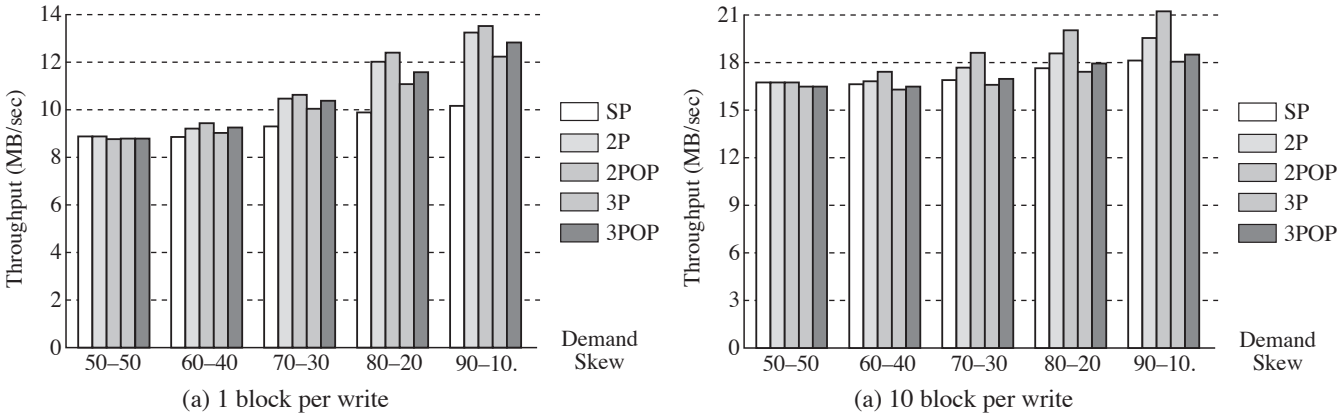


FIGURE 10. Performance for 50% reads and 50% writes.

(the RAIDs will thrash between these sequential request streams). Figure 10 indicates that, no matter whether in fault-free state or in degradation mode, mP-RAIDs performed better than SP over most ranges of demand-skew levels.

6.2.4. Remarks

The simulation results show that mP-RAID exhibits many benefits, especially for applications with demand skew. Fortunately, several real systems are characterized by highly skewed data-access patterns. For movie systems, there are empirical results showing that accesses to movies are often non-uniform [26] and can be approximated by a Zipf distribution with parameter 0.271, i.e. between skew level 70–30 and 80–20 [27]. For text-based database systems with indexed files based on terms or keywords, the distribution of term accesses has been shown to approximately follow a Zipf law with a '80–20' rule [28]. According to the simulation results, mP-RAIDs performed better than conventional RAIDs for most cases with skew level 80–20. This implies that mP-RAID does have an advantage for some real-world applications.

Moreover, since all the RAIDs were simulated with equal amounts of redundancy as we have shown in Table 2c, we can conclude that not only does mP-RAID achieve a better performance (throughput)/cost (storage) ratio than conventional RAIDs, but also that the performance gains provided by mP-RAID do not come at the cost of consuming more storage. We also found that 2POP and 3POP obtained better performance than 2P and 3P over most parameter ranges. This means that, by using organ-pipe assignment for locating partitions, the performance of mP-RAID can be further improved.

7. CONCLUSIONS

In this paper, we proposed a new variety of disk arrays mP-RAID to improve the performance in the presence of disk failures. In mP-RAID, we divide a disk array into several partitions with different block organizations (and thus different performance) and then assign data to appropriate partitions based on the data popularity. Two schemes were proposed to configure the mP-RAID, which are the data-independent scheme and the popularity-based scheme. In order to demonstrate the superiority of mP-RAID, both analytical and simulation-based experiments have been conducted. The results show that mP-RAID is more tolerant of disk failure than conventional RAIDs because performance degradation during failure was reduced. Moreover, gains can also be achieved in the fault-free state for write workloads. In conclusion, we believe that mP-RAID does show promise for improving the performance of disk arrays in failure states without incurring extra storage cost.

The work in this paper is based upon the assumption that data popularity changes slowly over time and is known in advance, and thus appropriate data migrations between partitions can be performed periodically. However, for applications where data popularity and changes over time are not predictable, a working set model would probably be more suitable so that data migration could be performed dynamically in the background as the access patterns change. More work needs to be done in this area and is the subject of our future research.

ACKNOWLEDGEMENTS

We thank the anonymous referees for their valuable comments and suggestions. This work was partially sponsored by the National Science Council, Taiwan, ROC under the Contract No. NSC85-2622-E009-006R.

REFERENCES

[1] Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H. and Patterson, D. A. (1994) RAID: high-performance, reliable secondary storage. *ACM Comput. Surv.*, **26**, 145–185.

[2] Ganger, G. R., Worthington, B. L., Hou, R. Y. and Patt, Y. N. (1994) Disk arrays: high-performance, high-reliability

storage subsystem. *IEEE Comput.*, **27**, 30–37.

[3] Chen, P. M., Gibson, G., Katz, R. and Patterson, D. A. (1990) An evaluation of redundant arrays of disks using an Amdahl 5890. In *Proc. of the 1990 ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems*, Colorado, pp. 74–85.

[4] Chen, P. M. and Patterson, D. A. (1990) Maximizing performance in a striped disk array. In *Int. Symp. on Computer Architecture*, pp. 322–331. IEEE, New York.

[5] Rosenblum, M. and Ousterhout, J. K. (1991) The design and implementation of a log-structured file system. In *Proc. of the 13th ACM Symp. on Operating Syst. Principles*, pp. 1–15. ACM, New York

[6] Menon, J. and Kasson, J. (1992) Methods for improved update performance of disk arrays. In *Proc. of the Hawaii Int. Conference on System Sciences*, pp. 74–83.

[7] Stodolsky, D. and Gibson, G. A. (1993) Parity logging: overcoming the small write problem in redundant disk arrays. In *Proc. of the 1993 Int. Symposium on Computer Architecture*, San Diego, CA.

[8] Holland, M. and Gibson, G. (1992) Parity declustering for continuous operation in redundant disk arrays. In *Proc. of the 5th Int. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)*, pp. 23–35. New York, IEEE.

[9] Menon, J. and Mattson, R. (1992) Comparison of sparing alternatives for disk arrays. In *Proc. of Int. Symposium on Computer Architecture*, Gold Coast, Australia. IEEE.

[10] Chandy, J. and Reddy, A. L. N. (1993) Failure evaluation of disk array organization. In *Proc. of the Int. Conf. on Distributed Computing Systems*, pp. 319–326. Los Alamitos, CA, IEEE Computer Society.

[11] Holland, M., Gibson, G. and Siewiorek, D. (1993) Fast, on-line failure recovery in redundant disk arrays. In *Proc. of the 23th Int. Symposium on Fault Tolerant Computing*, pp. 422–431. IEEE, Washington, DC.

[12] Patterson, D. A., Gibson, G. and Katz, R. H. (1988) A case for redundant arrays of inexpensive disks (RAID). In *Proc. of the Int. Conf. on Management of Data (SIGMOD)*, pp. 109–116. ACM, New York.

[13] Copeland, G. and Keller, T. (1989) A comparison of high-availability media recovery techniques. In *Proc. of the ACM SIGMOD Conference*, pp. 98–109. ACM, New York.

[14] Hsiao, H. I. and DeWitt, D. (1990) Chained declustering: a new availability strategy for multiprocessor database machines. In *Proc. of the 6th Int. Data Engng. Conf.*, pp. 456–465. IEEE, New York.

[15] Muntz, R. R. and Lui, C. S. (1990) Performance analysis of disk arrays under failure. In *Proc. of the 16th Conference on Very Large Databases*, pp. 162–173.

[16] Wilkes, J., Golding, R., Staelin, C. and Sullivan, T. (1996) The HP AutoRAID hierarchical storage system, *ACM Trans. Comput. Syst.*, **14**, 108–136.

[17] Ng, S. and Mattson, R. (1992) Maintaining good performance in disk arrays during failure via uniform parity group distribution. In *Proc. of the 1st Int. Symposium on High Performance Distributed Computing*, pp. 260–269. IEEE, New York.

[18] Merchant, A. and Yu, P. S. (1992) Design and modelling of clustered RAID. In *Proc. of the 22th Int. Symposium on Fault-Tolerant Computing*, pp. 140–149. IEEE, Washington.

[19] Floyd, R. A. and Schlatter, E. C. (1989) Directory reference patterns in hierarchical file systems. *IEEE Trans. Knowl. Data Eng.*, **1**, 238–247.

[20] Deshpande, M. B. and Bunt, R. B. (1988) Dynamic file management techniques. In *Proc. of the 7th IEEE Phoenix Conf. on Computers and Commun.*, pp. 86–92. IEEE, New York.

[21] Grossman, D. D. and Silverman, H. F. (1973) Placement of records on secondary storage device to minimize access time. *J. ACM*, **20**, 429–438.

[22] Yue, P. C. and Wong, C. K. (1973) On the optimality of probability of ranking scheme in storage applications, *J. ACM*, **20**, 624–633.

[23] Knuth, D. E. (1973) *The Art of Computer Programming*, 3rd edn. Addison-Wesley, Reading, MA.

[24] Chen, P. M. and Lee, E. K. (1995) Striping in a RAID level 5 disk array. In *Proc. of the Int. Conf. on Measurement and Modelling of Comp. Sys.*, pp. 136–145. ACM, New York.

[25] Carson, S. and Setia, S. (1992) Optimal write batch size in log-structured file systems. In *USENIX Workshop on File Systems*, pp. 79–91. USENIX Assoc., Berkeley, CA.

[26] Video Store Magazine (1992), December.

[27] Dan, A., Sitaram, D. and Shahabuddin, P. (1994) Scheduling policities for an on-demand video server with batching. In *Proc. of the ACM Multimedia '94*, San Francisco, CA, pp. 15–23.

[28] Salton, G. and McGill, M. (1983) *Introduction to Modern Information Retrieval*. McGraw-Hill, New York.

APPENDIX

The clustering algorithm

Input: A set of files $(f_1, f_2, \ldots, f_n)$ sorted in a descending order of workloads.

Output: A sequence of clustered file sets $(\tau_1, \tau_2, \ldots, \tau_m)$.

Begin

$k := 1, \tau_1 := \{f_1\}$ /* k is the index of the working set (the last set clustered) */

$min_{\tau_1} := workload(f_1)$

for $i := 2$ to n {

if $(|min_{\tau k} - workload(f_i) < \Phi|)$ {

/* Current file f_i is merged into the working set τ_k */

$\tau_k := \tau_k \cup \{f_i\}$

$min_{\tau k} = workload(f_i)$

} **else** {

/* A new set is created only containing the current file f_i */

$k := k + 1$

$\tau_k := \{f_i\}$

$min_{\tau k} = workload(f_i)$

}

} /* end of **for** loop */

End