DORA: A Dynamic File Assignment Strategy with Replication

Jonathan Tjioe, Renata Widjaja, Abraham Lee, and Tao Xie
of *Computer Science Department*
San Diego State University
San Diego, USA
e-mail: jontjioe@gmail.com, renata_widjaja@yahoo.com, ajsonlee@gmail.com, xie@cs.sdsu.edu

***Abstract*—Compared with numerous static file assignment algorithms proposed in the literature, very few investigations on the dynamic file allocation problem have been accomplished. Moreover, none of them has integrated file replication techniques into file assignment algorithms in a highly dynamic file system where files are created or deleted on the fly and their access patterns varied over time. We argue that file replication and file assignment can act in concert to boost the performance of parallel disk systems. In this paper, we propose a new dynamic file assignment strategy called DORA (dynamic round robin with replication). The advantages of DORA can be attributed to its two main characteristics. First, it takes the dynamic nature of file access patterns into account to adapt to a changing workload condition. Second, it utilizes file replication techniques to complement file assignment schemes so that system performance can be further improved. Experimental results demonstrate that DORA performs consistently better than existing algorithms.**

Keywords-file assignment problem; replication; dynamic; heat; round robin

I. INTRODUCTION

To satisfy the QoS (Quality of Service) demanded by end-users, prompt responses to user requests are essential for many real-world applications. For example, a Web server application that publishes significant amounts of data stored in a back-end database must reply to end-users' queries instantly before they lose patience [1][10]. Another example is an online stock broker system that facilitates the selling and buying of shares based on dynamic stock market information [18]. This real-time data service must provide guarantees on transaction timeliness and data freshness. Thus, quick response time is of paramount importance since late information could result in the loss of significant amounts of money. Obviously, the performance of these data-intensive applications largely depends on the performance of underlying parallel I/O systems, where disk arrays serve arrival requests simultaneously. More precisely, reducing mean response time of parallel disk storage systems is a must for these applications.

Among a broad spectrum of ways to reduce mean response time for parallel disk systems, file assignment, allocating files onto disk arrays before they are accessed, is an effective approach that can significantly affect the overall performance of a parallel I/O system [11][15]. A file assignment problem (FAP) can be summarized as follows. Given a set of *M* files and *N* disks, find the file-disk allocation that optimizes some cost functions or performance metrics [5]. It is well-known that finding the optimal solution for a cost function or a performance metric in the context of file assignment on multiple disks is an NP-complete problem [5]. Therefore, numerous heuristic algorithms have been proposed.

File assignment algorithms can be generally divided into two camps: static [6][11][20][21] and dynamic [2][15]. Static file assignment algorithms require a complete knowledge of workload characteristics including service time and arrival rate of requests for each file. They might be impractical in some applications because file systems are highly dynamic where many files are created or deleted on the fly [15]. In addition, the workload pattern of a file system might change over time [14]. As such, dynamic file assignment algorithms that can dynamically reorganize files to adapt to the varying access patterns become indispensable [15].

However, in a completely dynamic environment where a sub-set of files are extremely popular and receive a dominant percentage of user requests, a dynamic file assignment algorithm may no longer be helpful. The reason is that no matter where it places these hot files the load imbalance across the disks cannot be solved. In this situation, file replication techniques [3][7][8][12] can be employed to make replicas for these popular files and to distribute them onto other disks. For example, in an online streaming video service such as youtube, videos usually experience the highest demand during the first several weeks after being posted [22]. During this period, file replication techniques can make copies of the hot files to several other disks, and thus, balance the incoming requests. As a result, the response time for user requests can be reduced. Once these videos are no longer popular, their replicas can be then deleted and only the original file is kept. Unfortunately, existing dynamic file assignment algorithms [2][15] have typically overlooked the power of file replication.

In this paper, we propose a new dynamic file assignment strategy called DORA (dynamic round robin with replication), which integrates file replication techniques into file assignment schemes for a user access pattern changing environment. DORA first sorts all files according to file size. Next, it assigns the files to disks in a round-robin fashion so as to distribute the load of all files evenly across all disks. Finally, DORA dynamically keeps track of the load of all

0190-3918/09 $26.00 © 2009 IEEE
DOI 10.1109/ICPP.2009.8

files and the load on each disk. For some extremely hot files, it then creates replicas to effectively distribute request accesses on these files across all disks in a disk array.

The rest of this paper is organized as follows. Section II provides a brief summary of the related work and motivation. Section III discusses the DORA strategy. Section IV presents the experimental results. Conclusions and future work are provided in Section V.

II. Related Work And Motivation

Compared with numerous static file assignment algorithms [6][11][20][21] in literature, very few dynamic file assignment algorithms have been proposed so far [2][15]. In a multi-user environment, several workloads on one disk are concurrently active, which drives the disk head to seek back and forth between their respective locations [2]. It is evident that frequent disk head seeking significantly increases the response times of I/O requests. To address this problem, Arnan et al. proposed a data reallocation scheme that separates interfering workloads by moving one or several of the workloads to other disks where they will cause less interference [2]. Scheuermann et al. proposed an array of heuristic algorithms for the placement of dynamically created files [15]. Their algorithms are self-adaptive in the sense that they can dynamically redistribute the data when access patterns change [15]. In addition to minimizing the queuing delays by distributing the load across the disks evenly, their algorithms selectively redistribute the load by means of "disk cooling" steps [15]. Among the algorithms proposed in [15], Cool Vanilla (hereafter, C-V) exhibits the best performance.

Essentially, C-V is a vanilla Greedy algorithm [11] with additional disk cooling ability. In the original vanilla Greedy algorithm [11], the load of each file is defined as the product of the file access rate and the access service time [11]. This metric is named as the heat of the file (2) [16]. When a new file arrives, the vanilla Greedy algorithm greedily assigns the file to the disk having the least accumulated heat. Consequently, the heat of the target disk is incremented by the heat of this new file. In addition to taking the greedy file assignment steps above, the C-V algorithm dynamically checks the heat imbalance of the disks in a disk array. If the heat of a disk is higher than 1.1 of the average disk heat in a disk array, C-V relocates the hottest files from the overheated disk to the coolest disk so that the "temperature" of the overheated disk can be lowered down [15].

Evidently, disk cooling can escalate system performance by dynamically balancing the heat across the disks in a disk array through file reallocation. However, its effectiveness almost vanishes in a greatly dynamic environment where access distribution on a set of files could be overly skewed and the popularity of a file can dramatically change over time. In the former case, a small sub-set of files receive excessive accesses, and thus, the benefit of relocating these extremely hot files to the coolest disk is negligible since the coolest disk will immediately turn into a new overheated disk after receiving these hot files. In the latter case, after a hot file is migrated to the coolest disk it might quickly become unpopular, which renders the disk cooling process less than optimal. The inefficiency of the disk cooling mechanism in a completely dynamic file system becomes evident when hot file reallocation alone cannot evenly redistribute the heat of some extremely hot files across disks. By contrast, a dynamic algorithm that creates replicas for a hot file can effectively even load distribution by splitting the heat of the hot file across multiple disks. This motivates a dynamic file assignment strategy that combines file reallocation with file replication.

III. The DORA Strategy

A. System Model

A parallel disk array can be represented by a linked group D of n independent homogeneous disk drives: $D = \{d_1, ..., d_j, ..., d_n\}$. The set of files with m files that will be placed onto D can be represented by $F = \{f_1, ..., f_i, ..., f_m\}$. For simplicity, we do not consider file partitioning, and thus, each file is allocated entirely onto one disk. This assumption does not limit the generality of our model since if a file is partitioned, each partition can be viewed as a stand alone file. In fact, the majority of files in file systems and Web servers are small files where file partitioning is not necessary. A disk d_j is modeled as $d_j = (c_j, t_j, l_j)$, where c_j is the capacity in GByte, t_j is the transfer rate in MByte/second, and l_j is the disk load (i.e., sum of the heat of all files on the disk). We assume that c_j is always larger than the total size of all files assigned on d_j. This assumption is reasonable as current server-class hard disks may have a terabyte magnitude of storage capacity. A file f_i can be modeled as $f_i = (s_i, \lambda_i, v_i, h_i)$, where s_i is the size of the file, λ_i is the mean arrival rate of requests to the file, v_i is the expected service time, and h_i is the heat of the file. In this paper, disk access to a file f_i is modeled as a Poisson process with a mean access rate λ_i, which is also adopted by [11]. Also, we assume a fixed service time v_i for file f_i. The reason why this assumption is valid is two-fold. First of all, each access to file f_i could be a sequential read of the entire file, which is a typical scenario in most file systems or WWW servers [9]. Second, for large files, when the access unit is the entire file, the seek times and rotation latencies are negligible compared with the file transfer time [11]. Thus, v_i is determined by s_i and t_j if f_i is allocated on d_j

$$v_i = \frac{s_i}{t_j}. \tag{1}$$

Since the product of λ_i and v_i represents the load of f_i, we define it as the heat of f_i as below

$$h_i = \lambda_i * v_i. \tag{2}$$

Hence, the average disk load ρ can be calculated using the following equation

$$\rho = \frac{1}{n} \cdot \sum_{i=1}^{m} h_i \tag{3}$$

We use the First-Come-First-Serve (FCFS) scheduling heuristic. Suppose there are totally u requests in the request set, which is modeled as $R = \{r_1, ..., r_k, \ldots, r_u\}$. Each request is modeled as $r_k = (fid_k, a_k)$, where fid_k is the file identifier targeted by the request and a_k is the request's arrival time. In fact, the request set is a multi-class workload with each class of requests having its fixed λ_i and v_i. When a request arrives, the FCFS scheduler finds its destination disk and then directs it to the disk's local scheduling queue. To calculate the response time of a request r_k, we need to know the start time and the finish time of r_k on a disk d_j, which are denoted by $st_j(r_k)$ and $ft_j(r_k)$, respectively. There are three situations when r_k arrives in Q_j ($1 \le j \le n$), the local queue of disk d_j. First, d_j is idle and Q_j is empty. Second, d_j is busy and Q_j is empty. Third, d_j is busy and Q_j is not empty. Thus, $st_j(r_k)$ is expressed as

$$st_j(r_k) = \begin{cases} a_k, \text{ if } d_j \text{ is idle and } Q_j \text{ is empty} \\ a_k + r_j, \text{ if } d_j \text{ is busy and } Q_j \text{ is empty} \\ a_k + r_j + \sum_{r_p \in Q_j, a_p \le a_k} t_{fid_p}, \text{ otherwise} \end{cases} \quad (4)$$

where r_j is the remaining service time of the request currently running on d_j, and $\sum_{r_p \in Q_j, a_p \le a_k} t_{fid_p}$ is the overall service time of requests in Q_j that have arrival times earlier than that of r_k. It then follows that $ft_j(r_k)$ can be represented by

$$ft_j(r_k) = st_j(r_k) + t_{fid_k} \quad (5)$$

where t_{fid_k} is the service time of the file that is targeted by request r_k. The response time of request r_k can then be calculated by

$$rt_j(r_k) = ft_j(r_k) - st_j(r_k). \quad (6)$$

As a result, the mean response time of the entire request set R is

$$mrt(R) = \frac{1}{u} \sum_{k=1, 1 \le j \le n}^{u} rt_j(r_k). \quad (7$$

B. Algorithm Description

Fig. 1 outlines the DORA strategy with three basic stages: initial file assignment, dynamic replication and replica allocation, and replica garbage collection.

Stage One: Initial File Assignment (step 1 – step 4 in Fig. 1). It is well understood that an even distribution of files' heat across disks in a disk array is the key principle of successful file assignment algorithms like *sort partition* (SP) [11] and *static round-robin* (SOR) [20] whose goal is to minimize the queuing delay. As such, DORA takes a similar approach as SOR [20] in its initial file assignment stage.

Input: A parallel disk array D with n identical disks, a collection of m files in a queue F, number of hot files *HOT_NUM*, and number of epoch *EPOCH_NUM*
Output: A file allocation matrix X (n, m) and a file replication matrix R (*HOT_NUM*, n-2)
/* Initial File Assignment Process */
1. Use (3) to compute the average heat ρ
2. Sort all files in F in ascending order of v_i
3. Assign files in F across n-1 disks in a round-robin manner until the load of each disk reaches ρ
4. Assign all rest files in F to disk d_n
/* Dynamic Replication and Replica Allocation */
5. Compute *AVE_HOTFILE_HEAT* (8), *HEAT_MAX*, and *HEAT_MIN*
6. *num_epoch*=1
7. **while** *num_epoch* ≤ *EPOCH_NUM* **do**
8. f_i = 1
9. **while** f_i ≤ *HOT_NUM* **do**
10. **if** heat(f_i) ≥ *HEAT_MAX*
11. num_replica is computed from (9)
12. **end if**
13. Check R to see if f_i is already replicated
14. **if** YES
15. **if** more replicas are needed
16. Create the replicas
17. **for** each replica **do**
18. Assign it to the coolest disk among the n-2 disks where no replica exists
19. Update the coolest disk's heat and R
20. **end for**
21. **end if**
22. **else**
23. Create num_replica replicas for f_i
24. **for** each replica **do**
25. Assign it to the coolest disk among the n-2 disks
26. Update the coolest disk's heat and R
27. **end for**
28. **end if**
29. $f_i = f_i + 1$
30. **end while**
/* Garbage Replica Collection */
31. **for** each file f in R **do**
32. **if** heat(f) ≤ *HEAT_MIN*
33. Delete all replicas of f
34. Update corresponding disks' heat and R
35. **end if**
36. **end for**
37. *num_epoch* = *num_epoch* + 1
38. Update *AVE_HOTFILE_HEAT*, *HEAT_MAX*, *HEAT_MIN*, *FPT* table
39. **end while**

Figure 1. The DORA strategy.

First, DORA computes the average disk load (heat) ρ in step 1. In step 2 it sorts the file set F by file size so that files with similar sizes can be allocated onto the same disk, a smart idea used by SP [11] as well. Next, DORA separates the

most popular files onto different disks by utilizing a round-robin placement approach such that the heat of each of the n-1 disks does not exceed ρ (step 3). Note that the round-robin assignment of the sorted file set F only applies on the first n-1 disks (step 3) because DORA reserves the last disk d_n for very large files. The rationale behind this is that confining large files to a single disk could prevent them from significantly delaying responses to the requests for small files if both small files and large files are mixed together on one disk [20]. The main advantage of a round-robin based file assignment strategy is that files with higher load (heat) values will be distributed onto distinct disks so that the overall load balancing could be further improved [20]. Lastly, DORA allocates all the remaining files onto the disk d_n. In the initial file assignment stage, we assume that the characteristics of each file f_i such as its size s_i and its mean request arrival rate λ_i are known a priori. However, these parameters may vary after the initial file assignment stage.

Stage Two: Dynamic Replication and Replica Allocation (step 5 – step 30 in Fig. 1). Static file assignment algorithms such as Greedy [11], SP [11] and SOR [20] assume that workload characteristics remain unchanged over time, which is not realistic in some highly dynamic applications. For example, Roselli et al. found that the access pattern of a file system might change over a long-term period [14]. In order to adapt to changing workload conditions, DORA dynamically monitors each file's popularity (i.e., λ), during each epoch. During this fixed length of time, DORA is able to make adjustments in file replication and replica management. Several challenges arise from the dynamic nature of DORA and from the use of file replication. First of all, a File Popularity Table (FPT) needs to be constructed to record each file's mean request arrival rate in each epoch. Second, DORA keeps a record of each replicated file in matrix R, which includes its file identifier, number of replicas, and the corresponding disks that its replicas reside on. Third, at the end of each epoch, DORA needs to decide 1) if replicas should be created for a hot file; 2) if yes, how many replicas to generate; and 3) and which disks these replicas should be placed on. To answer these questions, DORA uses an array of simple yet effective empirical equations as follows.

1) Should replicas be created for a hot file? While load balancing can be performed effectively by allocating copies of hot files to multiple disks, creating replicas for hot files introduces overhead. Therefore, DORA utilizes a simple method to limit the number of hot files that are qualified to have replicas. More precisely, at the end of each epoch, it calculates the average heat of the hottest *HOT_NUM* files (i.e., *AVE_HOTFILE_HEAT*) where *HOT_NUM* is an integer input parameter provided by users (see Fig.1). For each hot file belonging to the hottest *HOT_NUM* file group, DORA uses (8) to determine whether it needs replicas (step 10).

$$replicas\ ? = \begin{cases} Yes, \text{if } h_i \geq AVE\_HOTFILE\_HEAT/2 \\ No, \text{ otherwise} \end{cases}, \quad (8)$$

where $AVE\_HOTFILE\_HEAT = \frac{1}{HOT\_NUM} \sum_{i=1}^{HOT\_NUM} h_i$.

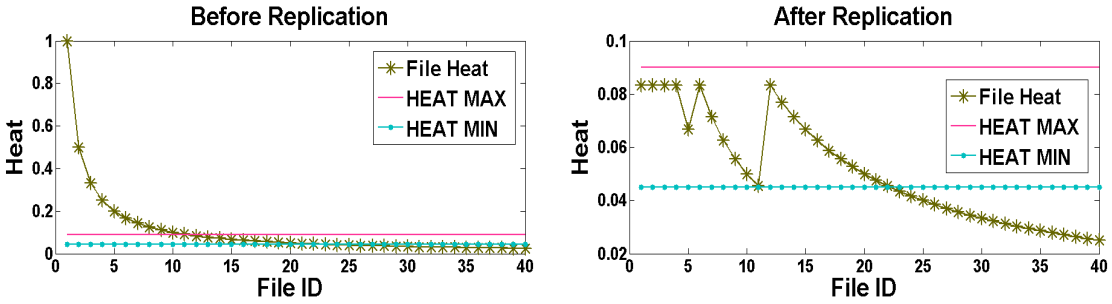
If the file's heat is no less than half of *AVE_HOTFILE_HEAT*, DORA will create replicas for it. Otherwise, no replicas will be made for this file. The maximum number of hot files that may have replicas is confined to *HOT_NUM*. Thus, the overhead caused by replication can be controlled by users. After all, in a Zipf-like file popularity distribution [4][11], only a very small percentage of files are popular. Furthermore, they are typically small in size [11]. We define two threshold values *HEAT_MAX* and *HEAT_MIN* as a half of *AVE_HOTFILE_HEAT* and a quarter of *AVE_HOTFILE_HEAT*, respectively. A file with heat no less than *HEAT_MAX* is considered an extremely hot file that needs replicas. By contrast, a file with heat no more than *HEAT_MIN* is viewed as a cold file and all its replicas if any need to be removed.

2) How many replicas should be generated? It is worth noting that the replicas of a hot file will be allocated only on n-2 disks. As mentioned before, DORA reserves disk d_n for very large files. Also, the disk where the original file is stored should not be assigned a replica as there is no benefit in having two copies of the same file on the same disk. DORA decides the number of replicas for an extremely hot file using (9) as below (step 11).

$$num\_replica = \min(floor(h_i / HEAT\_MAX), n-2) \quad (9)$$

As we can see from (9), the maximum number of replicas of a hot file is n-2.

3) Where should the replicas be placed? For each replica of a hot file, DORA assigns it to the coolest disk (i.e., the disk with lowest heat) among the n-2 disks where no replica has been stored (step 17 – step 20). This enables DORA to effectively distribute the heat of extremely hot files across disks in a disk array. In what follows, we give an example to illustrate how DORA balances the heat of extremely hot files by using replication. Assume that there are 40 files in disk d_j ($1 \leq j \leq n$) and their heat follows a Zipfian distribution [4]. Before file replication, when *HOT_NUM* is set to 20 the value of *AVE_HOTFILE_HEAT* is 0.1799 based on (8) (Fig. 2a). Therefore, *HEAT_MAX* is 0.0899 and *HEAT_MIN* is 0.0450. Also, based on (8), the first 11 files will be replicated. Fig. 2b shows an updated heat distribution after using replication for the 11 extremely hot files. Comparing Fig. 2a with Fig. 2b, one can notice that each file's heat is now below *HEAT_MAX* because for each replicated file, the original heat has been divided by the number of replicas plus 1. Consequently, the load of disk d_j is largely relieved as replicas of its hottest files have been allocated on other disks, which results in a substantially improved heat distribution across all disks in a disk array.



(a) (b)

Figure 2. An example of heat distribution before and after file replication.

Stage Three: Replica Garbage Collection (step 31 – step 36 in Fig. 1). Since file access pattern can change overtime, files that were formerly extremely hot might become unpopular after a period of time. As a result, the replicas of these once-hot-now-cold files must be removed. This requires a replica garbage collection mechanism to reclaim the disk space and update the heat on each disk. DORA employs (10) to decide which once-hot file's replicas should be deleted (step 32)

$$delete\_replicas? = \begin{cases} Yes, \text{if } h_i \le AVE\_HOTFILE\_HEAT / 4 \\ No, \text{ otherwise} \end{cases} . \quad (10)$$

Once the heat of a file f_i becomes no more than the threshold value *HEAT_MIN* (i.e., a quarter of *AVE_HOTFILE_HEAT*), all replicas of the file should be deleted and the heat of each disk involved must to be updated (step 33 – step 34).

IV. Performance Evaluation

In this section, using extensive simulations, we evaluate the performance of DORA by comparing it with one of the best existing dynamic file assignment algorithms, C-V. The advantage of using simulation is that we can easily vary all key parameters to understand their individual impact on system performance. We first introduce experimental settings in Section IV.A. Sections IV.B ~ IV.E present experimental results with detailed analysis.

A. Simulation Setup

We have developed an execution-driven simulator that models an array of Cheetah ST39205LC hard disks. The performance metrics by which we evaluate system performance are mean response time and mean disk utilization. Mean response time is the average response time of all file access requests submitted to the simulated parallel disk array, and it can be derived from (7). Since the response time rapidly grows by more than one order of magnitude as the aggregate access rate increases, the mean response times are normalized in the scale [0, 1] for all graphs in subsequent sections. Mean disk utilization is defined as the average ratio between a disk's total service time and its total operation time. The operation time is defined as the time period between the arrival time of the first file access request and the completion time of the latest finished file access request.

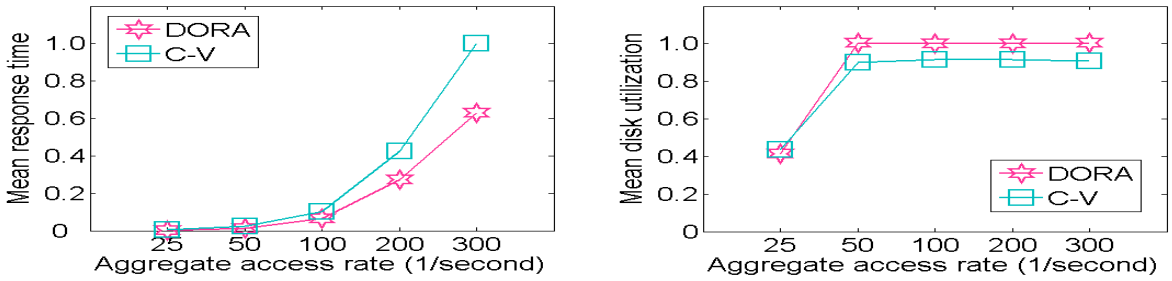
TABLE I. System Parameters

Parameter	*Value (Fixed) – (Varied)*
Number of files	(5000)
File load (heat)	Each file has heat defined as $h_i = \lambda_i * t_i$
Number of disks	(20) – (8, 12, 16, 20, 24)
Aggregate access rate (1/second)	(200) – (25, 50, 100, 200, 300)
Base file size (KB)	(30) – (15, 20, 25, 30, 35)
Skew degree	(70:30) – (50:50, 60:40, 70:30, 80:20, 90:10)

Table 1 summarizes the configurations of the synthetic workload used by experiments. Since in some real-world traces [9][13] one can observe that popular files are typically small in size while large files are relatively unpopular, we assume that the distributions of access rates across the files and file sizes were inversely correlated [11]. Disk accesses to each file are modeled as a Poisson process with a mean access rate λ_i. Because workload characteristics directly influence the performance of a file assignment algorithm we discuss the following three key parameters of the synthetic trace: aggregate access rate, skew degree, and file size distribution.

Aggregate access rate: Each file access represents a sequential read of the entire file. Hence, the service time of a file access request is proportional to the file's size. We assume that each file has a fixed request arrival rate λ_i during stage one (initial file assignment process) and the arrival interval times are exponentially distributed. Note that the value of λ_i might change after the first epoch starts. The aggregate access rate of the entire disk system is defined as $\sum_{i=1}^{m} \lambda_i$. The value of m is set to 5,000.

Skew degree: Since the frequency of file access usually exhibits a Zipf-like distribution [11], we assume that the distribution of file access requests is a Zipf-like distribution with a skew parameter $\theta = \log\frac{X}{100} / \log\frac{Y}{100}$, where X percent of all accesses were directed to Y percent of files. The value of X:Y is defined as skew degree in this paper. In our simulations, we tested five values of skew degree (X:Y) changing from 50:50 to 90:10.



(a) **(b)**

Figure 3. Performance Impact of aggregate access rate.

File size distribution: The distribution of access rates across the files and the distribution of file sizes were inversely correlated with the same skew parameter θ. Section 4.E. measures the performance results assuming a Zipf-like file size distribution. The parameter file size base is defined as the smallest file size in file set F based on a Zipf-like distribution.

B. Overall Performance Comparison

Aggregate access rate is a critical workload parameter because it indicates the intensity of the access load imposed on the disk array where *m* files have been assigned. To evaluate the performance of DORA in different levels of workload intensity, we compare it with C-V when aggregate access rate increases from 25 to 300 in this section. The results from Fig. 3 clearly show that the mean response time quickly escalates as the aggregate access rate increases. This consequence is expected as a higher aggregate access rate results in a much longer queuing delay. However, the improvement in mean response time of DORA over C-V becomes more significant with an increasing aggregate access rate (Fig. 3a). The reason behind is that when aggregate access rate increases DORA starts to fully exhibit its strength through its hot file replication and replica allocation steps, which effectively reshape an overly skewed heat distribution to a more even one. An improved heat distribution leads to a shorter mean response time. Compared with C-V, on average DORA reduces mean response time by 31.7%. As the workload becomes more intensive, the mean disk utilization of DORA quickly arises to 1 when aggregate access rate is larger than 25 (Fig. 3b). Meanwhile, C-V keeps its mean disk utilization around 91%, which indicates some disks still have idle times although the system workload is already very heavy. This is expected because when aggregate access rate increases, the heat distribution becomes overly skewed since many more requests concentrate on a small percentage of extremely hot files. In this situation, the disk cooling method used by C-V loses its efficiency as simply moving these extremely hot files to other disks cannot solve the uneven heat distribution problem.

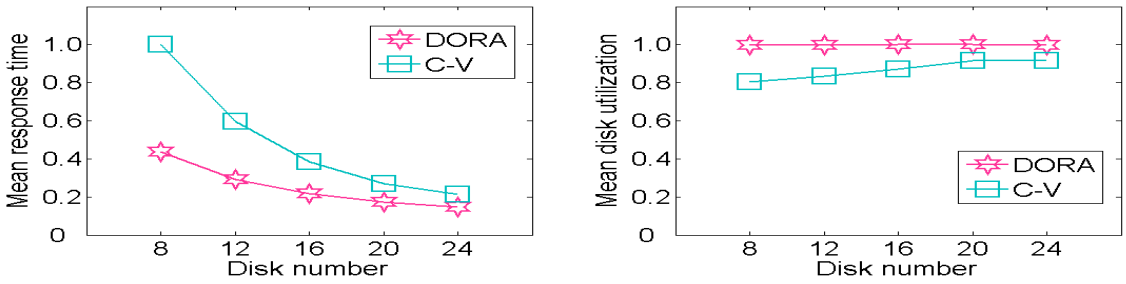
C. Scalability

To investigate the scalability of the two algorithms, we scale the number of disks in the system from 8 to 24. The aggregate access rate is configured to 200. The skew degree is still set to 70:30. Fig. 4 plots the performance of the two algorithms as functions of the number of disks.

Results from Fig. 4 demonstrate that both algorithms deliver better performance in mean response time and mean disk utilization when the number of disks increases. This is because each disk is assigned fewer files when the system is scaled up. One important observation is that DORA consistently outperforms C-V in all cases. Especially, when the number of disks is only 8, the improvement in mean response time of DORA over C-V is as high as 56.2% (Fig. 4a). The implication of this observation is that DORA is suitable for a parallel I/O system where the number of disks is not sufficient. Again, in terms of mean disk utilization, DORA keeps all disks busy to serve arrival requests at full speed (Fig. 4b). On the contrary, C-V can do so only when more disks are available so that some extremely hot files can be effectively migrated on to disks with lower heat (Fig. 4b).

D. Impact of Skew Degree

To verify the performance impact of the skew parameter θ, we evaluate the performance as functions of skew degree. When the skew degree increases from 50:50 to 90:10, both



(a) **(b)**

Figure 4. Performance Impact of number of disk.

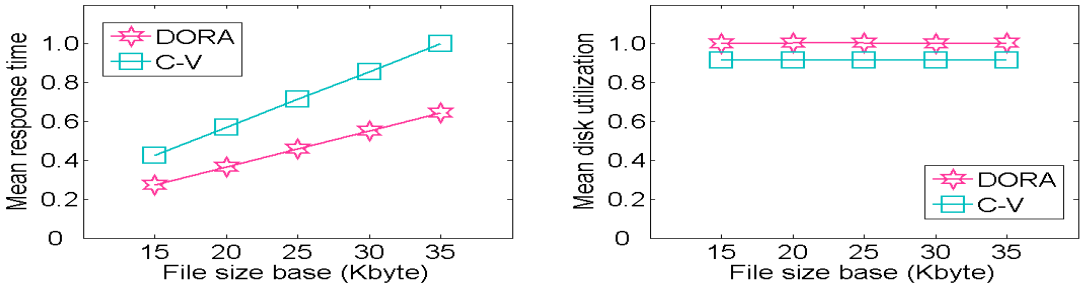


Figure 5. Performance Impact of file size.

algorithms perform considerably better in terms of mean response time (Fig. 5a). This expected since with an increasingly uneven heat distribution due to an enlarged skew degree, both disk cooling and file replication can effectively redistribute heat across disks. Still, DORA outperforms than C-V in both performance metrics.

In particular, when skew degree is 50:50, which means the access requests were evenly distributed across all files without any skew, DORA significantly improves mean response time by 49.5% (Fig. 5a). On the other hand, C-V cannot achieve good performance in this scenario because an even distribution of requests does not give any opportunity for C-V to further even the heat distribution. In terms of mean disk utilization, DORA keeps a constant value of 1 while C-V decreases its mean disk utilization after reaching the highest utilization level of 91.4% when skew degree is 70:30 (Fig. 5b). This is because C-V can no longer effectively even the heat distribution when the workload exhibits an overly skewed heat distribution.

E. Impact of File Size

We examine the performance impact of file size when base file size varies from 15 Kbytes to 35 Kbytes. Note that base file size is the size of the smallest file in a file set *F* and sizes of all other files can be generated based on the inverse Zipf-like distribution. Intuitively, when the size of files is enlarged, mean response time correspondingly increases as well (Fig. 6a). Nevertheless, DORA shows substantially better performance than C-V in all tested scenarios. On average, DORA decreases mean response time by 21.7%. In terms of mean disk utilization, DORA can fully utilize all disks during the simulation period, which results in a higher disk utilization (Fig. 6b).

V. CONCLUSIONS

The file assignment problem, the problem of allocating a set of files onto a disk array so that some cost functions or performance metrics can be optimized, has been studied extensively [2][5][11][15][16][19][20][21]. Conventional dynamic file assignment approaches [2][15][16] solely rely on disk cooling, a file reallocation technique that migrates some hot files from an overheated disk to the disk with least load, to balance the load across disks. However, file reallocation alone cannot realize load balancing in a highly dynamic file system where file popularity can dynamically change. To solve this problem, we developed a dynamic file assignment strategy called DORA (*d*ynamic r*o*und *r*obin with replic*a*tion). DORA integrates file replication techniques into a dynamic file assignment scheme so that load balancing can be achieved in an environment where user access patterns change significantly. Experimental results demonstrate that DORA consistently outperforms C-V, one of the best existing dynamic file assignment algorithms.

In this research DORA only considers read-dominant workload like Web search where read bandwidth is prevailing while write bandwidth is minimal. For example, 99.98% of the total number of operations are read requests in a Web search workload analyzed in [17]. In addition, the size of files in Web search applications tends to be small [1], which limits the overhead of file replication, and thus, makes DORA practical. However, DORA is not likely to work well under write-dominant workload as the overhead of maintaining data consistency among all replicas of a hot file could be prohibitive. Our future work is to extend DORA to write-dominant workloads.

ACKNOWLEDGMENT

This work was supported by the US National Science Foundation under grant number 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] R. Arnan, E. Bachmat, T.K. Lam, and R. Michel, "Dynamic data reallocation in disk arrays," *ACM Trans. on Storage*, Vol. 3, Issue 1, No.2, March 2007.

[3] S. Bucholz and T. Bucholz, "Replica Placement in adaptive content distribution network", *ACM Symp. Applied Computing*, pp.1705-1710, 2004.

[4] C. Cunha, A. Bestavros and M. Crovella, "Characteristics of WWW Client-based Traces," *Technical Report*, 1995-010, Boston University, 1995.

[5] L.W. Dowdy and D.V. Foster, "Comparative Models of the File Assignment Problem," *ACM Computing Surveys*, Vol. 14, No. 2, pp.287-313, 1982.

[6] R.L. Graham, "Bounds on Multiprocessing Timing Anomalies", *SIAM Journal Applied Math*, Vol. 7, No. 2, pp. 416 - 429, 1969.

[7] H. Huang, W. Hung, and K.G. Shin, "FS2: dynamic data replication in free disk space for improving disk performance and energy consumption", *Proc. 12th ACM SOSP*, pp. 263-276, 2005.

[8] M. Karlsson and C. Karamanolis, "Choosing replica placement heuristics for wide-area systems", *Proc. 24th ICDCS*, pp. 350-359, 2004.

[9] T. Kwan, R. Mcgrath, and D. Reed, "Ncsas World Wide Web Server Design and Performance," *Computer*, vol. 28, no. 11, pp. 67 - 74, Nov.1995.

[10] P. Merialdo, P. Atzeni, and G. Mecca, "Design and development of data-intensive web sites: The Araneus approach," *ACM Transactions on Internet Technology*, Vol. 3, Issue 1, pp. 49-92, Feb. 2003.

[11] 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.

[12] T. Loukopoulos, P. Lampsas, and I. Ahmad, "Continuous Replica Placement Schemes in Distributed Systems," *Proc. 19th ACM Annual International Conference on Supercomputing*, pp.284-292, 2005.

[13] J. Ousterhout, H. Da Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson, "A Trace-Driven Analysis of the UNIX 4.2BSD File System," Technical Report CSD-85-230, Univ. of California at Berkeley, 1985.

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

[15] P. Scheuermann, G. Weikum, and P. Zabback, "Dynamic File Allocation in Disk Arrays," *ACM SIGMOD Record*, Vol. 20, Issue 2, pp.406-415, 1991.

[16] P. Scheuermann, G. Weikum, and P. Zabback, "Data Partitioning and Load Balancing in Parallel Disk Systems," Journal of VLDB, pp. 48-66, 1998.

[17] P. G. Sikalinda, L. Walters and P. S. Kritzinger, "A Storage System Workload Analyzer," *Technical Report* CS06-02-00, University of Cape Town, 2006.

[18] S.H. Son and K. Kang, "QoS Management in Web-based Real-Time Data Services", Proc. 4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems, 2002.

[19] P. Triantafillou, S. Christodoulakis, and C. Georgiadis, "Optimal data placement on disks: a comprehensive solution for different technologies," *IEEE Trans. Knowledge Data Eng.*, Vol. 12, pp. 324-330, 2000.

[20] T. Xie, "SOR: A Static File Assignment Strategy Immune to Workload Characteristic Assumptions in Parallel I/O Systems", *Proc. ICPP*, 2007.

[21] 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.

[22] H. Yu, D. Zheng, B.Y. Zhao, W. Zheng, "Understanding user behavior in large-scale video-on-demand systems," *ACM EuroSys*, pp. 333-344, 2006.