A Competitive Online Algorithm for File Restriping

Mario R. Medina

***Abstract*—We present a 3-competitive online algorithm for determining when to restripe a file for the case when the optimum stripe depth can take one of two possible values, depending on the application's behavior. We show that a good restriping strategy is to restripe a file when the cost incurred by delaying the restriping process equals the cost of restriping itself. We present an intuitive proof that the cost of this online algorithm is at most 3 times the cost of the optimum offline algorithm.**

***Index Terms*—disk striping, restriping, storage systems.**

I. Introduction

The scientific challenges of today and tomorrow call for modern high-performance computer systems that can achieve multi-teraflops performance when executing highly complex algorithms on very large datasets. As such, many parallel processing systems and local memories with expansive disk storage and very high speed networking have been or are being built. Nevertheless, storage device performance remains an important obstacle to the full utilization of these computing systems, as it has not kept pace with the improvements in other system components, causing a growing imbalance between the computational power and the I/O capabilities of modern high-performance systems.

Furthermore, rapid improvements in applications and algorithms, and the continuing shift from centralized to distributed computing have led to a new generation of parallel applications, which need massive amounts of data storage and have time-varying input/output demands; these applications stress I/O systems even further.

Striping data across large arrays of disks has been proposed as a technique for improving I/O performance [3], [4], [15]. Disk arrays promise high-performance I/O by exploiting the bandwidth of several disks to service a single logical request or multiple independent requests in parallel. Given current projections of commercial disk technology evolution and the increasing number of large computational systems with petabyte-size archives, disk subsystems comprising hundreds of disk drives may soon be commonplace.

Manuscript received March 22, 2007. This research was supported in part by the Chilean National Commission for Science and Technology Research (CONICYT) through FONDECYT project 1050949. Mario R. Medina is with the Depto. de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de Concepción, Concepción, Chile. Phone: (56)-(41)-2203506; FAX: (56)-(41)-2246999; e-mail: *mariomedina@udec.cl*.

Several key research issues on disk striping still remain to be explored; for example, the development of analytical models for disk restriping, that is, the re-writing of a striped file with a different stripe depth to improve I/O performance, and the analysis of techniques for trading disk storage for bandwidth by redundantly storing multiple, striped copies of files, each striped in a different way for efficient access.

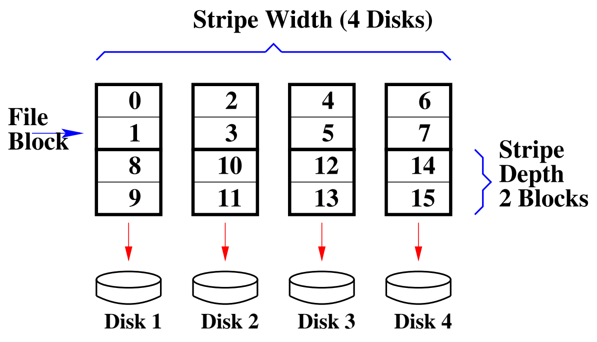

Figure 1: disk striping parameters

Figure 1 illustrates disk striping and its parameters. In this example, a 16-block file is striped across 4 disks, so that the first stripe unit (two file blocks) goes to the first disk, the second stripe unit goes to the second disk, and the n^{th} stripe unit goes to the $(n \bmod m + 1)^{th}$ disk, where m is the *stripe width*, in this case, 4 disks. The stripe unit size is usually called the *stripe depth*.

Parallel file system performance studies [6], [15] tell us of the importance of matching file system policies to application I/O access patterns. Likewise, several I/O characterization studies [10][17] have shown that many I/O intensive parallel applications exhibit complex, dynamic and often irregular I/O access patterns, rather than the regular, sequential patterns for which most file systems are optimized.

Developers wishing to improve application performance are often forced to tune the size, order and frequency of I/O requests to match the idiosyncrasies of a specific input/output system. No only does this place a substantial cognitive burden on the developers, but such optimizations are system-specific and may be inappropriate for other systems or other input/output configurations. It seems that a more promising approach would be for the parallel file system to adapt to the behavior of applications.

File systems that tune and reconfigure themselves are both feasible and increasingly necessary. The increasing complexity of I/O systems, combined with the decreasing fraction of users willing or able to tune such systems themselves, has made evident the need for self-tuning file systems.

An adaptive file system for parallel input/output must monitor both application I/O requests and I/O and file system performance so as to determine if a change to the tunable file system parameters would result in an overall performance benefit. Most file systems parameters are set at file system creation time, but some striping file systems allow dynamically setting stripe depth and/or width per file [5], [15]. Many general-purpose file systems utilize small stripe depths whereas many scientific applications perform piecewise sequential accesses to large files using large request sizes. Consequently, increasing the stripe depth for those files can result in significant execution time savings in the long run.

This paper is organized as follows: in the next section, we discuss file restriping and identify its relevant parameters. Next, we present related work pertaining to optimal stripe depth determination and dynamic adaptive file systems. Then, we discuss competitive online algorithms for file restriping and finally we derive a 3-competitive online algorithm for the 2-stripe-depth problem, giving an intuitive proof of its correctness.

II. File Restriping

An application's input/output behavior is determined by the I/O requests it performs. Hence, we can write the total time $t_{I/O}$ spent by an application performing input/output operations as:

$$t_{I/O} = \sum_{Type} \sum_{requests} t_{req}(q)$$

That is to say, it is equal to the sum over all I/O requests of all types of the time spent performing an I/O request in a striped file system with stripe depth q. This response time $t_{req}()$ depends on the stripe depth q and on parameters that are application-dependent, such as the average request size, the request type (READ(), WRITE(), LSEEK(), etc.), the average request rate λ and the I/O access pattern type (sequential, strided, random, etc.)

Given a file striped across several disks, it may be desirable for performance reasons to restripe the file, that is, to store the file across the disks using a different stripe depth. Restriping a file, then, involves reading the file stored using a given stripe depth q_1 and writing it back to disk using a different stripe depth q_2, the idea being that future I/O accesses to the file will benefit from the restriping and that the restriping cost can be justified by the future time savings due to this performance improvement. Restriping a file is an expensive operation, which is best performed between application runs, so as not to excessively disturb the system.

If a new stripe depth q_2 that improves I/O performance has been found, we must determine if the benefits of restriping the file outweigh the costs associated with the restriping itself. Given $t_{I/O}(q_1)$ and $t_{I/O}(q_2)$, that is, the total time spent by an application performing input/output operations on a file striped with stripe depths q_1 and q_2, respectively, and $t_{\text{RESTRIPE}}(q_1, q_2)$, the time to restripe a file from stripe depth q_1 to stripe depth q_2, it is usually the case that:

$$t_{RESTRIPE}(q_1, q_2) > (t_{I/O}(q_1) - t_{I/O}(q_2))$$

In other words, the cost of restriping a file is generally high, a situation that precludes it being done frequently and/or for every file. However, if we assume that the file will be accessed k times by an application with the same access patterns, and if k is large enough, then the costs of restriping can be amortized across all k accesses.

We define the *breakeven point* k_{be} as the number of times a particular file must be accessed in a similar manner by a given application to make restriping cost-effective, from a performance point of view. An approximate value for this break-even point k_{be} can be obtained by solving:

$$\frac{t_{RESTRIPE}(q_1, q_2)}{t_{I/O}(q_1) - t_{I/O}(q_2)} < k_{be}$$

Making file restriping decisions requires evaluating all the terms in this equation. The total I/O times for both stripe depths can be estimated by using an analytical model of disk access times such as the one presented in [15]. Estimating the restriping $t_{\text{RESTRIPE}}(q_1, q_2)$ depends on the algorithm used to restripe the actual data.

III. Related Work

Previous studies have sought to characterize an optimal striping unit, that is, the amount of logically contiguous data to be stored on each disk. In [3], Chen and Patterson simulated data striping across small disk arrays with synchronized spindles subject to a single class workload and random access patterns, showing that the choice of striping unit size is critical to the I/O system's performance, and that the optimum striping unit size depends significantly on only two parameters: workload concurrency and I/O system behavior. The number of outstanding requests determines workload concurrency in the disk system at any given time, and I/O system behavior is determined by the positioning time and data transfer rate of the disks. Their results also showed that the optimum striping unit size is not affected by the request size distribution of the workload.

In [14], Shenoy and Vin presented an analytical model for determining the optimum stripe depth for storing variable bit rate continuous media across disk arrays, and introduced a scheme for disk array partitioning so as to minimize the load imbalance between disks.

Of special interest is the work on *Disk Cooling* done by Scheuermann, Weikum and Zabback. In [18] they studied the problems of dynamically allocating space across a disk array when a new file is created, and of when and how to reorganize the existing files to make space for a new one. The authors proposed heuristic algorithms that provide a good compromise between maximizing the I/O performance of the disk array and minimizing the work spent in partial disk reorganizations. Their test bed is an experimental file system called FIVE that allows the stripe unit size to be chosen individually for each file or even portions of a file [19]. Other references describe an adaptive method for data allocation and dynamic load balancing in disk arrays that works by migrating file extents from heavily-accessed ("hot") disks to less loaded ("cooler") disks [11], [12]. The temperature of a disk extent is given as the ratio between the access frequency and the size of the disk extent.

Many parallel file systems allow users some measure of control over their policies, so that knowledgeable users can tailor file system behavior to suit their application. GPFS [13] allows modifications to the file system to be made online, so that disks can be added, deleted or replaced. Rebalancing the file system would then redistribute existing files across the current disk set, thus effectively changing the number of disks across which files are striped. When there are multiple copies of a file, the rebalancing procedure attempts to keep the replication status of the file system's data and metadata blocks. Nevertheless, the striping unit size is set only at file system creation time.

In [8], Matthews *et al.* showed how adaptive algorithms can be used with a log-structured file system to provide high performance across a wide range of workloads. Trace-driven simulations were used to show that, by using self-tuning principles, LFS can provide high write performance across a broader range of workloads. Also, an adaptive garbage collection mechanism that chooses between two cleaning algorithms depending on observed usage patterns is presented.

Madhyastha, Elford and Reed [7] presented an automatic technique for selecting and refining file system policies based on application access patterns and the execution environment. An automatic I/O access pattern classification framework allows an adaptive user-level parallel file system (PPFS) to select appropriate caching and prefetching policies, while performance sensors provide feedback that is used to tune policy parameters.

Among research file systems, PPFS II is a portable parallel file system with real-time control and adaptive policy control capabilities developed by the Pablo Group [15]. PPFS II is based upon the Autopilot real-time adaptive resource control library and the Nexus/Globus distributed computing infrastructure [9]. Autopilot provides PPFS II with a flexible set of performance sensors, decision procedures, and policy actuators to realize adaptive control of applications and resource management policies on both parallel and wide area distributed systems. PPFS II gives the user a large measure of control over the file system's behavior via the tuning of several parameters such as client cache sizes, cache block sizes, replacement policies, etc. PPFS II also allows self-tuning by incorporating a fuzzy-logic rulebase for adaptive striping of files across multiple disks. This rulebase is integrated into Autopilot, an adaptive control framework used to control the I/O system parameters of the PPFS II parallel file system.

The ZFS file system [5] created by Sun for its Solaris 10 operating system promises self-healing and self-managing capabilities through mechanisms such as dynamic striping, automatic block size selection and automatic filename-based performance tuning. It is currently being ported to the Linux, Mac OS X and Dragonfly BSD operating systems.

For our research, we are using the Parallel Virtual File System 2 (PVFS2) developed at Clemson University. PVFS2 is an open-source file system that allows both serial and parallel applications to store and retrieve file data distributed across a set of I/O servers utilizing traditional UNIX file semantics. PVFS focuses on file partitioning for concurrency control and allows applications to define striping parameters individually for each file. In other work, we have extended PVFS2 by adding to it file restriping capabilities.

IV. COMPETITIVE ONLINE ALGORITHMS FOR FILE RESTRIPING

In this section, we present a competitive online algorithm for determining when to restripe a file F for the case when the optimum stripe depth q_{opt} can take one of two possible values, depending on the application's behavior.

Sleator and Tarjan introduced competitive analysis, a technique for comparing online algorithms, in [16]. An online algorithm, as opposed to an offline algorithm, is designed to receive its input data as the computation proceeds. Competitive analysis evaluates online algorithms by comparing the performance of an online algorithm to that of the best-known offline algorithm [1][2][16].

In this study, we will limit the analysis to the case in which an application A is executed k times, and each execution accesses file F several times. We will represent these multiple executions of application A by the sequence $\alpha = e_1, \ldots, e_k$. We will consider for this analysis that file restriping is performed between application executions.

Let O be an online algorithm that transforms the execution sequence $\alpha = e_1, \ldots, e_k$ into a new sequence $\alpha' = g_1, \ldots, g_l$,where $k \leq l$, based only on the sequence of application executions seen so far, that is, on the sequence $e_1, \ldots, e_k$. For this purpose of this analysis, we will assume that, after every application execution, it is possible to estimate an optimum stripe depth q_{opt} for file F. Furthermore, we assume that $Cost_{\text{RESTRIPE}}(q, q_{opt})$, that is, the cost of restriping file F from its current stripe depth q to the optimum stripe depth q_{opt} can be estimated as well.

Sequence α' is generated by inserting RESTRIPE(q_{m-1}, q_m) operations into the sequence α, where q_{m-1} is the current stripe depth, and q_m is the new stripe depth to be used. Thus, every element g_i of the sequence α' is either an application execution or a restriping operation. Also, we note that removing the restriping operations from the sequence α' gets us the original sequence α back.

For any operation g_j in the transformed sequence α', we say that the stripe depth q_i is *active* at g_j if the closest restriping operation preceding g_j in the sequence α' is of the form RESTRIPE(q_{i-1}, q_i), or if i is 1 and there is no RESTRIPE() operation preceding g_j.

We define $Cost_{\alpha}^{O}$ as the cost of processing sequence α using the online algorithm O. However, given that the result of this processing is, by definition, the sequence α', we can write the cost $Cost_{\alpha}^{O}$ as the sum of the costs $C(g_i)$ of the l operations $g_1, \ldots, g_l$ that comprise α'. That is to say,

$$Cost_{\alpha}^{O} = \sum_{j=1}^{l} C(g_j)$$

As was mentioned before, the *j*-th operation g_j can be either an application execution or a restriping operation. In the first case, the cost $C(g_j)$ is equal to the cost of executing application *A* using whatever stripe depth is active at the time. Else, the cost $C(g_j)$ is equal to $Cost_{\mathrm{RESTRIPE}}$ (q_{i-1}, q_i), where q_i is the stripe depth active at g_j.

Finally, let *O** be an optimum online algorithm. Then, for any sequence α and for any online algorithm *O*, it must hold that $Cost_{\alpha}^{O*} \leq Cost_{\alpha}^{O}$.

We say an online algorithm *O* is *c*-competitive if its worst-case behavior can be bound to the cost of the optimum online algorithm by a factor of *c*. More formally, *O* is *c* -competitive if and only if, for any sequence α, there exist constants *c* and *d* such that $Cost_{\alpha}^{O} \leq cCost_{\alpha}^{O*} + d$. The adaptive restriping challenge, then, is to find an online *c*-competitive algorithm for file restriping.

V. A 3-Competitive Online Algorithm for the 2-Stripe-Depth Problem

Let us assume that, depending on the application's inputs and the execution environment, application *A* can present only two behaviors, which are best serviced by stripe depths q_1 and q_2. We assume there is at least one application execution *e* such that $Cost(e, q_1) < Cost(e, q_2)$ and, likewise, that there is at least one application execution *e*' such that $Cost(e', q_1) > Cost(e', q_2)$. Otherwise, there would be at most one RESTRIPE() operation in the *α*'sequence and the adaptive problem is trivial.

Let $Cost_{\mathrm{RESTRIPE}}(q_1, q_2)$ be the cost of restriping a file from stripe depth q_1 to stripe depth q_2, and $Cost_{\mathrm{RESTRIPE}}(q_2, q_1)$ the cost of restriping the file from stripe depth q_2 to stripe depth q_1. Let $Cost_{\mathrm{RESTRIPE}}$ be the sum of both, that is to say, $Cost_{\mathrm{RESTRIPE}} = Cost_{\mathrm{RESTRIPE}}(q_1, q_2) + Cost_{\mathrm{RESTRIPE}}(q_2, q_1)$.

Then, we propose a 3-competitive online algorithm that acts as follows:

Algorithm

Assuming that stripe depth q_1 is active, and the *k*-th execution of algorithm *A* has just ended, file *F* should be restriped with stripe depth q_2 if there exists a $j \leq k$ such that $Cost((e_j, \ldots, e_k), q_2) + Cost_{\mathrm{RESTRIPE}} \leq Cost((e_j, \ldots, e_k), q_1)$

In other words, a competitive online algorithm for the adaptive restriping problem would restripe the file whenever the additional cost incurred by choosing a non-optimal stripe depth is greater or equal to the sum of the restriping costs $Cost_{\mathrm{RESTRIPE}}(q_1, q_2) + Cost_{\mathrm{RESTRIPE}}(q_2, q_1)$.

To characterize a competitive algorithm *P* as being *c*-competitive, we must study the algorithm's worst-case behavior. In this particular case, we have that the performance of algorithm *P* is linked to how well it can track application *A*, which can switch between two behaviors. The worst-case scenario for algorithm *P* occurs when the application switches constantly between these two behaviors, forcing *P* to restripe the file *F* many times. In the following paragraphs, we will study the costs associated with these multiple restripings in more detail.

Let *P* be any online algorithm and *Q* be its "adversary", that is, an algorithm that can control the behavior of application *A*. Algorithm *Q* observes *P* and tries to devise a worst-case sequence α on which *P* performs worse than *Q*. For example, if *P* uses stripe depth q_1 exclusively, *Q* would devise a sequence of application executions that is very costly when q_1 is active and cheap when q_2 is active.

Hence, any competitive algorithm will be forced to switch between stripe depths so as to minimize the costs of performing the *α* sequence. But, if *P* changes stripe depths too often, the restriping costs will dominate, while the adversary *Q* can keep one stripe depth active, avoid the restriping costs and easily minimize the total costs.

The proposed online algorithm will restripe the file whenever it accumulates $Cost_{\mathrm{RESTRIPE}}(q_1, q_2)$ more in cost than its adversary algorithm. Choosing any value other than $Cost_{\mathrm{RESTRIPE}}(q_1, q_2)$ results in worse performance. Also, the performance of this algorithm is no more than 3 times the performance of the best offline algorithm. The following paragraphs present an intuitive proof of these statements.

Let's suppose that application *A* changes its behavior at time t_0. This change is detected and acted upon by the online algorithm *P* at time t_{DELAY}. We will call $Cost_{\mathrm{DELAY}}$ the cost incurred by the algorithm during this interval. Now, we will compare the performance of the online algorithm with the best known offline algorithm, for the two possible cases: when $Cost_{\mathrm{DELAY}} < Cost_{\mathrm{RESTRIPE}}$ and $Cost_{\mathrm{DELAY}} > Cost_{\mathrm{RESTRIPE}}$.

In the first case, let's assume that the current executions of application *A* benefit from using stripe depth q_1, which is the stripe depth chosen by online algorithm *P*. When the application behavior changes, making a restripe operation necessary, algorithm *P* will incur in the additional cost $Cost_{\mathrm{DELAY}}$, and then perform the restripe operation at a cost of $Cost_{\mathrm{RESTRIPE}}(q_1, q_2)$. The worst-case scenario calls for the application behavior to change then back to using stripe depth q_1 so as to force another restripe operation. Algorithm *P* will again incur in an additional cost $Cost_{\mathrm{DELAY}}$, and then perform the restripe to stripe depth q_1 at a cost of $Cost_{\mathrm{RESTRIPE}}(q_2, q_1)$.

An offline algorithm *Q* can balance the cost of not restriping ($Cost_{\mathrm{DELAY}}$), which we have assumed is less than the cost of restriping from stripe depth q_1 to q_2 and back ($Cost_{\mathrm{RESTRIPE}}$), and decide against restriping. Hence, the performance penalty of the best offline algorithm is $Cost_{\mathrm{DELAY}}$.

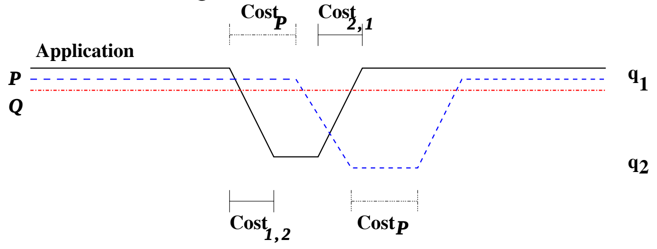

Figure 2:Online and offline algorithm behavior, $\mathbf{Cost_{DELAY} < Cost_{RESTRIPE}}$

In the second case, we assume $Cost_{\mathrm{DELAY}} > Cost_{\mathrm{RESTRIPE}}$. Again, let's assume that the current executions of application *A* benefit from using stripe depth q_1, which is the stripe depth chosen by online algorithm *P*. When the behavior of application *A* changes so as to make a restripe operation necessary, algorithm *P* will incur in the additional cost $Cost_{\mathrm{DELAY}}$, and then perform the restripe operation at a cost of $Cost_{\mathrm{RESTRIPE}}(q_1, q_2)$. The worst-case behavior corresponds to the application immediately changing its behavior again, back to using stripe depth q_1. Online algorithm *P* again incurs in the additional cost $Cost_{\mathrm{DELAY}}$, and then performs the restripe operation back to stripe depth q_1 at a cost of $Cost_{\mathrm{RESTRIPE}}(q_2, q_1)$.

An offline algorithm *Q* can balance the cost of not restriping ($Cost_{\mathrm{DELAY}}$), which we have assumed is greater than the cost of restriping from stripe depth q_1 to q_2 and back ($Cost_{\mathrm{RESTRIPE}}$), and decide to restripe the file twice. But, as the offline algorithm has complete advance knowledge of the application, it can restripe the file the instant the application behavior changes. Hence, there is no delay cost and the performance penalty of the best offline algorithm is just $Cost_{\mathrm{RESTRIPE}}$.

Figure 3 illustrates the behavior of the online and offline algorithms when $Cost_{\mathrm{DELAY}} > Cost_{\mathrm{RESTRIPE}}$.

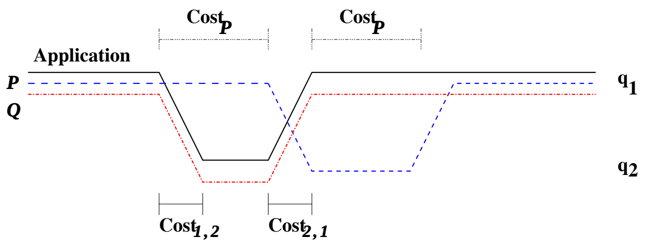

Figure 3:Online and offline algorithm behavior, $\mathbf{Cost_{DELAY} > Cost_{RESTRIPE}}$

From the figure, we see that the performance penalties of these algorithms when $Cost_{\mathrm{DELAY}} > Cost_{\mathrm{RESTRIPE}}$ are:

$$Cost_{OFFLINE} = Cost_{RESTRIPE}$$

$$Cost_{ONLINE} = 2Cost_{DELAY} + Cost_{RESTRIPE} < 3Cost_{DELAY}$$

In summary, we have two cases:

$$Cost_{DELAY} < Cost_{RESTRIPE} : Cost_{ONLINE} < 3Cost_{RESTRIPE}$$

$$Cost_{DELAY} > Cost_{RESTRIPE} : Cost_{ONLINE} < 3Cost_{DELAY}$$

If we choose $Cost_{\mathrm{DELAY}}$ to be equal to $Cost_{\mathrm{RESTRIPE}}$, we have that, for both these cases the following equation holds:

$$Cost_{OFFLINE} < Cost_{ONLINE} < 3Cost_{OFFLINE}$$

Choosing $Cost_{\mathrm{DELAY}}$ to be equal to $Cost_{\mathrm{RESTRIPE}}$ means that the online algorithm *P* should restripe file *F* as soon as the cost of not restriping when the behavior of application *A* changes is equal to the cost of restriping $Cost_{\mathrm{RESTRIPE}} = Cost_{\mathrm{RESTRIPE}}(q_1, q_2) + Cost_{\mathrm{RESTRIPE}}(q_2, q_1)$. If this condition is met, then the online algorithm described is 3-competitive.

VI. Conclusions

In summary, we have presented a online restriping algorithm for an adaptive file system. This algorithm delays the restriping process until the delay cost is equal to the cost of restriping itself. We have shown an intuitive proof that this algorithm's cost is at most 3 times the cost of the best offline algorithm. Future work will explore issues related to how to efficiently restripe a file, the development of analytical file restriping models and will tackle issues related to redundant file striping.

References

[1] Y. Bartal, A. Fiat, Y. Rabani,"Competitive algorithms for distributed data management," *24th ACM Symposium on Theory of Computing* (STOC'92), Victoria, Canada, May 1992, pp. 39-50.
[2] D. L. Black, D. D. Sleator, *Competitive algorithms for replication and migration problems*, Technical Report CMS-CS-89-201, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1989.
[3] P. M. Chen, D. A. Patterson, "Maximizing performance in a striped disk array," *17th Intl. Symposium on Computer Architecture* (ISCA'90), Seattle, WA, Jun. 1990, pp. 322-331.
[4] P. M. Chen, E. K. Lee, "Striping in a RAID Level 5 disk array," *Intl. Conference on Measurement and Modeling of Computing Systems* (SIGMETRICS'95), Ottawa, Canada, May 1995, pp. 136-145.
[5] V. Henson, M. Ahrens, J. Bonwick, "Automatic performance tuning in the Zettabyte file system," *1st. Workshop on Algorithms and Architectures for Self-Managing Systems* (SELF-MANAGE03), San Diego, CA, 2003.
[6] T. M. Madhyastha, D. A. Reed, "Intelligent, adaptive file system policy selection," *6th Symposium on the Frontiers of Massively Parallel Computation* (Frontiers'96), Annapolis, MD, Oct. 1996, pp. 172-179.
[7] T. M. Madhyastha, C. L. Elford, D. A. Reed, "Optimizing input/output using adaptive file system policies," *5th NASA Goddard Conference on Mass Storage Systems and Technologies*, College Park, MD, Sep. 1996, pp. 493-514.
[8] J. N. Matthews, D. Roselli, A. M. Costello, R. Y. Wang, T. E. Anderson, "Improving the performance of log-structured file systems with adaptive methods," *6th ACM Symposium on Operating System Principles* (SOSP'97), St. Malo, France, Oct. 1997.
[9] R. L. Ribler, H. Simitci, D. A. Reed, "The Autopilot performance-directed adaptive control system," *Future Generation Computer Systems*, 18(1), 2001, pp. 175-187.
[10] E. Rosti, G. Serazzi, E. Smirni, M. S. Squillante, "Models of parallel applications with large computation and I/O requirements," *IEEE Transactions on Software Engineering*, 28(3), Mar. 2002, pp. 286-307.
[11] P. Scheuermann, G. Weikum, P. Zabback, "Adaptive load balancing in disk arrays, " *4th Intl. Conference on Foundations of Data Organization and Algorithms* (FODO'93), Chicago, IL, Oct. 1993, pp. 345-360.
[12] P. Scheuermann, G. Weikum, P. Zabback, "Disk cooling in parallel disk systems," *Data Engineering Bulletin*, 17(3), 1994, pp. 29-40.
[13] F. Schmuck, R. Haskin, "GPFS: A shared-disk file system for large computing clusters," *1st. Conference on File and Storage Technologies* (FAST'02), Monterey, CA, Jan. 2002, pp. 231-244.
[14] P. J. Shenoy, H. M. Vin, *Efficient Striping Techniques for Variable Bit Rate Continuous Media File Servers*, Technical Report UM-CS-1998-053, University of Massachusetts at Amherst, Amherst, MA, 1998.
[15] H. Simitci, D. A. Reed, "Adaptive disk striping for parallel input/output," *7th NASA Goddard Conference on Mass Storage Systems and Technologies*, San Diego, CA, Mar. 1999, pp. 88-102.
[16] D. D. Sleator, R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Comm. ACM*, 28(2), Feb. 1985, pp. 202-208.
[17] E. Smirni, D. A. Reed, "Workload characterization of input/output intensive parallel applications," *9th Intl. Conference on Computer Performance Evaluation*, St. Malo, France, Jun. 1997, pp. 169-180.
[18] G. Weikum, P. Zabback, P. Scheuermann, "Dynamic file allocation in disk arrays," *Intl. Conference on Management of Data* (SIGMOD'91), Denver, CO, May 1991, pp. 406-415.
[19] G. Weikum, P. Zabback, "Tuning of striping units in disk-array-based file systems", *2nd. Intl. Workshop on Research Issues on Data Engineering: Transaction and Query Processing* (RIDE-TQP'92), Tempe, AZ, Feb. 1992, pp. 80-87.