SAIL: Self-Adaptive File Reallocation on Hybrid Disk Arrays

Tao Xie[1] and Deepthi Madathil[1]

[1] Department of Computer Science, San Diego State University,
San Diego, CA 92182, USA
xie@cs.sdsu.edu, madathil@rohan.sdsu.edu

Abstract. Flash-memory based solid state disks, though currently more expensive and inadequate in write cycles, offer much faster read accesses while consume much less energy compared with hard disk drives. In order to gain complementary merits of hard disks and flash disks, we propose a hybrid disk array based storage architecture for data-intensive server-class applications. Further, on top of the proposed storage architecture, a *s*elf-*a*daptive f*i*le rea*l*location strategy, called SAIL, which is able to adapt to dynamically changed file access patterns, is developed. Comprehensive trace-driven experiments demonstrate that compared with a very recent file placement technique PB-PDC, which also employs the combined advantages of a hard disk and a flash memory device, SAIL exhibits its strength in both performance and energy consumption while maintains the reliability of flash disks by confining their write cycles.

Keywords: File reallocation, flash disk, hybrid disk array, energy conservation.

1 Introduction

File assignment problem (FAP), the problem of allocating a set of files onto a disk array before they are accessed so that some cost functions or performance metrics can be optimized, have been extensively studied [9][16][20][22][23][24]. Typically, file assignment algorithms reported in the literature can be categorized into two camps: static [16][23][24] and dynamic [2][20][22]. While static file assignment algorithms require a prior knowledge about the workload statistics, dynamic file assignment strategies can adapt to varying access patterns without the prior information of the characteristics of files. In this paper, we address the problem of dynamically assigning and reallocating files in a hybrid disk array storage system where hard disks and flash disks are structured in a RAID-0 fashion, respectively.

1.1 File Allocation and Reallocation

There have been many studies [16][23][24] over static file assignment problem where the following two assumptions are held: (1) all files are to be allocated at the same

time; (2) the access frequency of each file is known as a priori and it does not change over time. In reality, however, these two assumptions are largely unrealistic. This is because file systems are highly dynamic, which implies that many files are created or deleted on the fly [22]. Moreover, the access pattern of a file system might change over a long-term period [19]. Therefore, dynamic file allocation and reallocation algorithms, which are able to intelligently allocate files dynamically created and to reorganize files to adapt to varying access pattern become indispensable.

Unfortunately, compared with numerous static file assignment algorithms [16][23][24], only very few investigations on dynamic file allocation [22] and reallocation problem [20] have been accomplished. Weikum et al. first proposed an array of heuristic algorithms for the placement of dynamically created files on a hard disk array [22]. Later on they extended their algorithms to accommodate dynamic redistributions of the data when access patterns change [20]. However, all of their algorithms bear the following three major limitations [20]. 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 [16][19]. Second, their approaches just assume that the relevant workload parameters can be estimated with sufficient accuracy without actually implementing any dynamic file access monitoring mechanisms. Finally, all their algorithms employ a file-specific striping policy, which means the size of a stripe unit is file-dependent. The non-uniform file striping method is not practical because it will impose a prohibitive overhead on disk array controller. Therefore, a new dynamic file allocation and reallocation strategy without the limitations mentioned above is needed to fully address the challenging dynamic file assignment and reorganization problem. Besides, the new strategy should be energy-aware as disk arrays contribute a significant percentage of total energy consumption in a computing infrastructure.

1.2 Why Flash Disks?

Flash memory is useful for more than just consumer devices. Current flash memory assisted hard disk storage systems are mainly proposed to be applied in mobile platforms like personal laptops [6][15] or embedded systems [4]. Essentially, these flash memory and hard disk mixed storage systems only take flash memory as an extra layer of cache buffer [1] [15]. Very recently, Kim et al. extended the usage of flash memory device by developing an energy-efficient file placement technique named PB-PDC (pattern-based PDC), which adapts the existing PDC (Popular Data Concentration) algorithm [17] by separating read and write I/O requests. More precisely, PB-PDC locates all read-only data on a flash drive while puts all the rest of data on a hard disk. Still, the PB-PDC technique only concentrated on one flash drive with a single hard disk in a mobile laptop computing environment.

We believe that the application of flash memory can go far beyond personal mobile computing and embedded system domains because it is also well-suited for enterprise level applications, where performance, energy conservation, and disk reliability need to be taken into account simultaneously [4]. Compared with hard disk drives, flash disks possess the following salient advantages [10][18]. First, they inherently consume much less energy than mechanical mechanism based hard disks [4]. Second,

because of their solid state design they are free of mechanical movements, and thus, their reliability are enhanced. Third, they offer much faster random access without seek time delays and rotation latencies [13][14]. The main concern on current flash disks is their considerably higher prices. 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.

1.3 Self-Adaptive File Reallocation

In this section we re-examine the dynamic file allocation and reallocation problem in the context of a hybrid disk array. Flash disks, though energy-saving in nature, have inferior performance in write speed compared to hard disks. Besides, they have limited number of erasure cycles. To fully exploit the advantages of flash disks and hard disks, we develop a *s*elf-*a*daptive f*i*le rea*l*location strategy named SAIL. SAIL dynamically monitors the access patterns of each file. Files can be dynamically created or deleted. In addition, the access pattern of each file could vary over time. Initially, all files including newly created files are distributed across the hard disk array in RAID-0 manner. At the end of each epoch, after obtaining statistics of each file's access pattern, SAIL first separates all files into three broad categories: write-excessive, read-exclusive, and read-write. If the frequency of a file's write accesses exceeds the suggested flash disk write frequency threshold value (e.g., 1 million times within 5 years), it will be defined as a write-excessive file and will stay on the hard disk array. All rest files will be further divided into two groups: read-exclusive and read-write. Files with both read and write accesses are in the read-write group, whereas files with only read accesses go into the read-exclusive group. Next, SAIL selects a set of files that are appropriate for being allocated on the flash disk array from the read-exclusive and the read-write groups based on each file's popularity, performance gain pg_i (Eq. 4), and energy gain eg_i (Eq. 5). And then SAIL reallocates these files onto the flash disk array. When file access pattern changes, SAIL redistributes files between the flash disk array and the hard disk array accordingly.

Based on the observations from some real-life traces [8][19], the popularity of a piece of data normally does not change dramatically in a short period of time. Thus, we argue that although the access pattern of a particular file may noticeably vary over a long run it only smoothly changes within each epoch. Therefore, it is feasible for SAIL to use the most recent access statistics of a file to predict its next epoch file access pattern in a dynamic I/O workload scenario.

2 The Hybrid Storage Architecture

There are two main types of flash memory in the market: NAND flash memory and NOR flash memory [4]. Since NAND flash memory is more appropriate for data storage [4], 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 [12]. They are MTD (memory technology device) driver and FTL (flash translation layer) 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 [12]. 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 design and implement these two software layers of modules is out of the scope of this paper. We assume that MTD and FTL drivers exist between file system and the flash disk.

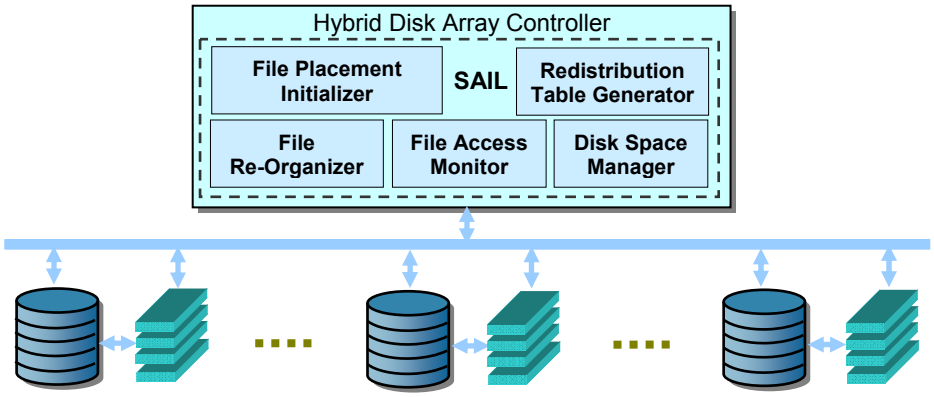

Fig. 1. Overview of the hybrid disk array architecture.

The hybrid disk array storage architecture is depicted in Fig. 1, where 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. Similarly, all flash disks are managed in the same RAID organization as the hard disk array. Besides, the number of hard disks is equal to the number of flash disks and each flash disk cooperates with a hard disk through a dedicated high-bandwidth connection to compose a disk pair. The rationale behind the disk pair configuration is three-fold. 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 file reallocation between the two disk arrays. Last but not least, the disk pair configuration obviously enhances storage system's 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 two ways. First of all, since part of data was on the flash disk, the replacement hard disk only needs to recover the data that was originally on the failed hard disk. Second, during the hard disk reconstruction process, the flash disk can still serve part of normal requests from outside clients, which greatly alleviates the workload of the replacement hard disk, which in turn speeds up the disk recovery process. Thus, the hard disk reconstruction time in the proposed hybrid disk storage architecture is shorter than that of in a pure hard disk

array architecture. The SAIL strategy is implemented as a software component within the hybrid disk array controller. It consists of the following five modules: file placement initializer, file access monitor, redistribution table generator, file re-organizer, and disk space manager. The five modules coordinate together to dynamically allocate and reallocate files between the hard disk array and the flash disk array (Fig. 2).

3 The SAIL Strategy

The methodology behind the SAIL strategy is to judiciously yet adaptively divide the entire file set into a *flash-preferred* subset and a *hard-preferred* subset based on dynamic I/O workload characteristics. Each subset of files 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 largely avoided. Supported by the proposed hybrid disk storage architecture, the goal of SAIL is to achieve a high performance, energy conservation, and desirable system reliability at the same time.

3.1 Design Methodology

SAIL realizes its goal by exploiting two critical I/O workload characteristics: file access locality and file 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% of files accessed on a web server account for 90% of the server requests and 90% of the bytes transferred [1]. Similar workload locality has also been observed in OLTP applications running in large financial institutions [11]. 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 files can be served more efficiently. File access locality suggests us concentrate on the allocation and reallocation of the minority popular files. The second important I/O workload characteristic is file 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 [19]. This observation confirms that it is feasible for SAIL to separate files into the aforementioned three categories based on the type of accesses that they received. It is easily understood that read-exclusive files are suitable for flash disks as they don't contribute any erasure cycles to flash disks. 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 file allocation and reallocation strategy is to decide where some read-write popular files should go. Unlike existing conservative algorithms such as PB-PDC [15][17], which immediately puts all read-write files onto hard disks to avoid any write cycles on flash

disk, SAIL adopts a more open attitude and makes a smart decision based on a good trade-off between performance and energy saving.

3.2 System Models

The set of files is represented as $F = \{f_1, ..., f_i, \ldots, f_m\}$. Also, the flash disk array is modeled as $FD = \{fd_1, ..., fd_j, \ldots, fd_n\}$, whereas the hard disk array is denoted by $HD = \{hd_1, ..., hd_j, \ldots, hd_n\}$. Since each file will be allocated onto either a set of hard disks or a set of flash disks in a striping manner, let sp denote the size of a stripe in Kbyte and it is assumed to be a constant in the system. A file f_i ($f_i \in F$) is modeled as a set of rational parameters, e.g., $f_i = (s_i, r_i, w_i, b_i)$, where s_i is the file's size in Mbyte, r_i is the file's read access rate (1/second), w_i is the file's write access rate (1/second), and b_i is the number of batches of the file, which is defined in Eq. 1. Assume that d_i^{start} is the starting disk of file f_i's striping distribution.

$$b_i = \begin{cases} 1, \text{if } s_i/sp \le (n - d_i^{start} + 1) \\ 1 + \left\lceil ((s_i/sp) - (n - d_i^{start} + 1))/n \right\rceil \text{ otherwise} \end{cases}. \quad (1)$$

Each hard disk's transfer rate (for both read and write) is t^h (Mbyte/second). For both hard disks and flash disks, we only consider a two- level power model: active mode and idle mode. In other words, a flash disk or a hard disk can only work in either active mode when it reads/writes data or idle mode when no such read/write activities occur. This assumption is valid for server-class applications because intensive server-level workload does not allow hard disks to spin up/down to save energy due to very slim time slots between requests. A hard disk's active energy consumption rate and idle energy consumption rate are p^h (Watts) and i^h (Watts), respectively. Similarly, a 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 active energy consumption rate (Watts), and i^f is its idle energy consumption rate (Watts). In addition, *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 file f_i served by a hard disk is

$$mst_i^h = b_i (SK + RT + sp / t^h). \quad (2)$$

However, if the file is served by a flash disk, its mean service time becomes

$$mst_i^f = [r_i T_e b_i (sp/r^f) + w_i T_e b_i (sp/w^f)]/(r_i + w_i) T_e = bi(r_i/r^f + w_i/w^f)/(r_i + w_i). \quad (3)$$

Hence, the performance gain pg_i in terms of mean service time reduction ratio of file f_i is defined in Eq. 4.

$$\begin{aligned} pg_i &= mst_i^h / mst_i^f \\ &= (SK + RT + sp / t^h)(r_i + w_i) / (r_i / r^f + w_i / w^f) \end{aligned} . \quad (4)$$

For each read-write file, we need to decide where to store it. Thus, we need to calculate its energy gain eg_i in one epoch in Eq. 5, where ec_i^h is the energy consumption of file f_i in one epoch if it is stored in the hard disk array, and ec_i^f is the energy consumption of f_i in one epoch if it is in the flash disk array.

$$eg_i = ec_i^h / ec_i^f = [\frac{p^h}{t^h}(r_i + w_i)] / [p^f (\frac{r_i}{r^f} + \frac{w_i}{w^f})] . \quad (5)$$

Since in some situations it is desirable to trade performance for energy-saving, SAIL employs a parameter named PDA (*performance degradation allowed*) to make a good trade-off between performance and energy when it makes reallocation decisions for read-write files. Essentially, PDA is a constant value set by system administrator and it is in the range [0, 1). If the system administrator believes that performance is the most important goal, he can set PDA as zero, which implies that performance degradation caused by allocating a read-write file onto the flash disk array is not permitted. If sacrificing some performance for energy-saving is desirable, he can set PDA to a value larger than zero (e.g., 20%). In this case, if a read-write file's performance gain pg_i (Eq. 4) is within the range [1-PDA, 1] and its energy gain eg_i (Eq. 5) is larger than 1, the file will be reallocated onto the flash disk so that energy-saving can be realized at the price of performance.

The total number of write cycles of a flash disk is a constant *WC*, which is assumed to be 1 million in our simulation experiments. Besides, *DY* represents the duration years of a flash disk and we set *DY* as 5 years. As a result, *WCPS* (write cycles per second) that is allowed by a flash disk is defined in Eq. 6 as below.

$$WCPS = (WC / DY) / (365 * 24 * 60 * 60) . \quad (6)$$

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

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

The request set is designated as $R = \{r_1, ..., r_k, \ldots, r_x\}$. Each request is modeled as $r_k = (fid_k, a_k, t_k)$, where fid_k is the file ID that is accessed by the request r_k, 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.

3.3 Implementations

The SAIL strategy consists of five modules (Fig. 2) that coordinate with each other via five data structures (Fig. 3): file position and popularity table, file re-distribution table, free flash space queue, free hard space queue, and deleted file queue.

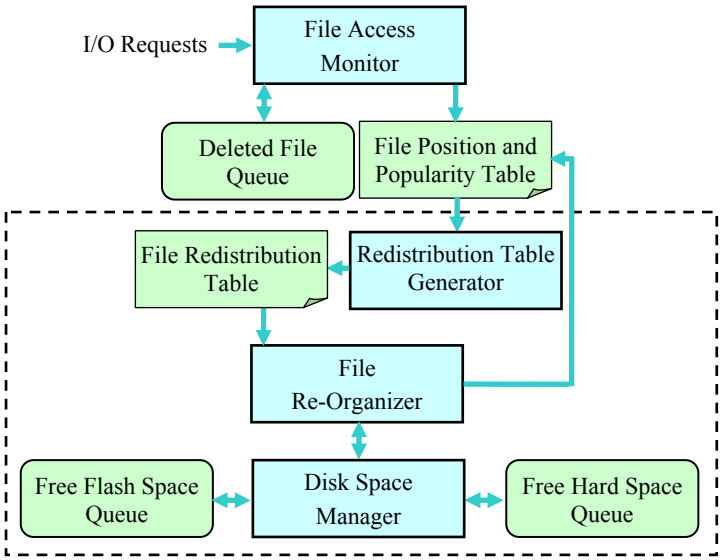

Fig. 2. The SAIL strategy; modules in the dotted rectangle execute once per epoch.

At the beginning, all files are striped across the hard disk array in a RAID-0 fashion. Dynamically created files are also distributed initially across the hard disk array. SAIL first starts the file placement initializer, which creates some important data structures such as file position and popularity table for later use (Fig. 3). After the hybrid disk array begins to serve I/O requests, SAIL launches the file access monitor to record each file's popularity in terms of number of accesses within one epoch in the file position and popularity table. The file position and popularity table, which contains the latest popularity information of each file, will be used later by the redistribution table generator to generate the file re-distribution table. After labeling all popular files, the redistribution table generator generates the file re-distribution table, which lists all files that need to be reallocated between the hard disk array and the flash disk array. Guided by the file re-distribution table, the file re-organizer reallocates all files in the file re-distribution table to their preferred destinations. During the file reallocation process, the file re-organizer consults to the disk space manager, which is responsible for managing disk space for both hard disk array and flash disk array.

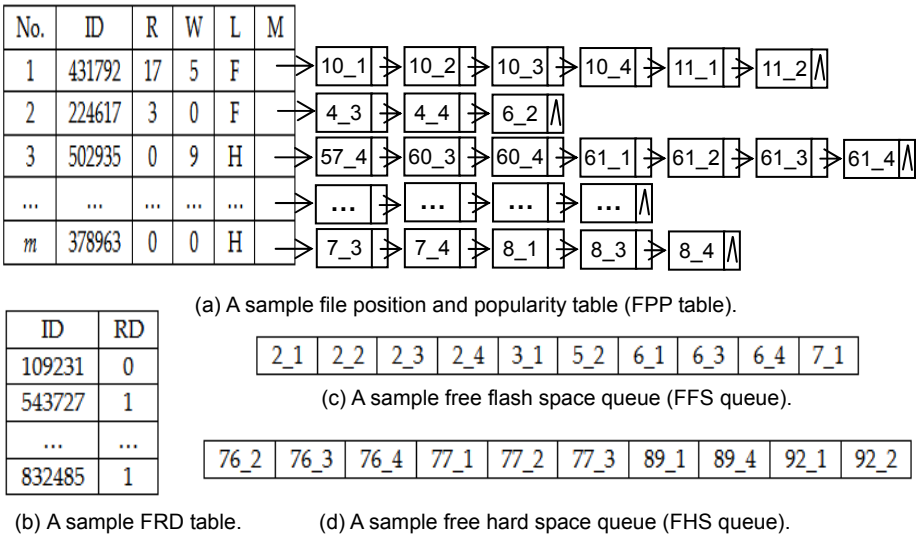

Fig. 3. Major data structures for the SAIL strategy.

Obviously, file reorganization is achieved at the cost of both performance degradation and extra energy consumption. Fortunately, SAIL only needs to re-organize a small portion of popular files at the end of each epoch due to the smooth changes in file access pattern. Also, to reduce the overhead associated with file re-organization, SAIL confines the time span of each epoch so that frequent file reallocation can be avoided.

4 Performance Evaluation

This section presents results of a comprehensive experimental study comparing the proposed SAIL strategy with the PB-PDC algorithm. To the best of our knowledge, PB-PDC is the only existing data placement algorithm that partitions data between a hard disk and a flash memory device. This is largely because how to combine newly manufactured flash disks with traditional hard disk drives to form efficient storage systems for data-intensive applications is a brand new research topic. Note that one of the most significant differences between SAIL and PB-PDC is that SAIL is integrated with RAID structures on top of a hybrid disk array for enterprise applications, whereas PB-PDC in its current status merely employs one hard disk with one flash memory device in a personal laptop computing environment. In this section, we first introduce experimental setup including performance metrics, the real trace, hard disk and flash disk characteristics, and simulation parameters that we used. Next, in Sections 4.2 we analyze experimental results.

4.1 Experimental Setup

We developed an execution-driven simulator that models a hybrid disk array, which has one hard disk array and one flash disk array (see Fig. 1). The main characteristics of the hard disk and the flash disk used in simulations are shown in Table 1. The performance metrics by which we evaluate system performance include:

- *Mean response time*: average response time of all access requests submitted to the simulated hybrid disk array storage system.
- *Energy consumption*: energy consumed by the hybrid disk array during the process of serving the entire request set.
- *Write cycles*: the maximal number of write on one flash disk during one epoch.

We evaluate the SAIL and the PB-PDC algorithms by running trace-driven simulations over the Auspex trace originated from Berkeley [8], which has been widely used in the literature. Since the simulation times in our experiments are much shorter compared with the time span of the trace, we only choose the first 1100,000 I/O requests from the trace in our experiments. We examined the impacts of flash disk capacity on system performance by controlling the parameter.

Table 1. Hard disk and flash disk parameters.

Hard disk	Seagate Cheetah 15K.4	**Flash disk**	Adtron Flashpak
Model number	ST373454FC	Model number	A25FB-20
Capacity (GB)	73.4	Capacity (GB)	4, 8, 16, 24, 32
Spindle speed (RPM)	15 K	Access time (ms)	0.272
Ave. seek time (ms)	3.5	Seek time	0
Ave. latency (ms)	2.0	Read (Mbytes/sec)	78
Transfer rate (Mbytes/sec)	77	Write (Mbytes/sec)	47
Active power (watts)	17	Read/write power (watts)	3.43
Idle power (watts)	11.4	Idle power (watts)	1.91

4.2 Impact of Flash Disk Capacity

The first group of experiments was conducted to study the impact of flash disk capacity on the performance of the two algorithms (Fig. 4). An average improvement of 24.2% in mean response time and 28.2% in energy consumption were observed by SAIL over PB-PDC (Fig. 4).

With an increased capacity of each flash disk, it is easy to understand that both SAIL and PB-PDC can improve their performance in terms of mean response time (Fig. 4). Meanwhile, energy consumption of both algorithms is reduced (Fig. 4). This is because more popular files can be placed on flash disks when more flash disk space is available. In terms of the maximal write cycles on one flash disk during an epoch, SAIL results in only 35 write cycles within one epoch (1000 seconds) when the capacity of a flash disk is 16 GB. Considering the huge capacity of a flash disk and the relatively very small number of total write cycles on it, the write cycles per block

within one epoch caused by SAIL is far from a flash disk's write cycle threshold value *WCPS* (see Eq. 6). Besides, modern flash disks normally have built-in wear-leveling techniques [21]. Thus, we believe that the impact of SAIL on flash disk reliability can be safely omitted.

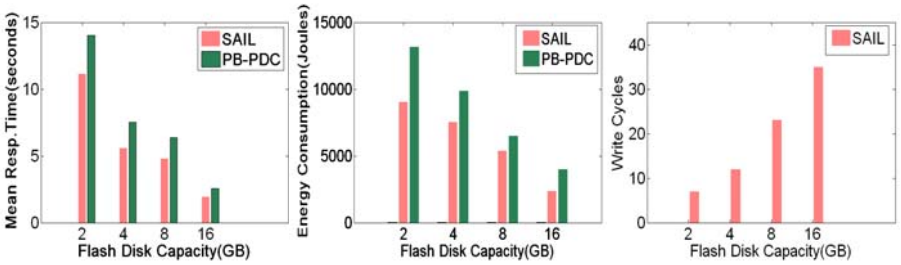

Fig. 4. An overall comparison of the two algorithms with respect to flash disk capacity.

5 Conclusions

In this paper, we address dynamic file allocation and reallocation problem in the context of a hybrid disk array. A new disk array architecture was proposed to replace traditional pure hard disk based disk arrays in server-class data-intensive applications. Powered by the proposed storage architecture, we further designed and implemented a novel self-adaptive dynamic file allocation and reallocation strategy SAIL, which judiciously separate files between one hard disk array and one flash disk array based their access patterns. Thus, the complementary merits of hard disk and flash disk can be mostly utilized while their respective shortcomings can be avoided. Comprehensive simulation experiments demonstrate that SAIL consistently outperforms an existing dynamic file assignment algorithm PB-PDC, which also employs both hard disk and flash device. Specifically, our trace-driven experimental results show that the SAIL strategy results in an average 24.2% and 28.2% performance and energy consumption improvement compared with PB-PDC. Meanwhile, in terms of write cycles, SAIL guarantees that its impact on flash disk reliability is trivial and can be safely ignored.

Acknowledgments. This work is partially supported by the US National Science Foundation under grants CNS-0834466 and CCF-0742187.

References

1. Arlitt, M., Williamson, C.: Web server workload characterization: the search for invariants. In: ACM SIGMETRICS Conference, pp. 126--137, ACM Press, New York (1996)
2. Arnan, R., Bachmat, E., Lam, T.K., Michel, R.: Dynamic data reallocation in disk arrays. ACM Transactions on Storage, Vol. 3, No. 1, 2 (2007)

3. Bisson, T., Brandt, S.: Reducing energy consumption with a non-volatile storage cache. In: International Workshop on Software Support for Portable Storage, New York (2005)
4. Cash, K.: Flash Solid State Disks - Inferior Technology or Closet Superstar? BiTMICRO Networks, http://www.storagesearch.com/bitmicro-art1.html
5. Chang, L.P., Kuo, T.W.: Efficient management for large-scale flash-memory storage systems with resource conservation. ACM Transactions on Storage, Vol. 1, 4, pp. 381--418 (2005)
6. Cheetah 15K.4 Mainstream enterprise disc drive storage, http://www.seagate.com/content/docs/pdf/marketing/Seagate_Cheetah_15K-4.pdf
7. Chen, F., Jiang, S., Zhang, X.: SmartSaver: Turning Flash Drive into a Disk Energy Saver for Mobile Computers. In: International Symposium on Low Power Electronics and Design. pp. 412--417, IEEE Press, New York (2006)
8. Dahlin, M.D., Wang, R.Y., Anderson, T.E., Patterson, D.A.: Cooperative caching: using remote client memory to improve file system performance. In: USENIX Operating Systems Design and Implementation, Vol. 1, Article No. 19 (1994)
9. Dowdy, W., Foster, D.: Comparative Models of the File Assignment Problem. ACM Computing Surveys, Vol. 14, No. 2, pp. 287--313, ACM Press, New York (1982)
10. Fitzgerald, A.: Flash Disk Reliability Begins at the IC Level. COTS Journal, http://www.cotsjournalonline.com/home/article.php?id=100053
11. Goyal, P., Jadav, D., Modha, D.S. Tewari, R.: CacheCOW: providing QoS for storage system caches. In: SIGMETRICS Conference, pp. 306--307, ACM Press, New York (2003)
12. Hsieh, J.W., Kuo, T.W., Chang, L.P.: Efficient Identification of Hot Data for Flash Memory Storage Systems. ACM Transactions on Storage, Vol. 2, No. 1, pp. 22--40 (2006)
13. Kawaguchi, A., Nishioka, S., Andmotoda, H.: A flash-memory-based file system. In: USENIX Technical Conference, pp. 155--164 (1995)
14. Kim, H., Lee, S.G.: A new flash-memory management for flash storage system. In: The 23rd International Computer Software and Applications Conference, pp. 284--289 (1999)
15. Kim, Y.J., Kwon, K.T., Kim, J.: Energy-efficient file placement techniques for heterogeneous mobile storage systems. In: The 6th ACM & IEEE International Conference on Embedded Software, pp. 171--177 (2006)
16. Lee, L.W., Scheuermann, P., Vingralek, R.: File assignment in parallel I/O systems with Minimal Variance of Service Time. IEEE Transactions on Computers, Vol. 49, No. 2, pp. 127--140, IEEE Press, New York (2000)
17. Pinheiro, E., Bianchini, R.: Energy Conservation Techniques for Disk Array-Based Servers. In: International Conference for High Performance computing, Networking, Storage and Analysis (Supercomputing 2004), pp. 88--95 (2004)
18. Product Specification, Adtron A25FB-20 Flashpak Data Storage, http://www.adtron.com/pdf/A25FB-20-sum052908.pdf
19. Roselli, D., Lorch, J.R., Anderson, T.E.: A Comparison of File System Workloads. In: USENIX Technical Conference, pp. 44--54 (2000)
20. Scheuermann, P., Weikum, G., Zabback, P.: Data partitioning and load balancing in parallel disk systems. The International Journal on Very Large Data Bases, 7, 1, pp. 48--66 (1998)
21. Storage Products, A25FB-20-R2spec101507.pdf.
22. Weikum, G., Zabback, P., Scheuermann, P.: Dynamic file allocation in disk arrays. In: ACM SIGMOD, Vol. 20, No. 2, pp. 406—415, ACM Press, New York (1991)
23. Xie, T.: SOR: A Static File Assignment Strategy Immune to Workload Characteristic Assumptions in Parallel I/O Systems. In: The 36th International Conference on Parallel Processing, IEEE Press, New York (2007)
24. Xie, T., Sun, Y.: No More Energy-Performance Trade-Off: A New Data Placement Strategy for RAID-Structured Storage Systems. In: The 14th Annual IEEE International Conference on High Performance Computing, pp.35--46, Springer Press (2007)