Dynamic Data Reallocation in Hybrid Disk Arrays

Tao Xie, *Member*, *IEEE*, and Yao Sun

Abstract—Current disk arrays consist purely of hard disk drives, which normally provide huge storage capacities with low cost and high throughput for data-intensive applications. Nevertheless, they have some inherent disadvantages such as long access latencies and energy inefficiency due to their build-in mechanical mechanisms. Flash-memory-based solid state disks, on the other hand, although currently more expensive and inadequate in write cycles, offer much faster random read accesses and are much more robust and energy efficient. To combine the complementary merits of hard disks and flash disks, in this paper, we propose a hybrid disk array architecture named *h*ybrid d*i*sk s*t*orage (HIT) for data-intensive applications. Next, a dynamic data redistribution strategy called *p*erformance, *e*nergy, *a*nd *r*eliability ba*l*anced (PEARL), which can periodically redistribute data between flash disks and hard disks to adapt to the changing data access patterns, is developed on top of the HIT architecture. Comprehensive simulations using real-life traces demonstrate that compared with existing data placement techniques, PEARL exhibits its strength in both performance and energy consumption without impairing flash disk reliability.

Index Terms—Allocation strategies, energy-aware systems, reliability, secondary storage.

1 Introduction

Data placement problem or file assignment problem (FAP), the problem of allocating data (e.g., a set of files) onto multiple disks prior to serving I/O requests so that some cost functions or performance metrics can be optimized, has been extensively investigated in the past years [12], [20], [32], [33], [34]. While common cost functions are communication costs, storage costs, and queuing costs, popular performance metrics include mean response time and overall system throughput. Existing algorithms for the data placement problem generally can be divided into two categories: static and dynamic. Static data placement algorithms [20], [33], [34] require a prior knowledge about the workload characteristics including the service time of each I/O request and the access rate of each file. In addition, they demand that all files need to be allocated at the same time. Further, they assume that each file's access rate keeps constant over time, which is not realistic in many scenarios where data access patterns vary [28]. Dynamic data placement algorithms [29], [32], on the other hand, do not need any prior statistics of workload features. Also, they can allocate dynamically created files. The principle idea of dynamic data placement algorithms is to use historic information of arrived files and their recorded access characteristics to make a good allocation for each arriving file so that load balancing among multiple disks can be maintained. Since they do not possess the global information about data and their access statistics, their performance is understandably inferior to that of their static brethren. It is well understood that even finding an optimal algorithm for a cost function or a performance metric in the context of static data placement on multiple disks is an NP-complete problem [12]. Therefore, heuristic algorithms become feasible solutions to the data placement problem. Moreover, data placement algorithms alone, no matter static or dynamic, are insufficient to retain load balancing because the access pattern of a file system might change over a long-term period [28]. Consequently, an originally good data placement scheme under an initial workload may no longer be the case in a later scenario [29]. Thus, dynamic data redistribution algorithms, which can intelligently reallocate data across multiple disks to adapt to the changing workload pattern, become essential.

While high performance is the only goal pursued by traditional data placement and data redistribution algorithms [12], [18], [29], [32], modern data placement strategies like SEA [34] also take energy efficiency into account as hard disks contribute a significant percentage of total energy consumption in a computing infrastructure. For example, they can consume 27 percent of overall electricity in a data center [26]. Although recent energy-aware data placement approaches can noticeably save energy when compared with conventional data placement algorithms [34], the improvement in energy conservation is limited due to the inherent energy inefficiency of the underlying hard disk drives. Unfortunately, current disk arrays consist purely of energy-inefficient hard disk drives. Hence, a novel storage architecture that not only offers high performance but also saves energy is needed.

Originally, flash memory was devised as a storage element for consumer electrical devices such as digital cameras, cellular telephones, and personal digital assistants [6], [10]. Current flash-memory-assisted hard disk storage

- *T. Xie is with the Department of Computer Science, San Diego State University, 5500 Campanile Drive, San Diego, CA 92182. E-mail: xie@cs.sdsu.edu.*
- *Y. Sun is with the Test Engineering Department, Teradata Corporation, 12309 Briardale Way, San Diego, CA 92128. E-mail: calvin.sun@teradata.com.*

Manuscript received 29 Apr. 2009; revised 16 Oct. 2009; accepted 30 Nov. 2009; published online 28 Dec. 2009.
Recommended for acceptance by H. Jiang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-04-0187.
Digital Object Identifier no. 10.1109/TPDS.2009.175.

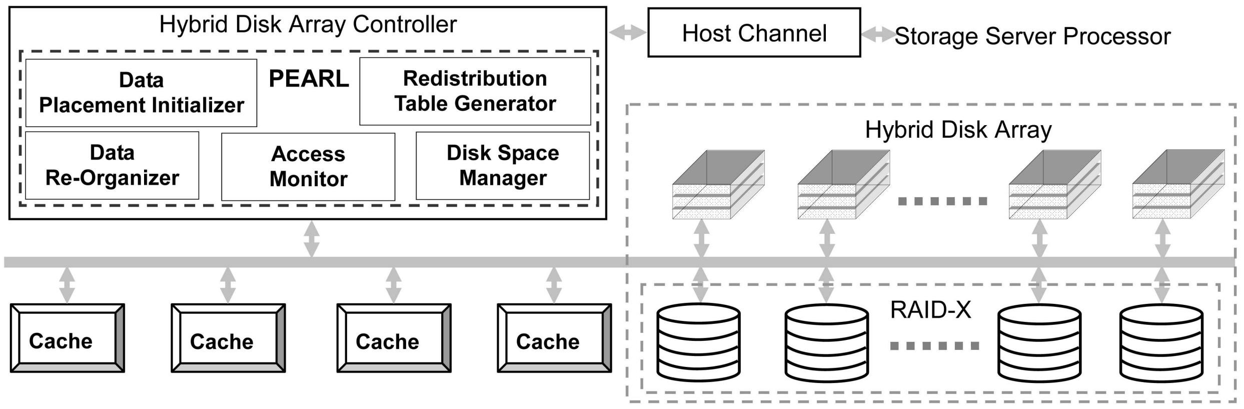


Fig. 1. The HIT architecture.

systems are mainly proposed to be applied in mobile platforms like personal laptops [10], [18] or embedded systems [7], [8]. The major purpose of using flash memory is to save energy, which is critical in a mobile computing environment or for an embedded system. Essentially, these flash memory and hard disk mixed storage systems only take flash memory as an extra layer of cache buffer [4], [18]. Very recently, flash-memory-based solid state disk (hereafter, flash disk) started to replace traditional hard disk drives in sever-class storage systems in enterprises such as Google and Baidu [11].

Flash disks have the following apparent advantages, which make them ideal storage devices for enterprise applications. First, they inherently consume much less energy than mechanical mechanism-based hard disks [6]. Second, due to their solid-state design, they are free of mechanical movements, and thus, have enhanced reliability [27]. Third, they offer much faster random access by eliminating unnecessary seek time delays and rotation latencies [16], [17]. Still, on-market flash disks also have some obvious disadvantages such as small capacity, slow random write speeds, and limited erasure cycles. Fortunately, latest breakthroughs in flash memory technology largely relax the three well-known constraints. For example, off-the-shelf flash disk products like Adtron A25FB-20 Flashpak can provide the highest capacities with up to 160 Gbytes. In addition, it offers a sustained random read speed of 78 Mbytes/second and a random write speed of 47 Mbytes/second [27]. In the mean time, some high-end Single-Level Cell (SLC) NAND flash memory now have several million erasure cycles [13], [14].

The main concern on current flash disks is their considerably higher prices. As of early 2008, flash disks are around $15 per gigabyte, whereas comparable conventional hard disk drives are typically less than $1 per gigabyte [23]. Therefore, it is wise to integrate small-capacity flash disks with high-capacity hard disk drives to form a hybrid disk array so that their complementary merits can be benefited by enterprise applications. To this end, we propose a novel *h*ybrid d*i*sk s*t*orage (HIT) architecture for next generation server-class disk arrays (see Fig. 1). As we will show later, the HIT architecture can readily outperform traditional hard disk arrays in energy conservation. As for performance and reliability, a new data management scheme is needed to judiciously utilize the complementary merits of flash disk and hard disk so that storage systems for data-intensive applications like online transaction processing (OLTP) can achieve a high performance and reliability level.

In this paper, we study dynamic data redistribution problem in the context of the new HIT architecture. An innovative dynamic data redistribution strategy called *p*erformance, *e*nergy, *a*nd *r*eliability ba*l*anced (PEARL), which periodically redistributes a set of data based on their access characteristics and the distinct features of hard disk and flash disk, is developed on top of the HIT architecture. Previous investigations on data access characteristics in real-world applications show that different data blocks may have distinct access patterns. For instance, Roselli et al. found that in a file set, some files are exclusively written without being read while some files are almost exclusively read [28]. Meanwhile, both flash disks and hard disks have their own pros and cons. For example, flash disks, though energy saving in nature, have inferior performance in write speed compared to hard disks. Besides, they have limited number of write cycles. Thus, considering both data access characteristics and features of different types of disks (flash or hard), PEARL intelligently redistributes data to its right place (a flash disk or a hard disk), where the requests targeting on it can be served efficiently.

The rest of the paper is organized as follows: In Section 2, we briefly introduce the related work and the motivation of this study. Section 3 presents the HIT architecture. The PEARL strategy and its overhead analysis are provided in Section 4. In Section 5, we evaluate the performance of PEARL based on three real-world traces. Finally, Section 6 concludes the paper with future directions.

2 Related Work and Our Idea

2.1 Dynamic Data Redistribution

Compared with numerous static data placement algorithms [12], [20], [33], [34], only very few investigations on dynamic data allocation and redistribution (or reallocation) problem [3], [29] have been accomplished. Arnan et al. observed that when several workloads on the same disk are concurrently active, the disk head has to seek back and forth between their respective locations [3]. Obviously, frequent disk head seeking significantly increases the response times of I/O requests. To alleviate this problem in multidisk systems, they proposed a data reallocation approach, which separates interfering workloads by moving one or several of the workloads to other disks where they will cause less interference [3]. The data reallocation activities are guided by a conservative interference estimation model named independent reference model (IRM). Their algorithm concentrates on reducing seek times without taking dynamically changing access patterns into account.

Scheuermann et al. proposed an array of heuristic algorithms for dynamic data redistribution by taking access pattern changes into consideration [29]. The basic idea of their algorithms is to minimize the queuing delays by distributing the load across the disks as evenly as possible and by selectively redistributing the load dynamically through a technique called "disk-cooling" [29]. However, all of their algorithms bear the following three major limitations [29]. First, they assume that all of the subrequests are uniformly distributed among the disks, which obviously contradicts the fact that real workloads generally exhibit skewed access frequencies [20], [28]. Second, their approaches just assume that the relevant workload parameters a priori can be estimated with sufficient accuracy without actually implementing any dynamic file access monitoring mechanisms. In the absence of such a monitoring mechanism, some critical workload characteristics such as the popularity of each data unit and the type of I/O requests (read or write or both) on each data unit cannot be appropriately estimated, which makes their automatic tuning methods ineffective. Even worse, the "known as a priori" assumption about workload parameters is against the spirit of dynamic data allocation and reallocation, where such workload characteristics cannot be obtained in advance. Finally, all their algorithms employ a file-specific striping policy, which means that the size of a stripe unit is file dependent. This nonuniform file striping method is not practical because it will impose a prohibitive overhead on disk array controller. Therefore, a new dynamic data allocation and reallocation (redistribution) strategy without the limitations mentioned above is needed to fully address the challenging dynamic data reorganization problem. Besides, the new dynamic data allocation and redistribution strategy should be energy aware as disk arrays are major energy consumers in a computing infrastructure [26].

2.2 Flash Disk: A Buddy of Hard Disk

NAND flash memory is traditionally used as an extra cache buffer [18], [24]. For example, a hybrid hard disk model, which embeds flash memory chips into a hard disk to make a hybrid disk, was proposed by Microsoft [24]. It takes flash memory chips as on-board memory buffers. Another typical example is SmartSaver, a disk energy-saving scheme for a mobile computer proposed by Chen et al. [10]. This scheme uses a flash drive as a standby buffer for caching and prefetching data to serve requests without disturbing disks. With the advances of flash memory technology, flash disks are integrated into personal computers or even enterprise-level applications [18], [19], [21], [22]. Kim et al. developed an energy-efficient file placement technique named pattern-based PDC (PB-PDC) [18], which adapts the existing Popular Data Concentration (PDC) algorithm [25] by separating read and write I/O requests. PB-PDC locates all read-only data upon a flash drive while it puts all rest data on a hard disk. The PB-PDC technique only concentrated on one flash drive with a single hard disk in a mobile laptop computing environment. In addition, it did not take changing workload patterns into account. Koltsidas and Viglas employed two magnetic disks and one flash disk to construct a storage layer for a database system, where data pages can be assigned to their appropriate destinations based on their access patterns [19]. Their schemes are specific to database systems. Sun ZFS, an enterprise-class file system, employed flash SSD as a secondary-level cache in the standard memory hierarchy [22].

Compared with HDD, flash disk exhibits a relatively poor random write performance. Also, they have a limited erasure/write cycle. Therefore, the proposed PEARL strategy, which is built on top of the HIT architecture, needs to take these two limitations into consideration so that neither the overall performance nor the flash disk reliability is degraded due to the intensive write activities.

2.3 Our Idea

The basic idea of the HIT architecture is to construct enterprise-level disk arrays using both small-capacity nonvolatile solid-state flash disks and traditional server-class hard disk drives. Each flash disk is coupled with its corresponding buddy hard disk to form a flash-hard disk pair. Meanwhile, all hard disks form a hard disk array organized in some RAID structure. The number of flash disks in the flash disk array is equal to the number of hard disks in the hard disk array (see Fig. 1).

Compared with existing data placement techniques [18], [19] that also employ both hard disks and flash devices, PEARL has some desirable features. First, it can be applied in a variety of applications such as Web server and OLTP. Second, it targets on a disk array-based server-class storage system rather than a single flash device [18], [19]. PEARL first divides the hard disk array into multiple data zones (hereafter, zone). Each zone is a contiguous data area that contains the same number of blocks with each block being 512 bytes. It then monitors the access pattern of each zone. Data can be dynamically created or deleted. In addition, the access pattern of each zone could vary over time. Initially, all data including newly created ones are distributed across the hard disk array in a RAID-X manner (e.g., X could be 0 or 5, see Fig. 1). At the end of each epoch, after obtaining statistics of each zone's access pattern, PEARL first separates all zones into three categories: write-excessive, read-exclusive, and read-write. If the frequency of a zone's write accesses exceeds the flash disk write cycle threshold value, it will be defined as a write-excessive zone and will stay on the hard disk array. All zones that do not belong to the write-excessive category will be further divided into two groups: read-exclusive and read-write. Zones with both read and write accesses are in the read-write group, whereas zones with read-only accesses go into the read-exclusive group. Next, PEARL selects a set of zones that is appropriate for being allocated on the flash disk array from the read-exclusive and read-write groups based on each zone's popularity and performance energy trade-off parameter (see (5)). And then, it reallocates these zones onto the flash disks. When data access pattern changes, PEARL redistributes zones between the flash disk array and the hard disk array accordingly.

3 The HIT Architecture

There are two types of flash memory: NAND flash memory and NOR flash memory [6]. Since NAND flash memory is more appropriate for data storage, we only consider NAND flash memory in this paper. Also, there are two options when one implements a flash-memory-based storage system: emulating a flash disk as a block-device like a hard disk or designing a brand new native file system directly over flash disks.

We adopt the first approach as it introduces little change of an operating system running on a host machine. In order to integrate a flash disk into an existing storage system, two important layers of software modules that sit between the file system and the flash disk are indispensable [15]. They are memory technology device (MTD) driver and flash translation layer (FTL) driver. Lower level functions of a storage medium such as read, write, and erase are provided by the MTD driver. Supported by the underlying lower level functions offered by the MTD driver, the FTL driver implements higher level algorithms like wear leveling, garbage collection, and physical/logical address translation [15]. With the assistance of the FTL driver, the file system can access the flash disk as a hard disk without being aware of the existence of the flash disk. How to develop these two software layers of modules is out of the scope of this paper. We assume that both MTD and FTL drivers are working smoothly between the file system and flash disks.

The HIT architecture is depicted in Fig. 1. Each flash disk cooperates with a hard disk through a dedicated high-bandwidth connection to compose a flash-hard disk pair. The rationale behind the one-to-one disk pair configuration is threefold. First, the equal number of the two types of disks makes balancing load between the hard disk array and the flash disk array easier. Second, it simplifies data redistribution between the two disk arrays. Last but not least, it enhances storage systems' fault tolerance and reliability by reducing disk reconstruction time when a hard disk or a flash disk fails. For example, when a hard disk fails, its partner flash disk can largely help the recovery of the failed hard disk.

In addition, both hard disks and flash disks are directly attached to the system bus. All hard disks are organized in a RAID structure like RAID-0. The hard disk array plus its associated flash disks construct a hybrid disk array. Since all flash disks are emulated as hard disks, from the hybrid disk array controller point of view, there exist two groups of same type disks in the hybrid disk array. Within the hybrid disk array controller, some data management modules like PEARL are implemented to manage data across the hybrid disk array and the controller caches. The hybrid disk array controller is connected to the storage server host through the host channel. Note that multiple hybrid disk arrays each with its own disk array controller can be connected with the storage server processor simultaneously. PEARL consists of five modules, Data Placement Initializer (DPI), Redistribution Table Generator (RTG), Data reOrganizer (DRO), Access Monitor (AM), and Disk Space Manager (DSM).

4 The PEARL Strategy

In this section, we first present a detailed description of the PEARL strategy with an illustrative example, which is followed by system models and implementation details, as well as a complexity analysis of PEARL.

4.1 Algorithm Description

The PEARL strategy judiciously yet dynamically designates each zone as either *flash favorite* or *hard favorite* based on its I/O access characteristics. Each zone is then allocated onto its favorite disk array so that the complementary merits of flash disks and hard disks can be mostly utilized while their respective disadvantages can be avoided. PEARL exploits two critical I/O workload characteristics: data access locality and data access type. The presence of access locality in I/O workload has long been recognized in the literature. For example, it is well known that 10 percent of files accessed on a Web server account for 90 percent of the server requests and 90 percent of the bytes transferred [1]. The implication of workload locality is that the overall system performance can be noticeably improved if the I/O requests on the small percentage popular data can be served more efficiently. Data access locality suggests us concentrate on the redistribution of the minority popular data. The second important I/O workload characteristic is data access type, namely write-excessive, read-exclusive, and read-write. In an investigation of file system workloads, Roselli et al. found that file access has a bimodal distribution pattern within which some files are written excessively without being read while others are almost exclusively read [28]. This observation confirms that it is feasible for PEARL to separate data into the aforementioned three categories based on the type of accesses that they have received. It is understood that read-exclusive files are suitable for flash disks as they don't contribute any erasure cycles. Further, accessing these read-exclusive files on flash disk can significantly save energy and gain potential performance enhancement due to no seek time and rotation latency any more. Similarly, write-excessive files are more appropriate for hard disks, where erasure cycle limitation doesn't apply. The most difficult task for a data redistribution strategy is to decide where some read-write popular zones should go. Unlike existing conservative algorithms such as PB-PDC [18], which immediately puts all read-write files onto hard disks to avoid any write cycles on flash disk, PEARL adopts a more open attitude and makes a smart decision based on a good trade-off between performance and energy saving.

Fig. 2 shows an illustrative example of data redistribution between hard disks and flash disks using the PEARL strategy. Assume that there are only two hard disks and two flash disks in the hybrid disk array. Further, assume that the capacity of a flash disk is half of that of a hard disk for illustration purpose. Initially, data were allocated across the hard disk array in some RAID structure and the flash disks are empty. PEARL divides the hard disk array into four zones and the size of each zone is 12 blocks. Each block is

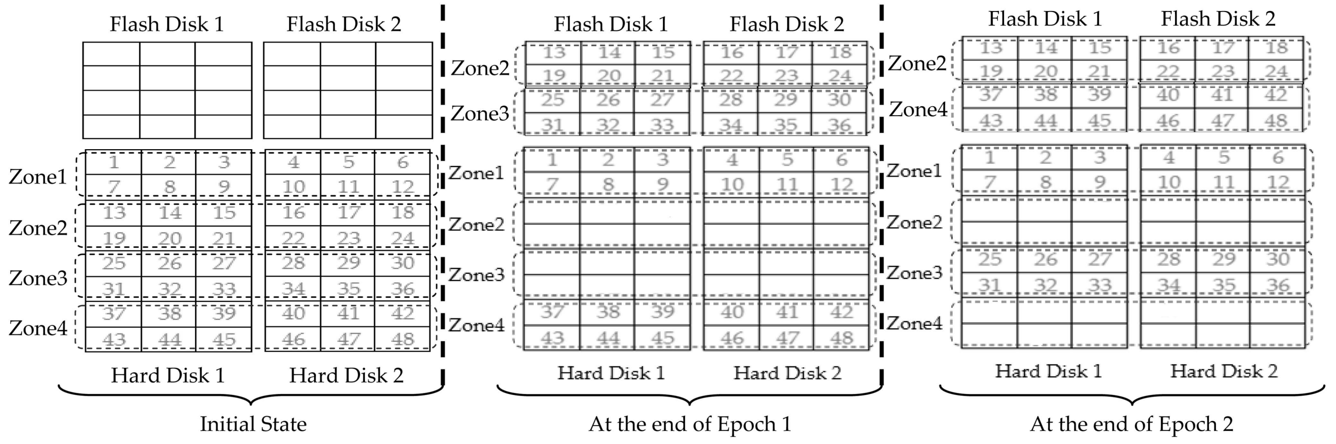


Fig. 2. An illustrative example of data redistribution by PEARL during the first two epochs.

512 bytes and has its Logical Block Address (LBA). Note that the extremely small size of a zone and the unrealistic small capacity of a disk in this example are only for illustration purpose. While the hard disk array was logically divided into four equal-size zones, the flash disk array was partitioned into two same-size zones as well. Immediately after the time instance 0, the AM starts to monitor the popularity (in terms of number of accesses) for each zone on the hard disk array. Assume that Zone 2 and Zone 3 are the hottest zones during the first epoch and they are all *flash favorite*. Thus, at the end of the first epoch, the RTG module generates a data reorganization table (DRT) and hands it to the DRO, which, in turn, transfers data of Zone2 and Zone3 from the hard disk array to the flash disk array. Therefore, all subsequent accesses targeting on the data of the two migrated zones will be directed to the flash disk array during the second epoch. Similarly, at the end of the second epoch, AM discovers that the hottest two zones change from (Zone2, Zone3) to (Zone2, Zone4). As a consequence, DRO first transfers data in Zone3 back to the hard disk array, and then, transfers data in Zone 4 to the flash disk array. By periodically updating popular flash-favorite data in the flash disk array, PEARL dynamically separates data into two disk arrays so that requests on different types of data can be served more efficiently by distinct types of disks.

4.2 System Models

The set of zones is represented as $Z = \{z_1, \ldots, z_i, \ldots, z_m\}$. The flash disk array is modeled as $FD = \{fd_1, \ldots, fd_j, \ldots, fd_n\}$, whereas the hard disk array is denoted by $HD = \{hd_1, \ldots, hd_j, \ldots, hd_n\}$. n is the number of disks. Let bl denote the size of a block in megabyte and it is assumed to be a constant in the system. A zone z_i ($z_i \in Z$) is modeled as a set of rational parameters, i.e., $z_i = (s_i, r_i, w_i)$, where s_i is the zone's size in terms of number of blocks, r_i is the zone's read access rate (1/second), and w_i is the zone's write access rate (1/second). Each hard disk's transfer rate is t^h (Mbyte/second). Its active energy consumption rate and idle energy consumption rate are p^h (Watts) and i^h (Watts), respectively. Similarly, each flash disk is modeled as $fd_j = (r^f, w^f, p^f, i^f)$, where r^f is its read rate (Mbyte/second), w^f is its write rate (Mbyte/second), p^f is its read/write power consumption rate (Watts), and i^f is its idle energy consumption rate (Watts). Although flash disks consume different amount of energy during different operations and data write patterns, we adopted a constant active energy consumption rate p^f here because it can be viewed as an average case. SK denotes average seeking time of a hard disk and RT represents average rotation latency of a hard disk. The time span of one epoch is denoted by T_e (second). Therefore, the mean service time of a block of data in zone z_i served by a hard disk is

$$mst_i^h = SK + RT + \frac{bl}{t^h}. \tag{1}$$

If the block is served by a flash disk, mean service time is

$$\begin{aligned} mst_i^f &= [(r_iT_e/s_i) * (bl/r^f) + (w_iT_e/s_i) \\ &\quad * (bl/w^f)]/[(r_i + w_i)T_e/s_i] \\ &= bl\left(\frac{r_i}{r^f} + \frac{w_i}{w^f}\right) \Big/ (r_i + w_i), \end{aligned} \tag{2}$$

where r_iT_e and w_iT_e are number of reads and writes in a zone in one epoch, respectively.

Hence, the performance gain pg_i in terms of mean service time reduction ratio of zone z_i is defined in (3):

$$\begin{aligned} pg_i &= mst_i^h/mst_i^f \\ &= \left(SK + RT + \frac{bl}{t^h}\right)(r_i + w_i) \Big/ bl\left(\frac{r_i}{r^f} + \frac{w_i}{w^f}\right). \end{aligned} \tag{3}$$

For each read-write zone, we need to decide where to store it. Thus, we need to calculate its energy gain eg_i in one epoch in (4), where ec_i^h is the energy consumption of a block in zone z_i in one epoch if it is stored in the hard disk array, and ec_i^f is the energy consumption of the block in zone z_i in one epoch if it is in the flash disk array:

$$\begin{aligned} eg_i &= ec_i^h/ec_i^f \\ &= \left[mst_i^h * p^h * (r_iT_e + w_iT_e)/s_i\right]/ \\ &\quad \left[mst_i^f * p^f * (r_iT_e + w_iT_e)/s_i\right] \\ &= \left(mst_i^h * p^h\right)/\left(mst_i^f * p^f\right). \end{aligned} \tag{4}$$

The performance energy ratio of zone z_i is defined as

$$per_i = \frac{eg_i - 1}{1 - pg_i}. \tag{5}$$

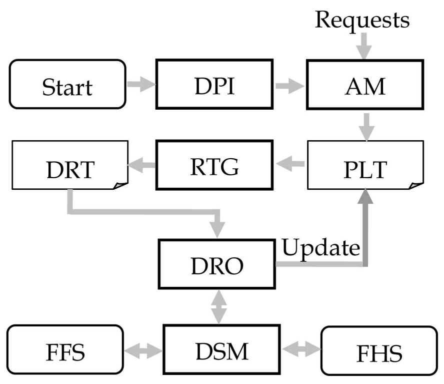


Fig. 3. The PEARL strategy.

PER represents the performance energy ratio threshold value set by administrator. Similarly, PDA is the *performance degradation allowed*, which is also a constant value set by administrator. The total number of write cycles of a flash disk is a constant WC, which is assumed to be one million in our simulation experiments. Besides, DY represents the duration years of a flash disk and we set DY as 5 years in Section 5. As a result, write cycles per second ($WCPS$) that is allowed by a flash disk is defined in (6) as below:

$$WCPS = (WC/DY)/(365 * 24 * 60 * 60). \tag{6}$$

For instance, the value of $WCPS$ in our simulations is around 0.0063 (1/second). Therefore, the reliability loss rl_i of zone z_i if it is stored on the flash disk array can be computed by

$$rl_i = \begin{cases} 1 & \text{if } w_i \geq WCPS, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

The request set is $R = \{r_1, \ldots, r_k, \ldots, r_x\}$. Each request is modeled as $r_k = (lba_k, len_k, a_k, t_k)$, where lba_k is the starting logical block address, len_k is the number of bytes that is the request accesses, a_k is the arrival time of request r_k, t_k is the type of the request r_k and it can be "*r*," "*w*," "*c*," and "*d*" representing "read," "write," "create," and "delete," respectively.

```
1.  Sort all zones in PLT into a list $Z$ in descending order of
    their no. of reads $R$ at the end of each epoch
2.  Create a blank DRT (Data Re-Distribution Table) table
3.  Initialize flash_room as the flash disk array capacity
4.  for each zone $z_i$ starting from the first one in $Z$ do
5.    case ($R$ = = 0 ) and ($W$ = = 0) /* no read, no write */
6.       Tag $z_i$ as a hard-favorite zone
7.    case ($R$ = = 0 ) and ($W$ > 0) /* only writes */
8.       Tag $z_i$ as a hard- favorite zone
9.    case ($W$ = = 0 ) and ($R$ > 0) /* only reads */
10.       Tag $z_i$ as a flash- favorite zone
11.   case ($R$ > 0 ) and ($W$ > 0) /* both reads and writes */
12.       Use Eq. 7 to compute its reliability loss $rl_i$
13.       if $rl_i$ = = 1
14.         Tag $z_i$ as a hard- favorite zone
15.       else
16.         Use Eq. 3 to compute its $pg_i$
17.         case $pg_i$ > 1
18.            Tag $z_i$ as a flash- favorite zone
19.         case (1- PDA) < = $pg_i$ < = 1
20.            Use Eq. 4 to compute its $eg_i$
21.            if $eg_i$ > 1
22.              Use Eq. 5 to calculate its $per_i$
23.              if $per_i$ > = PER
24.                  Tag $z_i$ as a flash- favorite zone
25.              else
26.                  Tag $z_i$ as a hard- favorite zone
27.              end if
28.            else
29.              Tag $z_i$ as a hard- favorite zone
30.            end if
31.         case $pg_i$ < (1-PDA)
32.            Tag $z_i$ as a hard- favorite zone
33.       end if
34.   if (flash_room – $s_i$) > = 0
35.     flash_room = flash_room – $s_i$
36.   else
37.     Exit
38.   end if
39. end for
35. Generate DRT based on PLT and the new tags
```

Fig. 4. The RTG module.

4.3 Implementations

The PEARL strategy consists of five modules that coordinate with each other via four data structures: popularity and location table (PLT), data redistribution table (DRT), free flash disk space array (FFS), and free hard disk space array (FHS) (see Fig. 3).

At the beginning, all data are striped across the hard disk array in some RAID fashion. Dynamically created files are also distributed initially across the hard disk array. PEARL first starts the DPI module, which creates PLT table for later use. After the hybrid disk array begins to serve I/O requests, PEARL launches the AM process to record each zone's popularity in terms of number of accesses within one epoch in the PLT table (see Fig. 3). The PLT table, which contains the latest popularity information of each zone and its location (flash or hard), will be used later by the RTG module (see Fig. 4) to generate the DRT table. A sample table of PLT is given in Table 1.

For example, zone3 has 452 read accesses and 37 write accesses during one epoch and its current location is on the flash disk array. After labeling all popular zones, RTG generates the DRT table, which lists all zones that need to be reallocated between the hard disk array and the flash disk array. Guided by the DRT table, the DRO module reallocates all zones in the DRT table to their favorite

TABLE 1
A Sample PLT Table

Zone ID	No. of Reads (R)	No. of Writes (W)	Place (P)
1	206	0	1
2	0	119	0
3	452	37	1
…		…	0
m	*x*	*y*	0

```
1. Re-initialize "R" and "W" fields as zero in PLT table
2. for each zone $z_k$ in the DRT that has RD = = 0 do
3.   Locate $z_k$'s record in the PLT table
4.   Return its flash space to DSM (Disk Space Manager)
5.   Change the "P" field from "1" to "0"
6.   Ask DSM for free had disk space
7.   Transfer $z_k$ from flash disk array to hard disk array
8. end for
9. for each zone $z_k$ in FRD that has RD = = 1 do
10.  Locate $z_k$'s record in the PLT table
11.  Return its hard space to DSM (Disk Space Manager)
12.  Change the "P" field from "0" to "1"
13.  Ask DSM for free flash disk space
14.  Transfer $z_k$ from hard disk array to flash disk array
15. end for
16. Delete the DRT table
```

Fig. 5. The DRO module.

destinations. During the data redistribution process, DRO consults to the DSM, which is responsible for managing both disk space for hard disk array and flash disk array. Based on the observations from real traces [28], normally the popularity of a piece of data either gradually changes or almost keeps constant. Therefore, it is feasible for PEARL to use the most recent access statistics of a zone to predict its next epoch data access pattern in a dynamic I/O workload scenario. Obviously, data redistribution is achieved at the cost of both performance degradation and extra energy consumption. Fortunately, PEARL only needs to redistribute a small portion of popular zones at the end of each epoch due to the smooth changes in data access pattern. Also, to reduce the overhead associated, PEARL confines the time span of each epoch so that frequent data reallocation can be avoided. Due to the space limit, we only present the RTG module (see Fig. 4) and the DRO module (see Fig. 5).

4.4 Complexity and Energy Overhead

In this section, we analyze the overhead of PEARL in terms of space, time, and energy. While AM is always running in hybrid disk array controller to dynamically update the PLT table, RTG, DRO, and DSM are launched only at the end of each epoch. First of all, the sizes of execution codes of all modules in PEARL are small because of their simple logic. Consequently, they only consume a very small portion of controller cache, which is normally in gigabyte scale. Second, the only extra memory required by PEARL is two tables and two arrays: PLT, DRT, FFS, FHS (see Fig. 3). Among the four data structures, PLT is the largest one. However, on average, each record in the PLT (see Table 1) only possesses 6 bytes. Thus, for a zone set with 10,000 zones, the PLT table only uses 600 KB cache memory, which is acceptable in modern storage systems. Although the AM module is running all the time in the controller cache, its runtime overhead is trivial. Note that each step in AM only takes $O(1)$ time to accomplish and the PLT table is only around 600 KB. Based on a typical I/O workload condition where 92.5 percent I/O requests are read/write, 5 percent are creation, and 2.5 percent are deletion [32], executing the loop in AM in one epoch (1,000 seconds) when the aggregate access rate is 100/second using a modern 3 GHz processor with 5 cycles per instruction only requires around 0.308 milliseconds. We also analyzed the time costs of other modules and found that they are also trivial.

PEARL has to pay extra data reallocation time and energy consumption caused by data redistribution at the end of each epoch. In the worst case of our trace-driven simulations when using the Financial2 trace [30], we observed that when each flash disk capacity is 4 GB, the total number of disks in each array is 6, and the length of an epoch is 1,000 seconds, the total size of data that swap between the hard disk array and the flash disk array is 40 MB. Hence, the file redistribution time in each epoch is around 2.06 seconds and the energy overhead caused by the file reallocation is only around 22.34 joules.

5 Performance Evaluation

This section presents results of a comprehensive experimental study comparing the proposed PEARL strategy with the PB-PDC algorithm. To the best of our knowledge, PB-PDC is the only existing data placement algorithm, which also employs the joint advantages of a hard disk and a flash memory device [18]. In this section, we first outline the experimental setup including performance metrics, system parameters, and real-life traces that we used. Next, we analyze experimental results from our simulator HITSim in Sections 5.2 and 5.3. Finally, we discuss simulation results from a validated disk storage system simulator DiskSim 4.0 [5] in Section 5.4.

5.1 Experimental Setup

In addition to using the DiskSim 4.0 simulator [5] in Section 5.4, we also developed a trace-driven simulator HITSim that models a hybrid disk array, which has one hard disk array and one flash disk array. The reason is that it is impossible for us to examine the impacts of some critical parameters like number of epochs using DiskSim. While experimental results in Section 5.4 are generated from DiskSim 4.0 [5], results presented in Sections 5.2 and 5.3 come from our own simulator HITSim. Note that some preliminary results of Section 5.4 were presented in [35]. For hard disk, HITSim uses the parameters of the Seagate Cheetah 15K.4 73.4 GB [9]. For flash disk, it adopts the specifications of the Adtron A25FB-20 Flashpak with capacity varying from 4 GB (default value) to 32 GB [31]. The main characteristics of the hard disk and the flash disk used by HITSim are shown in Table 2.

The number of flash disks is always equal to the number of hard disks and it varies in the range (6, 8, 10, 12, 14, 16) with 8 as the default value. The default length of an epoch is set to 1,000 seconds. The block size *bl* is set to 512 bytes. The default zone size is 10 Mbytes. The performance metrics are as follows:

- *Mean response time*: average response time of all file access requests submitted to the hybrid disk array.
- *Energy consumption*: energy consumed by the hybrid disk array during the process of serving all requests.
- *Write cycles per block*: the number of writes per block on a flash disk during one day.

We evaluate the PEARL and the PB-PDC algorithms by running trace-driven simulations over three real-life traces:

TABLE 2
Disk Parameters

Hard disk	**Seagate Cheetah 15K.4**
Capacity (GB)	73.4
Spindle speed (RPM)	15 K
Ave. seek time (ms)	3.5
Ave. latency (ms)	2.0
Transfer rate (Mbytes/sec)	77
Active power (watts)	17
Idle power (watts)	11.9
Flash disk	**Adtron A25FB-20 Flashpak**
Capacity (GB)	4, 8, 16, 24, 32
Access time (ms)	0.272
Seek time	0
Read (Mbytes/sec)	78
Write (Mbytes/sec)	47
Read/write power (watts)	3.43
Idle power (watts)	1.91

Financial 1, Financial 2, and WebSearch1, which have been widely used in the literature [30]. Financial1 and Financial2 were collected from requests to OLTP applications at two large financial institutions. WebSearch1 is an I/O trace from a popular search engine [30]. Since the simulation times in our experiments are much shorter compared with the time spans of the traces, we truncate each trace such that only the first 500,000 I/O requests are included. The statistics of each trace are listed in Table 3.

The impacts of four important parameters including number of disks, flash disk capacity, zone size, and number of epochs are examined in this simulation study using the three real-world traces. Besides, we also investigated the effects of access pattern changing rate by using synthetic traces in Section 5.3.

5.2 Impact of the Length of an Epoch

In this section, we examine the impact of the length of an epoch on performance, energy consumption, and flash disk reliability. Obviously, a shorter length of an epoch leads to a more frequent data reallocation, which is able to promptly adapt to the changing workload conditions. However, frequent data reallocation could incur a higher overhead in terms of CPU time, data transfer time, and energy consumption. Tardy data reallocation, on the other hand, results in lower overhead but cannot react to the changing access pattern in time, and thus, could generate a poor performance. Therefore, by tuning the length of an epoch, a good trade-off between the gain and the cost of dynamic data reallocation can be realized. In this group of experiments, we divide the length of each trace evenly into 2, 4, 6, 8, 10, and 12 epochs.

For Financial 1 trace (see Fig. 6), the optimal number of epochs is 6, which is equivalent to 893.2 seconds per epoch (see Table 3). Further increasing the number of epochs will only incur more energy consumption without any additional performance gain. For Financial 2 trace, the optimal value of the number of epochs is 2, or 1,975 seconds per epoch (see Fig. 6). The apparent difference in optimal number of epochs between Financial 1 and Financial 2 demonstrates that the workload pattern of Financial 1 is much more dynamic than that of Financial 2. Note that PEARL outperforms PB-PDC in both performance and energy consumption in all cases for Financial 1 and Financial 2 traces (see Fig. 7). The WebSearch 1 trace exhibits a similar behavior as the Financial 2 trace, which indicates that it has a stable access pattern. Further, PEARL ties with PB-PDC in mean response time and energy consumption when using the WebSearch 1 trace. This is because the I/O requests in WebSearch 1 trace are almost read-only as 99.98 percent requests are read requests. Consequently, PEARL degrades itself to PB-PDC for it allocates the same data sets onto flash disks. We also measured the impact of number of epochs on flash disk reliability in terms of write cycles caused by PEARL. More

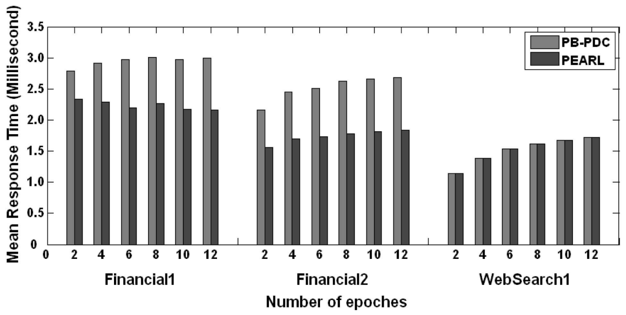


Fig. 6. Performance impact of number of epochs.

TABLE 3
Statistics of the Three Traces

Parameter	Financial 1	Financial 2	WebSearch1
Reads	370658	411344	499900
Writes	129342	88656	100
Length (sec.)	5450	3950	1500
Ave. size (bytes)	5359.2	2314.8	15402
Ave. inter-arrival time (ms)	10.9	7.9	3

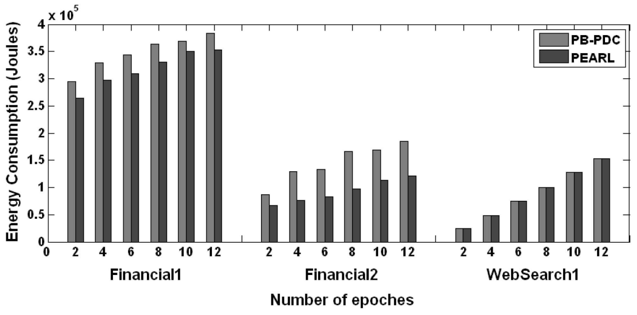


Fig. 7. Energy impact of number of epochs.

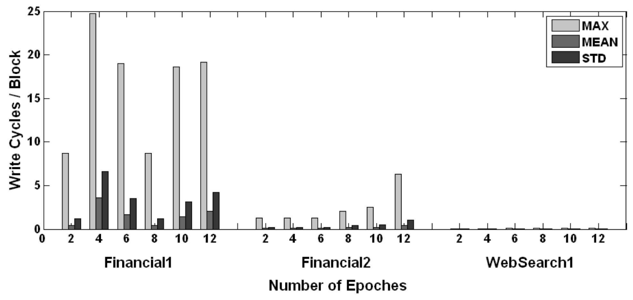


Fig. 8. Reliability Impact of number of epochs.

precisely, we monitored the maximum (MAX), mean (MEAN), and standard deviation (STD) of write cycles per block per day in Fig. 8. As we can see that the maximal write cycles per block per day of Financial 1 trace is below 25, which implies that the side effect of PEARL on flash disk reliability is rather trivial.

5.3 Experimental Results from Synthetic Workload

The advantage of use synthetic simulations is that the impacts of major workload parameters on the performance of data reallocation algorithms can be examined. In this section, we study the impacts of three major workload parameters including aggregate access rate, request size, and read/write ratio on both PEARL and PB-PDC. Aggregate access rate is defined as the average number of I/O requests arrived in a hybrid disk array per second. Request size is the mean size of requests in KB. Read/write ratio is the ratio between the number of read requests and the number of write requests.

Apparently, a higher aggregate access rate indicates a heavier workload condition. Thus, when the aggregate access rate increases, the mean response time and energy consumption of both PEARL and PB-PDC enlarge. However, PEARL only slightly increases its mean response time (see Fig. 9a). This is because the flash disks used by PEARL can share most of the increased workload, and thus, largely relieve the burden on hard disks. On the contrary, PB-PDC cannot fully take advantage of flash disks as it only allocates read-only data onto flash disks. As a consequence, heavy access rate dramatically degrades its performance. Although both algorithms consume more energy when the aggregate access rate increases, the energy gap between PEARL and PB-PDC becomes wider (see Fig. 9b). This observation implies that compared with PB-PDC, PEARL can save more energy when the aggregate access rate increases. The rationale behind is that the more requests flash disks can serve, the more energy PEARL can save. Very interestingly, when the workload is extremely light (i.e., 50 requests per second, see Fig. 9b), PEARL even consumes a little more energy than PB-PDC does. This is because the saved energy due to serving part of requests on flash disks cannot compensate the energy cost of reallocating data between the hard disk array and the flash disk array. Clearly, flash disk write cycles per block each day of PEARL boost with an increasing aggregate access rate (see Fig. 9c). However, the maximal write cycles per block per day for PEARL in Fig. 9c is smaller than 544, the threshold value of write cycles for a flash disk with one million write cycles and 5 years warranty. In other words, the impact of PEARL on flash disk reliability can be safely ignored.

Next, we examine the impact of average request size in Fig. 10. It's well understood that the mean response time increases with an enlarged average request size as disks need more time to serve each request (see Fig. 10a). Since the access latency, seek time plus rotation latency, is a

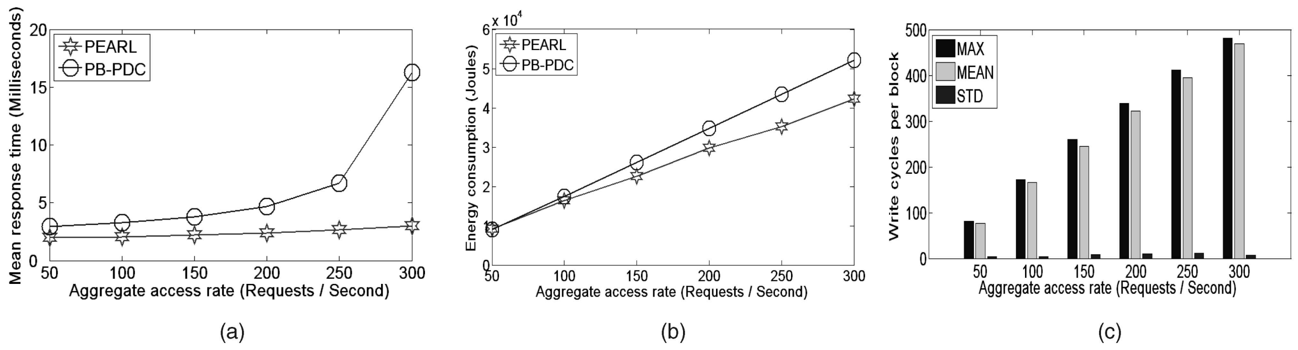


Fig. 9. Impact of aggregate access rate.

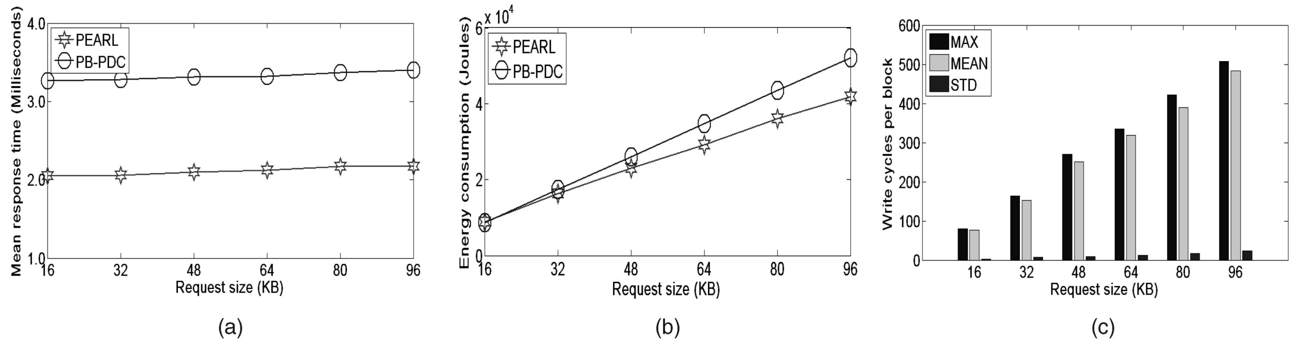


Fig. 10. Impact of request size.

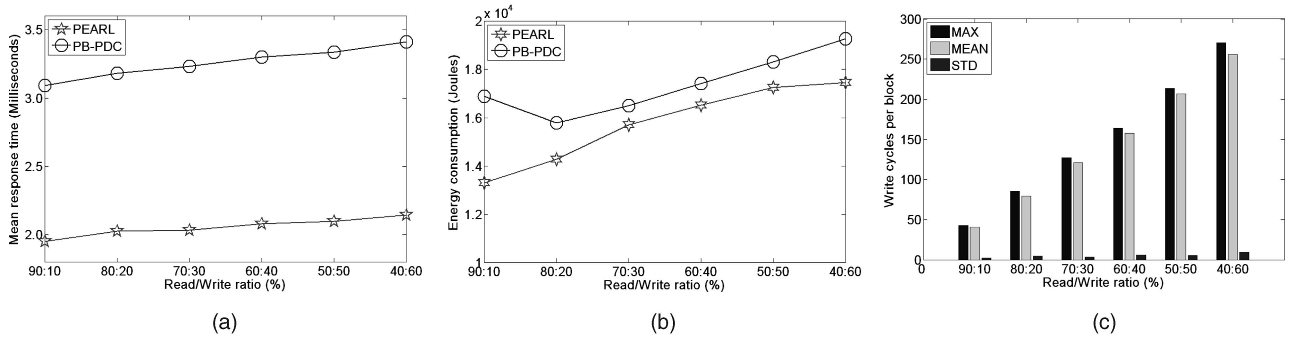


Fig. 11. Impact of read/write ratio.

dominant part in the response time of a small request visiting a hard disk, the mean response time of both PEARL and PB-PDC only increases gently when the request size is prolonged (see Fig. 10a). Since average request serving time is enlarged, energy consumption of the two algorithms increases as well (see Fig. 10b). Still, PEARL consumes less energy than PB-PDC does.

Finally, we investigate the impact of read/write ratio on performance, energy, and flash disk reliability. We observed in Fig. 11a that mean response time of both algorithms increases as read/write ratio decreases. This is because serving write requests takes more time than completing read requests. It is worth noting that PB-PDC incurs a high energy overhead when read/write ratio is 90:10 percent (see Fig. 11b). The reason is that extremely low write request percentage leads to a scenario where some hard disks even do not have opportunities to serve write requests. Although these hard disks are idle and don't contribute to serving arrived requests, they still consume energy. When the write request percentage increases, the energy consumption increases as well because write requests consume more energy than read requests do. Again, the reliability impact of PEARL on flash disk can be safely ignored (see Fig. 11c).

5.4 Experimental Results from DiskSim

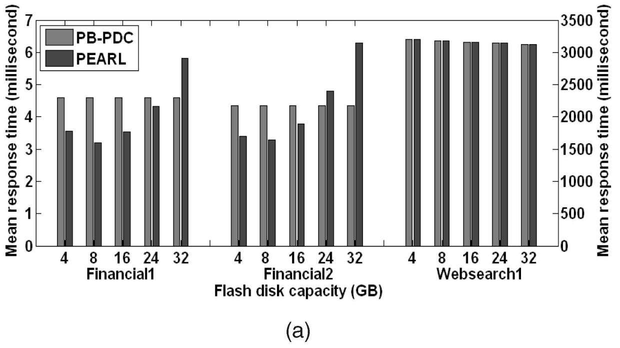
In this section, we investigate the impacts of three most important parameters (i.e., flash disk capacity, flash disk number, and zone size) by using a well-recognized and validated disk storage simulator DiskSim 4.0 from the CMU Parallel Data Lab [5]. DiskSim 4.0 emulates a hierarchy of storage components including buses, RAID array controllers, and hard disks [5]. All simulations in this section ran on an Intel Celeron 1.60 GHz 1 GB memory desktop with 32-bit SLES 10 (SUSE Linux Enterprise Server Version 10). Since DiskSim simulator does not provide Solid State Disk (SSD) model, we implemented a flash disk module derived from an existing rotating-based hard disk module, a workaround solution also adopted by Agrawal et al. [2]. Since flash disk has no seek time, we configured the two seek-time-related parameters as zero in Table 4. While the Quantum_Torando disk module was used for hard disks, a modified IBM DNES-309170W disk was used to emulate a flash disk (see Table 4). Considering that flash disk has a relatively poor data transfer rate, we choose a low-speed hard disk IBM DNES-309170W to approximate a flash disk. Since DiskSim does not measure energy consumed by each disk, we derived energy consumption for PEARL and PB-PDC from each disk's idle time and active time as well as disk active energy consumption rate and disk idle energy consumption rate.

In the first group of DiskSim simulations, we used nine hard disks and nine emulated flash disks. The capacity of the emulated flash disk was increased from 4 to 32 GB (see Fig. 12). When the flash disk capacity increases from 4 to 8 GB, the performance improvement of PEARL is also enlarged. This is because the flash disk array can accommodate more data, and thus, serve more requests when a larger size of flash disk is available. However, the performance gap between PEARL and PB-PDC narrows when the capacity of flash disk is further escalated (see Fig. 12a). The reason is that the extremely heavy workload on the flash disk array leads to a noticeable accumulated request service delay. As a result, the performance of PEARL is largely degraded. In all cases, PEARL consumes less energy than PB-PDC does (see Fig. 12b). Two important observations can be obtained from Fig. 12. First, small-capacity flash disks can significantly improve performance. Second, further increasing flash disk capacity is not helpful for performance improvement. As we can see that when flash disk capacity is 32 GB, PEARL is even worse than PB-PDC in mean response time. This scenario occurs because majority of popular data are allocated onto the flash disk array, which results in an extremely imbalanced workload distribution between the flash disk array and the hard disk array. The results of the WebSearch 1 trace in Fig. 12 verify the results in Figs. 6 and 7 in that PEARL is almost equivalent to PB-PDC when read-dominant traces are used.

In the second group of DiskSim simulations, we studied the impact of flash disk number on performance and energy consumption. The flash disk number was changed from 6 to 16 while the total capacity of the flash disk array is locked. Clearly, PEARL quickly improves its performance when the flash disk number is increased from 6 to 10 (see Fig. 13a). This

TABLE 4
Disks Used by DiskSim 4.0

Hard Disk	QUANTUM_TORNADO
Spindle speed	10025 rpms
Single cylinder seek time	1.24500 ms
Full strobe seek time	10.82800 ms
Emulated Flash Disk	IBM_DNES-309170W
Spindle speed	7200 rpms
Single cylinder seek time	0 ms
Full strobe seek time	0 ms



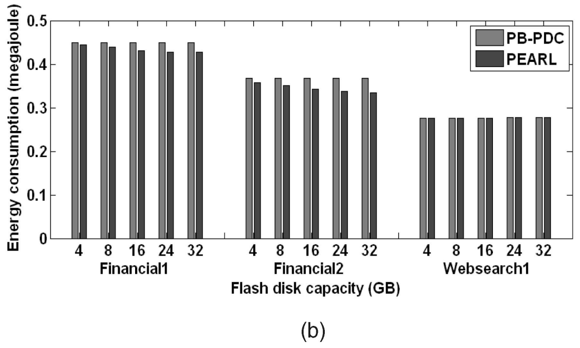
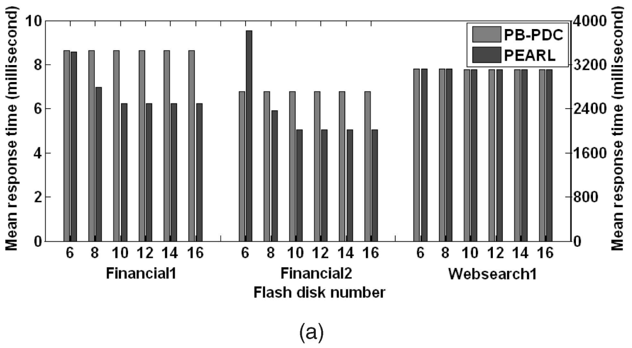


Fig. 12. Impact of flash disk capacity.



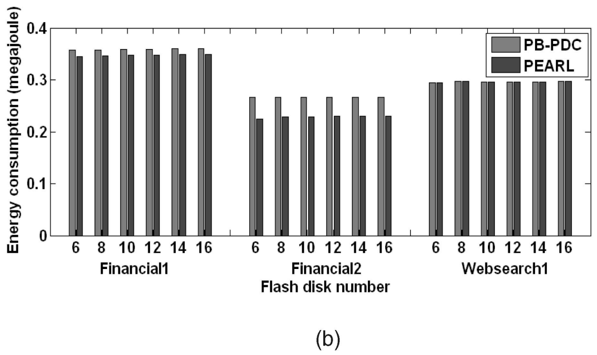
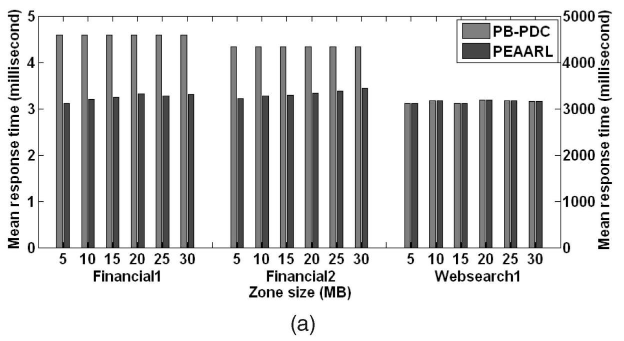


Fig. 13. Impact of number of flash disks.



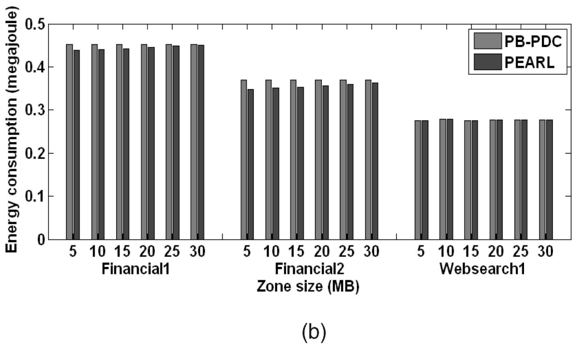


Fig. 14. Impact of zone size.

is because more flash disks can serve requests in parallel, which leads to a better performance. Again, further increasing flash disk number does not help as flash disk's poor write performance starts to negatively affect performance.

In the last group of DiskSim simulations, we tested the effect of zone size on performance and energy for the PEARL strategy. Fig. 14 shows that generally a larger zone size causes a worse performance for PEARL. The reason behind is that small zone size enables PEARL to accurately identify popular and unpopular data areas. As the zone size gets larger, the data reallocation cost is also increased. Even worse, some unpopular data are also unnecessarily transferred between the flash disk array and the hard disk array. Thus, the performance of PEARL lowers down. For WebSearch1 trace, this trend is not obvious for its request distribute is more even.

6 Conclusions

Dynamic data allocation and reallocation (redistribution) problem has been largely investigated in the past years [3], [20], [29], [32], [33], [34]. The only goal for conventional algorithms such as Hybrid Partition (HP) [20], Cool Vanilla (C-V) [32], and Simple Heat Balancing (HB) [32] is to improve performance in terms of mean response time. Nowadays, however, energy consumption becomes a severe concern in data-intensive applications like OLTP. Thus, modern dynamic data allocation and redistribution strategies need to be both performance driven and energy aware. Unfortunately, traditional pure hard-disk-based disk arrays leave little room for researchers to amend current algorithms to be energy efficient. Therefore, a new energy-efficient disk storage system is greatly needed. To this purpose, we propose a hybrid disk array architecture named HIT, which combines the complementary merits of hard disks and flash disks. More importantly, PEARL, a novel dynamic data redistribution strategy built on top of HIT, is developed and evaluated. PEARL noticeably improves performance and reduces energy consumption while maintaining flash disk reliability by limiting write cycles far below the threshold value. Comprehensive simulation experiments using synthetic workloads, real-world traces, and a validated disk simulator demonstrate that PEARL consistently outperforms an existing dynamic file assignment algorithm PB-PDC, which also employs a flash drive and a hard disk.

We will extend our scheme by considering a dramatically dynamic environment, where data access patterns may suddenly change. As a result, a high data redistribution cost may arise as the number of data zone migrations increases substantially. One possible solution is to use data replication technique. Also, we plan to investigate how to obtain an optimal epoch length for a particular workload automatically so that system performance can be further improved due to a prompt response to the changing workload.

Acknowledgments

This work was supported by the US National Science Foundation under grants CNS (CAREER)-0845105, CNS-0834466, and CCF-0702781.

References

[1] M. Arlitt and C. Williamson, "Web Server Workload Characterization: The Search for Invariants," *Proc. ACM SIGMETRICS Conf.*, pp. 126-137, May 1996.

[2] N. Agrawal, V. Prabhakaran, T. Wobber, J. Davis, M. Manasse, and R. Panigrahy, "Design Tradeoffs for SSD Performance," *Proc. USENIX Ann. Technical Conf.*, pp. 57-70, 2008.

[3] R. Arnan, E. Bachmat, T.K. Lam, and R. Michel, "Dynamic Data Reallocation in Disk Arrays," *ACM Trans. Storage*, vol. 3, no. 1, p. 2, Mar. 2007.

[4] T. Bisson and S. Brandt, "Reducing Energy Consumption with a Non-Volatile Storage Cache," *Proc. Int'l Workshop Software Support for Portable Storage*, Mar. 2005.

[5] J. Bucy, J. Schindler, S. Schlosser, G. Ganger, B. Worthington, and Y. Patt, *The DiskSim Simulation Environment Version 4.0 Reference Manual*, http://www.pdl.cmu.edu/DiskSim/index.html, 2008.

[6] K. Cash, "Flash Solid State Disks—Inferior Technology or Closet Superstar?" BiTMICRO Networks, http://www.storagesearch.com/bitmicro-art1.html, 2007.

[7] L.P. Chang and T.W. Kuo, "Efficient Management for Large-Scale Flash-Memory Storage Systems with Resource Conservation," *ACM Trans. Storage*, vol. 1, no. 4, pp. 381-418, 2005.

[8] L.P. Chang and T.W. Kuo, "An Adaptive Striping Architecture for Flash-Memory Storage," *Proc. Real-Time and Embedded Technology Systems of Embedded Systems and Applications Symp.*, pp. 187-196, 2002.

[9] "Cheetah 15K.4 Mainstream Enterprise Disc Drive Storage," Seagate Cheetah ST3146854LW.pdf.

[10] F. Chen, S. Jiang, and X. Zhang, "SmartSaver: Turning Flash Drive into a Disk Energy Saver for Mobile Computers," *Proc. Int'l Symp. Low Power Electronics and Design*, pp. 412-417, 2006.

[11] T. Claburn, "Google Plans to Use Intel SSD Storage in Servers," http://www.informationweek.com/news/storage/systems/showArticle.jhtml?articleID=207602745, 2010.

[12] W. Dowdy and D. Foster, "Comparative Models of the File Assignment Problem," *ACM Computing Surveys*, vol. 14, no. 2, pp. 287-313, 1982.

[13] D. Dumitru, "Understanding Flash SSD Performance," EasyCo LLC, Aug. 2007.

[14] A. Fitzgerald, "Flash Disk Reliability Begins at the IC Level," *J. COTS*, http://www.cotsjournalonline.com/home/article.php?id=100053, 2004.

[15] J.W. Hsieh, T.W. Kuo, and L.P. Chang, "Efficient Identification of Hot Data for Flash Memory Storage Systems," *ACM Trans. Storage*, vol. 2, no. 1, pp. 22-40, 2006.

[16] A. Kawaguchi, S. Nishioka, and H. Andmotoda, "A Flash-Memory-Based File System," *Proc. USENIX Technical Conf.*, pp. 155-164, 1995.

[17] H. Kim and S.G. Lee, "A New Flash-Memory Management for Flash Storage System," *Proc. 23rd Int'l Computer Software and Applications Conf.*, pp. 284-289, 1999.

[18] Y.J. Kim, K.T. Kwon, and J. Kim, "Energy-Efficient File Placement Techniques for Heterogeneous Mobile Storage Systems," *Proc. Sixth ACM IEEE Int'l Conf. Embedded Software*, pp. 171-177, 2006.

[19] I. Koltsidas and S.D. Viglas, "Flashing up the Storage Layer," *Proc. Int'l Conf. Very Large Databases (VLDB)*, pp. 514-525, 2008.

[20] L.W. Lee, P. Scheuermann, and R. Vingralek, "File Assignment in Parallel I/O Systems with Minimal Variance of Service Time," *IEEE Trans. Computers*, vol. 49, no. 2, pp. 127-140, Feb. 2000.

[21] S.W Lee, B. Moon, C. Park, J. Kim, and S.W. Kim, "A Case for Flash Memory ssd in Enterprise Database Applications," *Proc. ACM SIGMOD*, pp. 1075-1086, 2008.

[22] A. Leventhal, "Flash Storage Memory," *Comm. ACM*, vol. 51, pp. 47-51, July 2008.

[23] B. Levine, "Dell and Alienware Offer Samsung 64-GB SSDs," http://www.newsfactor.com/story.xhtml?story_id=55235, Sept. 2007.

[24] R. Panabaker, "Hybrid Hard Disk and ReadyDrive(tm) Technology: Improving Performance and Power for Windows Vista Mobile PCs," *Proc. Microsoft Windows Hardware Eng. Conf. (WinHEC)*, 2006.

[25] E. Pinheiro and R. Bianchini, "Energy Conservation Techniques for Disk Array-Based Servers," *Proc. Conf. Supercomputing*, pp. 88-95, June 2004.

[26] "Power, Heat and Sledgehammer," white paper, Maximum Inst., Inc., Apr. 2002.

[27] "Product Specification, Adtron A25FB-20 Flashpak Data Storage," http://www.adtron.com/pdf/A25FB-20-sum052908.pdf, 2008.

[28] D. Roselli, J.R. Lorch, and T.E. Anderson, "A Comparison of File System Workloads," *Proc. USENIX Technical Conf.*, pp. 44-54, June 2000.

[29] P. Scheuermann, G. Weikum, and P. Zabback, "Data Partitioning and Load Balancing in Parallel Disk Systems," *J. VLDB*, vol. 7, no. 1, pp. 48-66, 1998.

[30] "SPC Trace File Format Specification," http://traces.cs.umass.edu/index.php/Storage/Storage, June 2007.

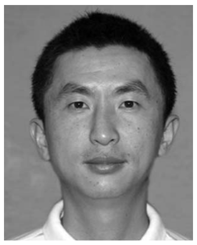
[31] "Storage Products, A25FB-20-R2spec101507.pdf," http://www.adtron.com, 2007.

[32] G. Weikum, P. Zabback, and P. Scheuermann, "Dynamic File Allocation in Disk Arrays," *Proc. ACM SIGMOD*, vol. 20, no. 2, pp. 406-415, 1991.

[33] T. Xie, "SOR: A Static File Assignment Strategy Immune to Workload Characteristic Assumptions in Parallel I/O Systems," *Proc. 36th Int'l Conf. Parallel Processing (ICPP)*, Sept. 2007.

[34] T. Xie, "SEA: A Striping-Based Energy-Aware Strategy for Data Placement in RAID-Structured Storage Systems," *IEEE Trans. Computers*, vol. 57, no. 6, pp. 748-761, June 2008.

[35] T. Xie and Y. Sun, "PEARL: Performance, Energy, and Reliability Balanced Dynamic Data Redistribution for Next Generation Disk Arrays," *Proc. 16th Ann. Meeting of the IEEE Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecomm. Systems (MASCOTS)*, Sept. 2008.



Tao Xie Tao Xie received the PhD degree in computer science from the New Mexico Institute of Mining and Technology in 2006. He received the BSc and MSc degrees from Hefei University of Technology, China, in 1991 and 2000, respectively. He is currently an assistant professor in the Department of Computer Science at San Diego State University, San Diego, California. In 2009, he received a US National Science Foundation (NSF) Early Career Award. His research interests are storage systems, high performance computing, security-aware scheduling, cluster and Grid computing, parallel and distributed systems, real-time/embedded systems, and information security. He is a member of the IEEE.

Yao Sun received the BS degree in computer science and engineering from Harbin Institute of Technology, China, in 2002, and the master degree in computer science from San Diego State University in 2008 with the honor "the most outstanding graduate student." He is currently a testing engineer at Teradata, which offers the industry's only real distribute computing database product.