DORA: Exploring a Dynamic File Assignment Strategy with Replication

Jonathan Tjioe, Renata Widjaja, and Abraham Lee
Computer Science Department
San Diego State University
San Diego, CA 92182

Abstract

The problem of managing and distributing files to maximize disk performance has been a popular topic of many discussions [1][2][3][4][5]. There are several effective static algorithms that have addressed this issue such as the static round robin (SOR) algorithm. SOR has been proven to produce better response time than other static algorithms such as Greedy, Sort Partition (SP), and Hybrid Partition (HP) [1]. SOR is unique compared to the other static algorithms because it provides considerable performance improvements even if the workload assumption, which says that there is an inverse correlation between file size and its popularity (small files are more popular than large files), does not hold [1]. However, as its name states, it is a static algorithm, and its functionality is limited by the assumption that files and their access patterns do not change over time. In reality, however, this assumption is not accurate for all workloads. We, therefore, propose a new dynamic algorithm called dynamic round robin with replication (DORA). There are two main characteristics of DORA: first, it takes into account the dynamic nature of file or data access patterns to uniquely adapt to changing user demand, and second, it utilizes file replication to further minimize response time and maximize throughput. Moreover, experimental results will show that DORA performs significantly better than another dynamic algorithm, Cool Vanilla (C-V).

1. Introduction

Fast response time is a technology factor that end-users are accustomed to. In a world of distributed applications and web pages that grow increasingly more bandwidth intensive, considerable research has been done to improve methods which can lead to providing instantaneous response to impatient end-users. A good example is a web server application used by an online stock broker company. The Stock Broker site facilitates trades or the buying and selling of shares based on extremely time-sensitive information. In this type of profit-based scenario, response time is of paramount importance since late or untimely information could result in the loss of significant amounts of money. On the other hand, relevant, timely information could bring significant financial gains to the experienced trader.

It can easily be seen that the performance of read-intensive applications such as the above example heavily depend on the underlying parallel I/O systems that enable the ability to serve requests almost instantaneously [1]. Oftentimes, the physical disk is the bottleneck to providing timely response to users' requests. Consequently, minimizing the response time is an important consideration for these large-scale parallel disk systems [1].

On the other hand, the system administrator's main goal is to maximize throughput – the total amount of user requests that can be served over a fixed period of time, or epoch. Throughput and response time are often closely related since in most cases, by minimizing the response time for individual requests, the overall throughput will be maximized [12].

As a result, much of today's research centers on file management efficiency and disk scheduling algorithms. Some examples of these research areas are in the realm of RAID architectures that focus on data striping, data replication, and data mirroring to achieve high data throughput and high data reliability [6][7]. Substantial research has also been done to reduce disk head latencies associated with moving the head to the appropriate physical location on a disk [2][3]. Once physical limitations of the hardware have been met, it is left up to the software architect to

accurately and efficiently manage available resources to fully exploit the capabilities of the parallel disk system. Furthermore, file assignment is the process of arranging all the files onto an array of disks structured in such a way as to significantly improve the overall performance of a parallel I/O system [1]. The file assignment problem (FAP) has been researched exhaustively in literature [5][12]. At its essence, the FAP attempts to organize M files onto N disks in a structure that minimizes specific cost functions such as bandwidth costs, storage costs, and queuing costs [1][5]. Additionally, the FAP attempts to optimize performance metrics such as mean response time and overall system throughput [1][5]. Finding the optimal solution for file assignment for a cost function or a performance metric on a parallel disk system is an NP-complete problem [1][12]. As a result, research for solving the FAP has migrated towards heuristic algorithms.

Typically, there are two classes of heuristic algorithms: static and dynamic [1]. File assignment algorithms that are static in nature require complete knowledge of workload characteristics in advance. This includes metrics such as service times, wait times, and arrival rates of requests for each file. Files are arranged onto a set of disks one time and user requests to those files are subjected to the same static file assignment for the duration of the epoch. Static algorithms fare very well when the workload assumption holds – when there is an inverse correlation between file size and its popularity. However, in certain applications, the workload assumption does not necessarily hold [14]. In these applications, static algorithms suffer considerable degradation in terms of performance. In an attempt to supplement existing static algorithms, a new static algorithm named SOR was designed. SOR is a static file assignment algorithm that proved to be a significant improvement to several existing static algorithms such as HP, Greedy, and SP [1]. SOR distinguished itself from the others by providing significant improvements in performance even when the workload assumption did not hold. Although SOR impressively beat out existing static algorithms, SOR – as with its predecessors – suffered as demand for the files changed over time. Dynamic file assignment algorithms, on the other hand, do not require prior knowledge of workload characteristics. Dynamic algorithms are able to keep track of the heat of each file – as well as the load on each disk – for every epoch. As user demand for those files change, these algorithms can dynamically reorganize the layout of files in an effort to minimize response time and maximize throughput. Dynamic algorithms are superior to static algorithms in that they are able to evolve with changing user demand. As an example, consider the online streaming video website www.youtube.com. Videos on this popular site usually experience the highest demand and generate considerable amounts of traffic during the first several weeks of being posted. During this period of high traffic, dynamic algorithms can assess the user demand and make copies of the hot files to several other disks and load balance the incoming requests to several other high speed disks or physical servers. As a result, this can lower the response time for user requests. Moreover, once these videos are no longer popular, the replicas can be deleted from the servers and only one copy of the original file kept in order to satisfy the occasional request for the video. It is self-evident that dynamic algorithms are more suitable for real-world applications where comprehensive workload characteristics are not usually known ahead of time and user demand evolves over time.

In this paper, we propose a new dynamic algorithm called DORA which combines the best features of several algorithms. First, DORA takes from the philosophy of SP in that it sorts all files according to file size so as to take advantage of the improvements achieved when the workload assumption holds true. Inheriting from SOR, DORA assigns the files to disks in a round-robin fashion so as to distribute the heat of all files evenly across all disks. Thus, DORA also will achieve performance improvements when the workload assumption does not hold true. Finally, DORA is a dynamic algorithm that keeps track of the heat of all files and the load of all disks. It then creates replicas of extremely hot files and effectively load balances the requests for these files across all disks. Consequently, this allows DORA to provide performance improvements as the demand for files change over time.

To prove the merit of DORA over other algorithms, we will compare it against a dynamic algorithm, Cool Vanilla (C-V), which will be discussed in greater detail in the Related Work section. Inheriting from SOR, DORA provides performance improvements regardless of the workload assumption. In addition, results will show that DORA continues to perform well in the face of changing user demand.

To achieve these goals, performance will first be measured for C-V and DORA when the workload assumption holds. Next, results will be compared between the two algorithms to show that DORA's method for replication of extremely hot files provides performance improvements over C-V.

The rest of this paper is organized as follows. Section 2 provides a brief summary of the dynamic algorithm C-V. Section 3 discusses the DORA

algorithm in detail. In addition, the issues that arise with processing write requests in a replicated environment are addressed as well. Section 4 presents the experimental results for both DORA and C-V. The performance metrics and the parameters used to generate the synthetic workload are explained. Finally, Section 5 provides conclusive arguments and supporting data.

2. Related Work

Several dynamic algorithms have proposed noteworthy solutions to solve the FAP. One example, C-V, uses file relocation and disk selection in order to balance the heat on the disks.

C-V is also known as Greedy Algorithm with disk cooling. In addition to using the Greedy file assignment algorithm, C-V adds the ability to perform dynamic load balancing to keep the disks "cool". As the heat for previously allocated files change, C-V reorganizes files in such a way as to balance out the new heat across all disks. More precisely, when the disk cooling method is invoked, C-V will relocate the hottest files from an overheated disk to the coolest disk [5].

C-V provides the ability for disk cooling, where an overheated disk can offload some of its files to a cooler disk. This is beneficial only when the overall load on the system decreases. As the overall load on the system increases, the performance of C-V will suffer as each extremely hot file's wait time will continue to increase. So C-V is downward scalable, but not upward scalable. DORA, on the other hand, is upwards scalable and has limited downward scalability. Since DORA has the ability to create replicas, the heat of an extremely hot file can be distributed across multiple disks. DORA provides moderate downward scalability since replicas can be deleted when they are no longer needed.

3. The DORA algorithm

In its most general form, a parallel disk system can be represented by a linked group D of independent homogeneous disk drives: $D = \{d_1, ..., d_j, ..., d_n\}$. The set of files that will be placed onto D can be represented by $F = \{f_1, ..., f_i, ..., f_m\}$. A disk d_j can be represented by $d_j = (c_j, t_j, l_j)$, where c_j is the capacity in GByte, t_j is the transfer rate in MByte/second, and l_j is the load (total sum of the files' heat on the disk). The assumption is disk capacities are large enough to store all of the files. When considering the proliferation of server class drives with large capacities (i.e. 1 TeraByte), this is a reasonable assumption. A file f_i consists of four attributes and can be modeled as $f_i = (s_i, \lambda_i, t_i, h_i)$, where s_i is the size of the file, λ_i is the mean arrival rate of requests to a file, t_i is the expected service time, and h_i is the heat of the file. For this research, λ_i is the mean arrival rate of requests coming in for a file f_i. Disk accesses to a file exhibit a Poisson distribution with a mean arrival rate λ_i. A fixed service time t_i is assumed for each request r_k. This is a reasonable assumption since – for web server applications – the majority of all file accesses are sequential reads that read the entire file from beginning to end. In addition, when accessing large files, the overhead associated with rotational latencies involved in moving to the appropriate section of the hard disk is negligible since we assume a sequential read of the entire file. Consequently, if f_i is a file allocated to d_j, it follows that $t_i = s_i / t_j$. The load of a *file i* can be calculated from the product of the mean access arrival rate for a file and the expected service time for that file. Thus, the *heat* of the file can be represented by:

$$h_i = \lambda_i \cdot t_i \tag{1}$$

It also follows that the average disk load or *heat* can be represented by:

$$\rho = \frac{1}{n}\sum_{i=1}^{m} h_i \tag{2}$$

File assignment algorithms organize groups of files onto sets of homogeneous disks in order to reduce mean response time. Typically, incoming requests are served using a First-In-First-Out (FIFO) or First-Come-First-Serve (FCFS) scheduling heuristic. A request set R contains the total number of requests u and can be modeled as $R = \{r_1, ..., r_k, ..., r_u\}$. Each request can be represented by the equation $r_k = (fid_k, a_k, type_k)$, where fid_k, a_k, $type_k$ are the file identifier targeted by the request, the arrival time of the request, and the type of request (read or write). For this research, 99% of all requests generated will be reads since this is consistent with data obtained from web server traces [16]. When an incoming request arrives, the FCFS scheduler finds what disk the target file is located on. The request is then directed to the disk's local scheduling queue.

Two important parameters are the start time and the finish time of a request r_k on a disk d_j. Start time and finish time are represented by $st_j(r_k)$ and $ft_j(r_k)$, respectively. In order to get the response time of a request r_k, the start time and finish time must be calculated. Both will be derived below. There are three cases for when a request r_k arrives in Q_j, the local queue of disk d_j where ($1 \leq j \leq n$). The first is when d_j is idle and Q_j is empty. The second is when

d_j is busy and Q_j is empty. The third is when d_j is busy and Q_j is not empty.

$$st_j(r_k) = \begin{cases} a_k, \text{ if } d_j \text{ is idle and } Q_j \text{ is empty} \\ a_k + r_j, \text{ if } d_j \text{ is busy and } Q_j \text{ is empty} \\ a_k + r_j + \sum_{r_p \in Q_j, a_p \le a_k} t_{fid_p} \quad \text{otherwise} \end{cases} \tag{3}$$

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 the $ft_j(r_k)$ can be represented by

$$ft_j(r_k) = st_j(r_k) + t_{fid_k} \tag{4}$$

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 be calculated by

$$rt_j(r_k) = ft_j(r_k) - st_j(r_k) \tag{5}$$

The mean response time for the request set R can be calculated by

$$mrt(R) = \frac{1}{u} \sum_{k=1,1 \le j \le n}^{u} rt_j(r_k) \tag{6}$$

The FAP can now be generalized as having a set of files F and a parallel disk system D. Given F and D, find an allocation scheme X such that the mean response time expressed by Eq. 6 is minimized.

Fig. 1 shows a logical system diagram of how the DORA algorithm functions.

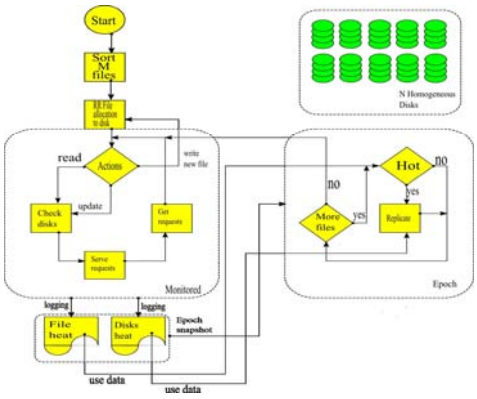

Fig. 1. System Model.

Fig. 2 lists the abbreviated pseudo code for DORA and provides some detailed explanations.

```
Input: A parallel I/O system D with n identical disks, a collection of m files in a
queue F, Epoch number EPOCH_NUM
Ouput: A file allocation matrix X(n,m) and performance matrix
%%%%%Run SOR algorithm to allocate files to each disk%%%
EPOCH_COUNTER = 0
while EPOCH_COUNTER < EPOCH_NUM

avg hot files heat = sum(all hot files' heat) / # of hot files
HEAT_MAX = avg hot files heat/2;
HEAT_MIN = aavg hot files heat/ 4;
MAX_REP = DISK_NUM-2;
HOT_FILES = 20;

if epoch_counter > 0

%%%%%Process User Requests%%%%%

%%%%% REPLICATION MODULE %%%%%%
fi = 1
rep_count = 0
while fi <= FILE_NUM && hot_files_count < HOT_FILES %only replicating up
to 20 hottest file

  if heat(fi) >= HEAT_MAX
    n_replica = floor(2*heat(fi) / avg heat of hot files)
    if n_replica > MAX_REP  %maximum number of replicas for each file
       n_replica = MAX_REP
    end
    Check Replication_matrix if fi is already replicated

    If YES
       1.count how many more replicas needed, create, and place each of them on
the coolest disk dc.
       2. Update load on each disk dc
    ELSE
      1. Create n_replica for fi, place each of them on the coolest disk dc.
      2. Update load on each disk dc
    end if
  end if
  fi++
  hot_files_count++
end while
%%%%% END REPLICATION MODULE %%%%%%
Generate new heat for each file
%%%%% GARBAGE COLLECTION MODULE %%%%%%
For every file fi in the Replication matrix
  If heat(fi) < HEAT_MIN
    1. Delete each replicas ri of fi from each corresponding disk dc
    2. Update the load on each disk dc
  end
end for
```

Fig. 2. DORA Pseudocode.

Initial allocation of files. The distribution of the workload among all disks and the minimization of the variance of service time are both important principles used by other algorithms such as SP, HP, Greedy, and SOR [1][5]. DORA also takes these principles into account. The average disk load ρ is computed and the load for each disk d_j must not exceed ρ. Similar to SP, HP, and SOR, the file set F is sorted by file size since the service time is proportional to file size. This effectively places files of similar size onto the same disk [5]. SOR separates the most popular files onto different disks in a round-robin fashion instead of a consecutive allocation of the sorted file set as with SP [1]. The last disk is used for very large files only. It will not be used for replication since the overhead of replicating a large file is very high. By limiting very large files to the last disk, DORA prevents the problem of the disk blocking all other requests until the completion of the large file.

Replication of extremely hot files. When considering SP, HP, Greedy, and SOR, they all have similar drawbacks. File assignment algorithms that are static in nature require future knowledge of all

workload characteristics. As a result, static algorithms are not suitable for many real-world applications as future knowledge of workload characteristics are not usually known. Static algorithms also suffer from the fact that they cannot vary with evolving user demand or access arrival rates. DORA does not require future knowledge of workload characteristics and can reconfigure file assignment as the access patterns change over time. Consequently, DORA is better suited for real world applications.

Several challenges arise from the dynamic nature of DORA and from the use of file replication. Records of all replicated files must be kept which include information about the file such as the file identifier, how many replicas are in circulation, and what disks the replicas reside on. Selecting the destination drive and the number of copies remains one of the main focuses of many file management algorithms that use replication. The choice of heuristics significantly affects performance [9].

Creating replicas. While load balancing can be performed effectively by making copies of hot files to multiple disks, creating the replicated files on multiple disks will introduce additional overheads. Cost versus Benefit trade offs must be examined before replicating a file. It is also worth noting that replicas will be created on N-2 disks. Similarly to SOR, DORA reserves one disk for very large files. This further takes advantage of the minimization of service time principle by isolating very large files to the last disk [5]. The other reserved disk contains the original file and should not be replicated to as there is no benefit in having two copies of the same file on the same disk. Several experiments were run to determine exactly when replicas should be created for a file and also how many replicas should be created for a file. When considering the results, empirical data showed that replicas should be created when the heat of a file was greater than half the average heat of all of the hot files, where *HOT_FILES* is an input parameter of the DORA simulator. This provides a HEAT_MAX threshold that can be represented by the following equation:

$$h_i \geq avg\_hot\_files\_heat / 2 \qquad (7)$$

where

$$avg\_hot\_files\_heat = \frac{1}{HOT\_FILES} \sum_{i=1}^{HOT\_FILES} h_i \cdot \qquad (8)$$

If the condition in Eq. 7 is true for a file f_i, then the file is considered to be extremely hot and replicas will be created. The amount of replicas that will be created is also dependent upon the average heat for all hot files *avg_hot_files_heat*. To obtain the number of replicas, two times the heat h_i of each extremely hot file will be divided by the *avg_hot_files_heat*. This quotient will be rounded down, resulting in the number of replicas to be created. This can be represented by the following equation:

$$\#replicas = floor(2 * h_i / avg\_hot\_files\_heat) \qquad (9)$$

This algorithm effectively normalizes the heat corresponding to the extremely hot files. Fig. 3 shows the original Zipfian heat distribution of 40 files without the use of replication. From Eq. 8, the *avg_hot_files_heat* value is computed to be .3696. Combining with Eq. 7, this results in the first 3 files being replicated. According to Eq. 9, files 1, 2, and 3 will each have 3 replicas created.

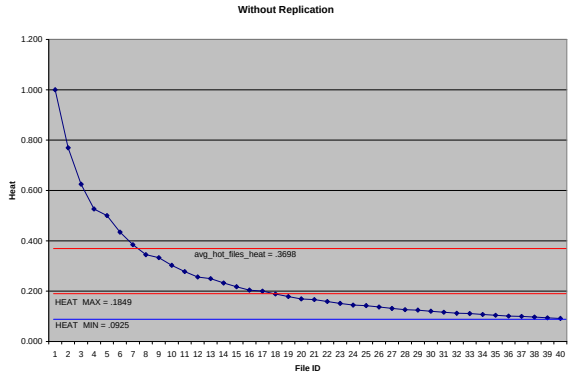

Fig. 3. Heat Distribution without Replication.

Fig. 4 shows the new heat distribution that has been normalized by using replication for the extremely hot files. Note that the distribution for the replicated files' heat is now closer to *avg_hot_files_heat* since, for each file, the original heat has been divided by the number of replicas plus 1. Consequently, a single disk d_j is relieved of having to process every request r_k that targets a file f_i that resides on that disk since now f_i is stored on multiple disks. This allows for a certain degree of parallel execution.

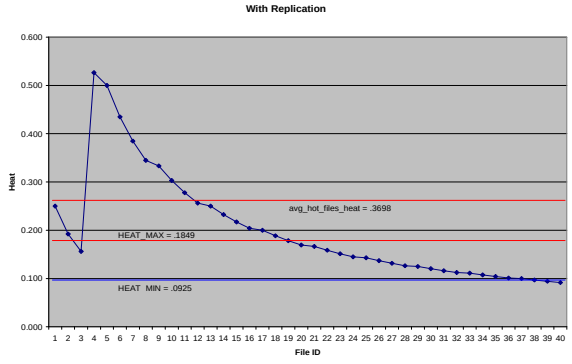

Fig. 4. Heat Distribution with Replication.

Updating replicas. Another improvement that DORA adds from other static algorithms is the ability to handle *write* requests. As a result, data consistency issues must be handled. For example, if an incoming request r_k is a write request that updates a replicated file, all replicas and the original file must be updated to ensure the latest copy of the file is being used to serve user requests. When a write request that targets a replicated file comes into the FCFS scheduler, DORA immediately queues the write request to the original file and also to all replicated files to ensure consistency. This prevents all Read-After-Write (RAW) data hazards.

Deleting replicas. File access patterns and user demand for files will change overtime, which eventually will result in files that were formerly extremely hot wasting disk space. These *cold* replicas must be removed, while leaving the original file intact. This requires a garbage collection mechanism to conserve space and update the heat on each disk. Empirical data showed that a replica for an extremely hot file should be deleted once h_i for f_i became less than or equal to the average heat of all of the hot files divided by 4. This provides a *HEAT_MIN* threshold that can be expressed by:

$$h_i \leq avg\_hot\_files\_heat / 4 \qquad (10)$$

Once Eq. 10 is true for a file that has previously been replicated, all replicas will be deleted and the heat will be updated on each disk to reflect this change.

4. Performance evaluation

In this section, we evaluate the performance of both C-V and DORA using a synthetic, yet realistic workload. Initially, we will present data that compares the response improvement of DORA with and without replication to illustrate the benefits of using replication under heavy workloads. Still more notable, DORA will be compared with another dynamic algorithm, C-V, which uses the disk cooling technique. Results will clearly show that DORA performs significantly better than C-V, especially under heavy workload conditions.

4.1 Simulation setup

Matlab software was used for all simulations. In addition to DORA, Cool Vanila (C-V) will also be simulated for benchmark comparison purposes. The A disk array of *DISK_NUM* homogeneous Cheetah ST39205LC hard disks were modeled as a parallel I/O system. However, the disks were not simulated in a RAID configuration. Non-partitioned files of varying sizes and loads corresponding to variable heat distributions were used to show that DORA performs significantly better than other dynamic algorithms such as C-V by adding replication for extremely hot files, allowing for the ability to dynamically recalibrate file assignment according to changing user demand. Since web page requests typically follow a Zipfian distribution [13][16], for this paper, the initial file-disk allocation is modeled as such. In addition, file requests are modeled with Poisson arrival rates with fixed service times.

Mean response time: average response time for all file access requests that are sent to the simulated parallel disk system.

Mean response time improvement: decrease in seconds of mean response time gained by DORA compared with C-V.

Mean disk utilization: the average ratio between a disk's total service time and its total operation time. The time between the arrival time of the first request and the finish time of the last request is the total operation time.

Table 1 summarizes the configuration parameters of the synthetic workload and the modeled system used for our experiments. Despite the fact that the workload was synthetically generated, all parameters were carefully controlled in order to model the workload as accurately as possible.

Table 1. System parameters.

Parameter	Value (Fixed) – (Varied)
Number of files	(5000)
File load (heat)	Each file has heat defined as $h_i = \lambda_i * t_i$
Coverage of the file system	(100%) – each file is accessed at least once
Number of disks	(20) – (8, 12, 16, 20, 24)
Aggregate access (1/second) rate	(200) – (25, 50, 100, 200, 300)

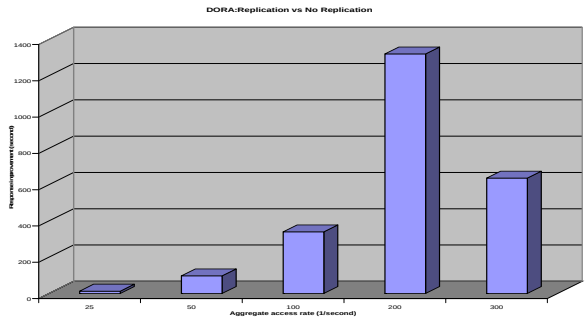

Fig. 5. DORA:
Replication vs. No Replication.

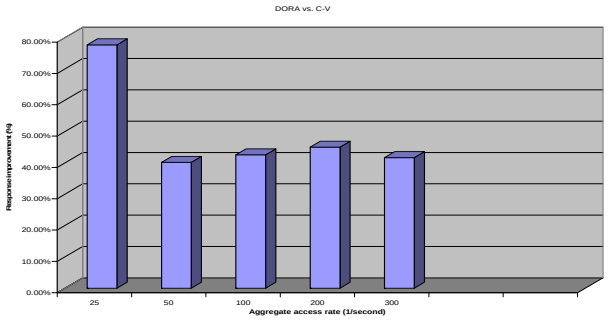

Fig. 6. DORA vs. CV:
Response Improvement (%).

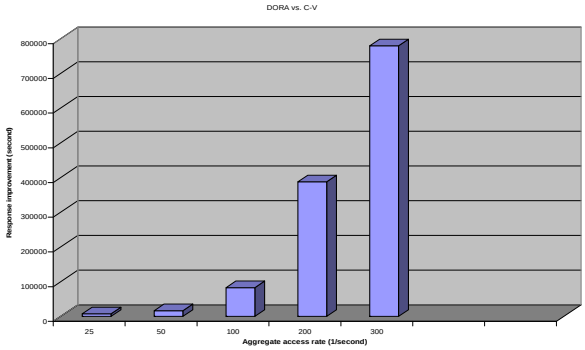

Fig. 7. DORA vs. CV:
Response Improvement (seconds).

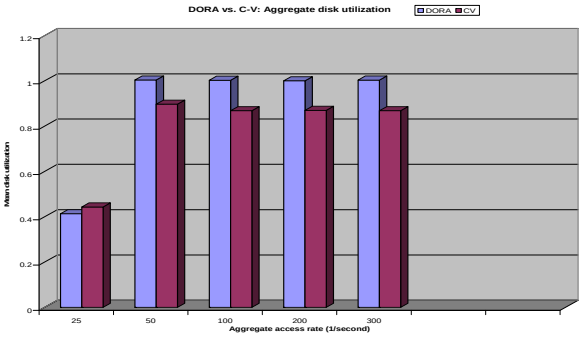

Fig. 8. DORA vs. CV:
Mean disk utilization.

4.2 Impact of aggregate access rate

Figures 4-8 are based on a Zipfian file distribution where the workload assumption holds. The workload assumption states that the popularity, or *heat*, of a file is inversely proportional to its size. The arrival rates are modeled after a Poisson distribution.

Fig. 5 shows a comparison of DORA with and without Replication. Similar to the fact that C-V is essentially the Greedy algorithm with disk cooling, DORA is SOR extended to be a dynamic algorithm that uses replication. The results in Fig. 5 show an increase in response improvement as the aggregate access rate λ increases. This is expected as replication is most beneficial for extremely heavy workloads. In these cases, the wait time for an extremely hot file is minimized since subsequent requests do not have to be scheduled onto a single disk, but can instead be scheduled onto the original disk or any of the replicated disks. A λ of 200 yields the best results.

Fig. 6 illustrates the response improvement percentage of DORA over the C-V algorithm. The results show that the greatest improvement of response time occurs initially at λ = 25 (77.71%), decreases sharply at λ = 50 (40.17%), and then remains relatively stable for higher aggregate access rates. The sharp decrease is due mainly because the total amount of requests are much larger after λ = 25. Since this graph shows percentage and the total amount of requests is much higher, the percentage should decrease. Fig. 7 shows the same dataset displayed in seconds. The results are consistent with Fig. 6 since even though Fig. 7 shows the highest response improvement in seconds at λ = 300 (779,396 seconds), the total response time at this rate is very high. Consequently, Fig. 6 illustrates a more conclusive representation.

In Fig. 8, we see that the aggregate disk utilization for DORA is noticeably higher than C-V's for larger values of λ. This is expected since DORA's use of replication will redistribute the heat of an extremely hot file to multiple replicated disks, which effectively load balances the disk array. By using replication, DORA will minimize the disk idle time of the disks, thereby increasing the total disk utilization. Since C-V is only able to scale downward (using disk cooling), and DORA is able to scale upward, the results shown correspond to the original assumption that replication can be used to minimize response time by load balancing across multiple disks, thereby increasing overall disk utilization.

4.3 Scalability

From the data, it can be concluded that DORA is much more scalable than C-V. C-V uses a technique known as disk cooling. A parallel system using C-V is able to move hot files from an overheated disk to the disk that contains the least heat. This disk cooling technique gives C-V the ability to be downwards scalable. More precisely, if the overall load of a disk decreases dramatically, files from the hottest disk can be moved to the coolest disk. However, C-V does not have a mechanism to enable it to be upwards scalable. In contrast, DORA's use of replication enables it to scale upwards. For example, when using C-V, if several requests arrive in close time proximity for the same file, the FCFS will schedule the first request to the appropriate disk. It follows that all subsequent requests will have to be queued behind the original request, thereby increasing the wait time and heat on a particular disk. DORA, on the other hand, will record the heat generated by the numerous requests during the epoch and will then create replicas of the file appropriately. During the next epoch, the benefit of replication will be realized as multiple requests for the same file that have very close arrival times can be distributed across several disks. This allows DORA to be extremely upwards scalable. DORA also provides the ability to have limited downwards scalability. For instance, for a replicated file, once the heat of the file falls below the designated *HEAT_MIN* threshold, the replicas of that file will be deleted, thereby conserving space on each disk.

5. Conclusions

Fast response time is a technology factor that end-users demand. Considerable research has been dedicated to find more efficient ways to arrange data such that fast response time can be achieved [1][4][5]. SOR effectively increased file search efficiencies over other published static algorithms, yet SOR, being static by nature, is not able to dynamically re-allocate files based on changing user demand. We proposed a new dynamic algorithm DORA that is able to use replication to adjust to varying user demand. DORA is better suited for a realistic environment where files' popularity is constantly changing.

Experimental results showed that DORA effectively reduced mean response time with the use of replication for extremely hot files. Intuitively, it follows that overall system throughput would also be reduced since the wait times for subsequent requests for the same file were minimized. DORA also proved to be far more scalable than C-V. It was able to use replication to effectively load balance by redistributing the heat to multiple disks that were used in replication. The benefit of replication becomes evident under extremely heavy workloads when the aggregate access rate is high. Thus, DORA's ability to evolve with changing user demand and the ability to use replication to load balance across multiple disks proves that it is suitable for many real world applications where the optimization of mean response time is critical.

References

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

[2] 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.

[3] W.W. Hsu, A.J. Smith, and H.C. Young, "The automatic improvement of locality in storage systems", *ACM Transactions on Computer Systems*, Vol. 23, Issue 4, pp. 424- 473, 2005.

[4] 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, Issue. 2, pp. 324 - 330, 2000.

[5] Lin-Wen Lee, Peter Scheuermann, and Radek Vingralek, "File Assignment in Parallel I/O Systems with Minimal Variance of Service Time", *IEEE TRANSACTIONS ON COMPUTERS*, VOL. 49, NO. 2,, pp. 127 -140, 2000.

[6] P.M. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, and D.A. Patterson, "RAID: High-Performance, Reliable Secondary Storage", *ACM Computing Surveys*, Vol. 26, No.2, pp. 145 -185, 1994.

[7] S.H. Lim, Y.W. Jeong, and K.H. Park, "Interactive Media Server with Media Synchronized RAID Storage System", *ACM*, NOSSDAV '05, June 2005.

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

[9] M. Karlsson and C. Karamanolis, "Choosing replica placement heuristics for wide-area systems", *Proc. 24th Int'l Conf. Distributed Computing Systems*, pp. 350 - 359, 2004.

[10] T. Loukopoulos and I. Ahmad, “Static and adaptive data replication algorithms for fast information access in large distributed systems”, *Proc. ICDCS*, pp. 385 - 392, April 2000.

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

[12] Lawrence W. Dowdy and Derrell V. Foster “Comparative Models of the File Assignment Problem”, *ACM Computing Surveys*, Vol 14, No. 2, pp.287-313, 1982.

[13] Kairui Chen, Hui-Chuan Chen, Richard Borie, and Jonathan C.L.Liu, “File Replication in Video on Demand Services”, *ACM-SE 43*, Vol 1, No. 2, pp.162-167, 2005.

[14] S.Bucholz and T.Bucholz, “Replica Placement in adaptive content distribution networkd”, *ACM Symp. Applied Computing*, pp.1705-1710, 2004.

[15] ThanasisLoukopoulos, Petros Lampsas, and Ishfaq Ahmad, “Continous Replica Placement Schemes in Distributed Systems”, *ACM ICS 2005*, pp.284-292, 2005.

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

[17] S. Glassman, “A caching relay for the World Wide Web,” *First conf. World-Wide Web*, pp. 165 - 173, 1994.