Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

CHAINED DECLUSTERING: LOAD BALANCING AND ROBUSTNESS TO SKEW AND FAILURES

Leana Golubchik
John C.S. Lui
Richard R. Muntz

July 1991
CSD-910055

Chained Declustering: Load Balancing and Robustness to Skew and Failures

Leana Golubchik John C.S. Lui
Richard R. Muntz

UCLA Computer Science Department

July 19, 1991

Abstract

The objective of this paper is to investigate the degree to which a dynamic load balancing disk scheduling algorithm in conjunction with chained declustering can respond robustly to variations in workload and disk failures (which destroy the symmetry of the system and introduce skewed load). Specifically, we define and investigate the behavior of two dynamic scheduling algorithms under various workload distributions and disk failure. We demonstrate that using these two simple dynamic scheduling algorithms, longest queue first (LQF) and majorization (MAJ), has greatly improved the average response time compared with static load balancing.

1 Introduction

In the past decade, we have observed an unprecedented growth in processor's speed, main memory and secondary storage capacity [11]. Yet, the improvement in performance of magnetic storage systems has been modest compared with other computer system components. In recent years, there has been considerable research concerning the use of arrays of disks in solving this emerging I/O bottleneck problem. The idea of disk arrays is to utilize a set of disks and conceptually present the abstraction of a high capacity, high throughput, and highly reliable logical disk. In order to make data highly available, some form of data redundancy is introduced. Two basic schemes for introducing data redundancy are (1) parity based schemes and, (2) mirroring.

In a parity based scheme a parity block is associated with several data blocks. Whenever a disk fails, a data block on the failed disk can be reconstructed by reading the corresponding parity and data blocks. Examples of parity based schemes include RAID level 5 [11], clustered RAIDs [10] and various parity striping schemes [6]. Full mirroring has a higher storage overhead than the parity based schemes because data is fully duplicated, but it can offer better performance in terms of throughput and response time [6] than the parity based schemes. Mirroring or shadow disks [1] typically replicate complete disks, i.e., one disk is replicated on a second. The disk farm is composed of a number of such pairs. In [3], the authors illustrate the idea of optimizing seek times in a mirrored disks environment. Disk scheduling policies for mirrored disks systems have been studied in [12], and disk scheduling policies with real-time applications using mirrored disks have been studied in [4]. In [7], chained declustering is considered as a replication scheme at the logical level of a shared nothing database machine. This scheme provides an alternative to the classical mirroring scheme when applied to physical level replication. We briefly describe the concept of chained declustering in [7], where it is applied at the logical level of a shared nothing database machine. Later, we apply the same concept to physical level replication.

Chained declustering has the same storage overhead compared to the classic mirroring scheme, but it offers better performance degradation properties when a single disk failure occurs. Figure 1 illustrates the chained declustering concept. Assume a file R is declustered into 8 fragments. At any point in time, two physical copies of this file, termed the primary copy and the backup copy, are maintained. During the normal mode of operation, read requests are directed to the primary copy and write operations update both copies. When a disk failure occurs (e.g. disk 1 in the figure), the chained declustering scheme is able to uniformly distribute the workload over the surviving disks and thereby obtain a less degraded performance. For example, if disk 1 has failed, 2/7 of the transactions to $R3$ can be routed to Disk 3 and 5/7 of the transactions to $R3$ can be routed to Disk 4. The idea is to adjust the load to both copies of the data in such a way as to balance the load among the surviving disks.

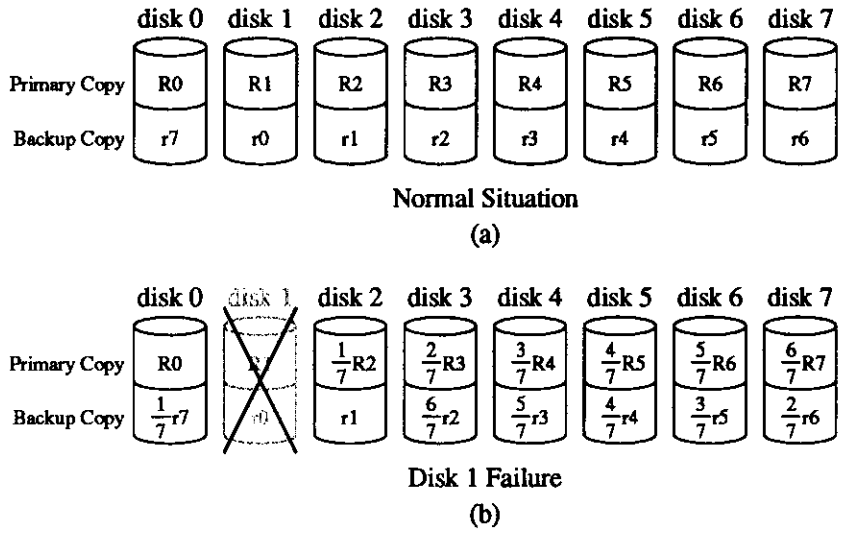

Figure 1: Chained Declustering Scheme

There are several ways to perform the load adjustment depending on table declustering methods, storage organization, and access plans. Since data is logically replicated, the query scheduler makes the decision on the access plan in order to balance the load. This form of load balancing has several limitations (1) the load is only approximately divided among the nodes; the assumption that a uniform division of the data corresponds to a uniform division of the load can be incorrect with skewed reference patterns, and (2) both short term and long term reference patterns change with time and a static balancing scheme can not adjust to variations in load. Another way to balance the load of the system is to apply some *dynamic* load balancing scheme, since it can adjust the load on each node in real time to respond to statistical variations. Several dynamic balancing schemes are discussed in [12], in the context of mirrored disks systems.

The objective of this paper is to investigate the degree to which a dynamic load balancing disk scheduling algorithm in conjunction with chained declustering can respond robustly to variations in workload and disk failures (which destroy the symmetry of the system and introduce skewed load). Specifically, we define and investigate the behavior of two dynamic scheduling algorithms under various workload distributions and disk failure. We demonstrate that using these two simple dynamic scheduling algorithms has greatly improved the average response time compared with static load balancing.

The paper is organized as follows. In section 2, we discuss the difference between logical and physical data replication. In section 3, we present two dynamic disk scheduling algorithms. We also show the improvement in response time compared with the static load balancing algorithm under different workload distributions and disk failure conditions. Section 4 summarizes the contributions of the paper.

2 Physical vs. Logical Replication

In general, data replication can be implemented on different levels within a database system. In particular, we distinguish between (1) physical replication and (2) logical replication.

With physical level replication the contents of one area of a disk are mirrored on an area of another disk (in the classical mirrored disk system, one entire disk is mirrored by another entire disk). The I/O controller generally handles the replication and higher levels of software are not concerned. With logical fragmentation as in the Teradata [2] and Gamma [5] shared nothing database machines, relations are fragmented and relation fragments are stored on independent nodes of the system. Replication is visible to the query processing software and is managed by the database system software.

The dynamic scheduling algorithms studied here can be applied to both physical and logical replication methods. There are significant problems associated with dynamic data sharing across multiple nodes of a system, e.g., concurrency control, and efficient use of buffer space [14, 13]. We do not address these problems here. With respect to logical replication one can view this study as an investigation of the *potential* benefits of dynamic load balancing with chained declustering, particularly with respect to robustness to workload imbalance and disk failure. Determining whether these benefits compensate for the overhead and complexity of logical level dynamic scheduling is beyond the scope of this paper. In the remainder of this paper we will concentrate on physical replication.

3 Analysis

In this section we will describe our model of the disk subsystem, state the algorithms for dynamic load balancing, present the results of analysis of these algorithms using a variety of workloads, as well as show the improvements in performance gained through dynamic scheduling.

3.1 System Description

As mentioned earlier, the physical system of interest is a disk subsystem with data replicated using the chained declustering scheme [7] at the physical level (refer to sections 1 and 2). In such a system, the data residing on disk i is backed up partly on disk $i+1$ and partly on disk $i-1$[1]. The replication not only provides higher data availability, but can also be used to improve performance, since it provides flexibility in servicing read requests[2].

[1] For the remainder of the paper, all arithmetic is done modulo D (the number of disks in the system).

[2] Each write request must be done on both disks, in order to maintain consistency between the two copies.

We can potentially improve performance by properly choosing from which copy to read. For example, consider the system in Figure 2; we will call each portion of data which is replicated a "fragment". The data fragment A is stored on disk 0 and disk 1. A write of a block of

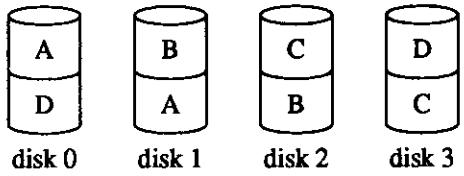

Figure 2: Chained Declustering Data Configuration Scheme

data from A must be executed on both disks 0 and 1. However, a read request for the same block can be serviced by either disk. The flexibility to read from either disk is the source of improvement possible via dynamic scheduling. Below, we first introduce our model of a disk system using chained declustering and the two dynamic scheduling algorithms.

3.2 Model Description

In our model, the I/O requests are separated into queues based on the data fragments they reference. Within each queue, requests are served in a first come first serve order. The disks can be viewed as servers in our model. Figure 3 illustrates the model of the physical system depicted in Figure 2. It has four queues, with queue 0 representing requests for data blocks in fragment A, queue 1 representing requests for data blocks in fragment B, etc. There are also four servers, each representing one of the disks in the original system. Each server can serve only *two* of the queues, i.e., each I/O request can be performed on either of the two disks[3] (refer to Figure 3). When an I/O subsystem is heavily utilized, especially when it is the

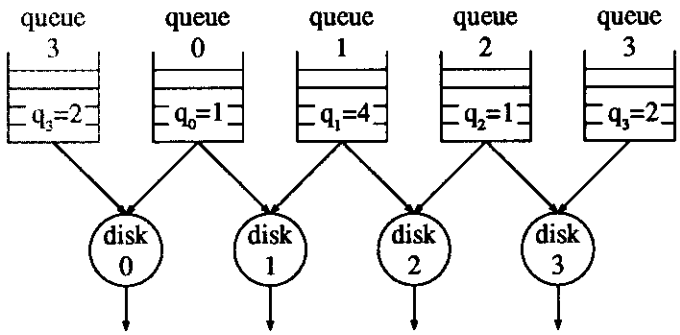

Figure 3: Model Terminology

bottleneck, this choice can affect the overall system performance. The basic goal is to avoid

[3]This is true for reads only. In the remainder of the paper a pure read model is assumed, except where otherwise specified.

having requests queued for some servers, while other servers are idle. A good scheduling scheme can decrease the amount of time spent in this state by, for example, balancing the workload among the servers. We will show that an adaptive (*dynamic*) scheduling scheme performs significantly better than a non-adaptive (*static*) one, especially when there are "hot spots" in the database or when failure occurs.

Our model parameters are as follows (refer to Figure 3):

D	=	number of disks
q_i	=	queue lengths of queue i
L_i	=	sum of queue lengths at queues i and $i-1$
λ_i	=	arrival rate to queue i
μ_i	=	service rate of server/disk i

In the following sections we will consider homogeneous servers, exponential arrival rates, and exponential service rates (this is done for ease of presentation; similar results can be show for other types of arrival and service distributions). Both uniform and skewed workloads will be considered, as well as systems with failures.

Given this model, our goal is to investigate the benefits associated with *dynamic* scheduling of I/O requests. Specifically, since a read request can be performed on one of two disks, the performance of the disk subsystem will improve or degrade depending on how we make this choice. Hence, a scheduling algorithm is characterized by how it determines *which customer should a disk serve next.*

3.3 Dynamic Scheduling Algorithms

In this section we will discuss two dynamic scheduling algorithms. These algorithms will differ in the type of state information they use and the complexity of the decision process.

Our first algorithm, *Longest Queue First* (LQF), uses the set of queue lengths, i.e., $(q_0, q_1, \ldots, q_{D-1})$, as its state information. In this case the decision process is simple. When a disk becomes available for service, it chooses the first customer from the longest of the two queues it is able to service. LQF is perhaps a natural heuristic to use for dynamic scheduling. However, a simple example can illustrate that it is not clearly the best heuristic. Consider the state of the system in Figure 3 when disk 3 becomes available. Our first instinct might be to serve the longer of the two queues, i.e., queue 3. On the other hand, if disk 3 empties queue 2 (there is only one customer in that queue), then server 2 will be free to concentrate on the backlog in queue 1. This example illustrates that in order to avoid the problem of some disks experiencing a high load while others are idle, we need to minimize the difference

in disk loads, instead of minimizing the difference in request queue lengths (as in the LQF algorithm).

First, we consider a simple definition of a disk load to be the sum of queue lengths of the two queues that a disk can service. Then the state information is expressed as $(L_0, L_1, \ldots, L_{D-1})$, where L_i is the sum of q_i and q_{i-1}, as defined in section 3.2. Our service decision criterion can be illustrated using the example discussed above. When disk 3 becomes available for service, the state information is $(3, 5, 5, 3)$, refer to Figure 4(a). If disk 3 chooses to serve a customer from queue 2, then the new state information will be $(3, 5, 4, 2)$, refer to Figure 4(b). If disk 3 chooses to serve a customer from queue 3, then the new state information will be $(2, 5, 5, 2)$, refer to Figure 4(c). It is clear that Figure 4(b) depicts a more balanced state than the one in Figure 4(c), i.e., the choice to serve from queue 2 *minimizes the difference in disk loads.* This criterion is the concept of majorization as it is

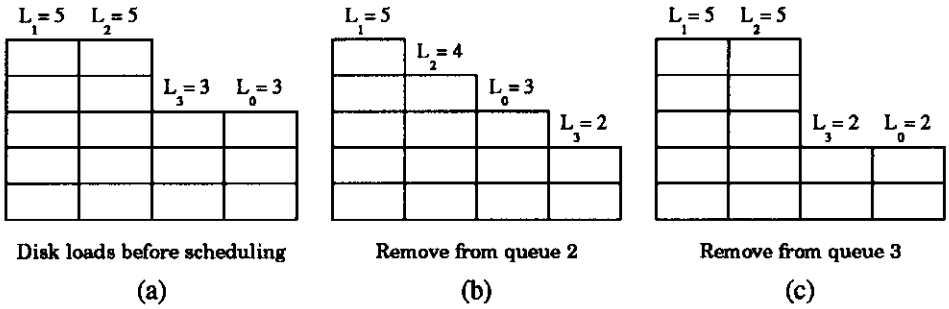

Figure 4: Minimizing Differences in Load

described in [9]. To state formally, given two vectors $X = (X_1, \ldots, X_n)$ and $Y = (Y_1, \ldots, Y_n)$ where X and $Y \in Z^n$, let $\hat{X}_i$, $\hat{Y}_i$ denote the i-th largest elements of X and Y respectively. Then Y majorizes X ($X \prec Y$) iff:

$$\sum_{i=1}^{k} \hat{X}_i \leq \sum_{i=1}^{k} \hat{Y}_i, \qquad k = 1, \ldots, n-1 \qquad \textit{and} \tag{1}$$

$$\sum_{i=1}^{n} \hat{X}_i = \sum_{i=1}^{n} \hat{Y}_i \tag{2}$$

Below is our second dynamic scheduling algorithm, which we refer to as the majorization algorithm. Given that the current state is $(L_0, \ldots, L_{i-1}, L_i, L_{i+1}, \ldots, L_{D-1})$, we illustrate the procedure for the case of both queues not being empty (since that is the only non-trivial case).

Majorization Algorithm (MAJ)

If $(L_0, \ldots, L_{i-1} - 1, L_i - 1, L_{i+1}, \ldots, L_{D-1}) \prec (L_0, \ldots, L_{i-1}, L_i - 1, L_{i+1} - 1, \ldots, L_{D-1})$
  serve the first customer in queue $i - 1$
If $(L_0, \ldots, L_{i-1}, L_i - 1, L_{i+1} - 1, \ldots, L_{D-1}) \prec (L_0, \ldots, L_{i-1} - 1, L_i - 1, L_{i+1}, \ldots, L_{D-1})$
  serve the first customer in queue i
Otherwise (either choice will give the same new state)
  apply the LQF algorithm

3.4 M/M/K Lower Bound

To better illustrate the significance of a dynamic scheduling scheme we will compare its performance not only to that of a static scheme, but also to some lower bound on the performance of the disk subsystem. If we can show that the performance of the dynamic scheme is much closer to that of the lower bound rather than that of the static scheme, then we can justify the added complexity associated with dynamic scheduling[4]. In this section we will present a relatively loose but still interesting lower bound for comparison.

We can achieve a lower bound by relaxing some constraint on the scheduling discipline. In this case we relax the condition that a disk can serve only two of the D queues; instead, we allow each disk to serve all D queues. This is then a simple $M/M/K$ [8] model of the disk subsystem (refer to Figure 5). Although this is an unachievable lower bound, it serves

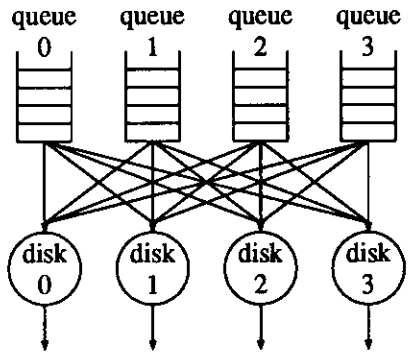

Figure 5: M/M/K Lower Bound Model

our purpose for comparison. We choose an M/M/K system because it is known to have a very flat response time curve as a function of the load, almost up to the point that the system is saturated (see Figure 6). As will be shown, the dynamic scheduling with chained declustering achieves similar behavior.

[4]This applies to physical level replication; of course, other issues must be considered in the case of logical level replication.

3.5 Results

In this section we compare the performance of the dynamic scheduling algorithms discussed in section 3.3, optimal[5] static scheduling, and the lower bound M/M/K model. We also discuss the difference in performance of the two dynamic algorithms, LQF and MAJ, in those cases where the difference is significant. The mean system response time is used as the measure of performance. For simplicity of illustration, each disk has an exponential service rate, μ_i, with mean equal to 1.0. The results for the dynamic schemes are obtained through simulations; the results for the static scheme and the lower bound model are obtained through simple equations (see [7] and [8] for details). All results presented in the remainder of the paper are for a 16 disk subsystem.

3.5.1 Uniform Load

In this section we consider a uniformly loaded system, i.e., each queue experiences the same arrival rate. Figure 6 is a graph of the system response time, as a function of utilization, for the M/M/K model, the static scheduling scheme, and one of the dynamic scheduling schemes[6]. We can see that there is a significant improvement in performance in changing from the static to the dynamic scheme. At 0.9 utilization, the response time of the M/M/K model is about 1.4, and the response time for the dynamic scheme is around 2.6. Note that the difference of 1.2 is relatively small compared to the 10.0 response time of the static scheme. Figure 7 illustrates results for the uniform workload where 20% of the workload is due to write requests. The results are similar to those of the pure read case. For the remainder of the paper we will discuss only the pure read cases, since inclusion of write requests does not provide any additional insight into our problem (i.e., qualitative results are similar).

3.5.2 Skewed Load

As mentioned earlier, skewed workload is one of the cases where dynamic scheduling is most advantageous (see section 3.3). In this section we present two types of skewed workload: (1) alternating between frequent and infrequent access, e.g., queue 0 has a high request rate, queue 1 has a low request rate, queue 2 has a high one, etc., and (2) gradual increase in frequency of access, e.g., queue 0 has a low request rate, queue 1 has a slightly higher request rate, queue 2 has an even higher one, etc. Figures 8 and 9 depict results for the two types of skewed workload. Here, the dynamic scheme's curve closely follows the M/M/K curve, i.e., it remains relatively flat almost until the system saturates. Note that the performance

[5]By optimal we mean that the response times for the static scheme are calculated after balancing the load among the disks as much as possible for a given case (for an example, refer to Figure 1). Note that it is not always possible (due to skewness in workload) to partition the load among the disks *equally*.

[6]We will show results for both dynamic schemes only when the difference between them is significant.

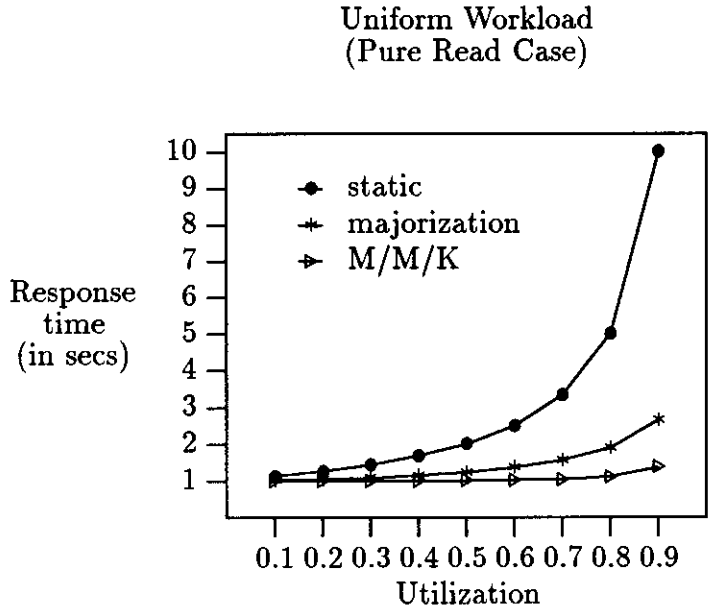

Figure 6: Uniform Workload – Pure Read Case

of the M/M/K model is immune to the skewness in the workload since any disk in the M/M/K model can serve any of the queues, i.e., it is a loose bound. Therefore, it is not surprising that the *quantitative* differences between the M/M/K model and the dynamic scheme performance are large for this type of workload.

3.5.3 Failures

One consequence of a disk failure is an increase in the workload of both of its neighbors, i.e., two of the D queues are left with only *one* server, instead of two. Hence, we can view a failure as a special case of skewness in the workload (assuming a pure read workload before the failure). Figure 10 illustrates the dynamic scheme's response time curve, which increases very gradually almost to the point of system saturation. It is clear that the static scheme is not able to distribute the increase in load, due to failure, nearly as well. The lower bound curve is obtained by using an M/M/K−1 model[7].

A disk failure is also the case where the two dynamic schemes exhibit significantly different performance. Table 1 contains the response times, as a function of utilization, for the uniform workload case; it also presents the percentage differences between these response times. We present only the high workload results, since that is where the significant difference occurs. Note that MAJ achieves as high as 19% better performance than LQF for the case of 0.92 system utilization.

[7]The graph for the skewed workload with a failed disk is not shown due to the lack of space; it exhibits similar characteristics.

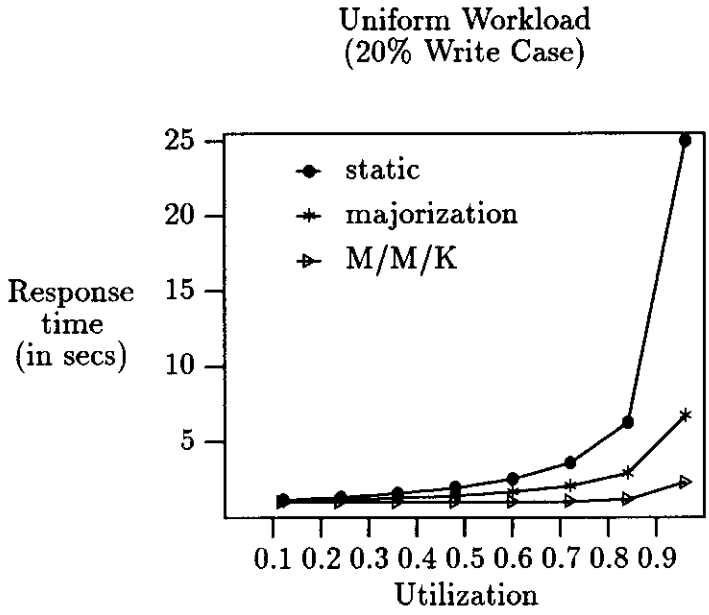

Figure 7: Uniform Workload – 20% Write Case

3.6 Stability Conditions

Since we are considering a highly utilized I/O subsystem, it is important to understand under what conditions this system can remain stable. In this section we investigate the problem of stability and characterize the saturation conditions for our model. We consider two issues: (1) how does a scheduling scheme approach the saturation point, and (2) where that saturation point occurs.

Consider again the graphs presented in section 3.5 where response time curves are given, using various workloads, for static scheduling, dynamic scheduling, and the M/M/K lower

utilization	LQF	MAJ	percentage difference
0.750	2.621	2.475	5.89 %
0.775	2.871	2.743	4.66 %
0.800	3.210	3.004	6.86 %
0.825	3.643	3.406	6.95 %
0.850	4.335	3.956	9.58 %
0.875	5.415	4.712	14.92 %
0.900	7.124	6.111	16.58 %
0.920	9.827	8.252	19.09 %

Table 1: Response times for LQF and MAJ algorithms.

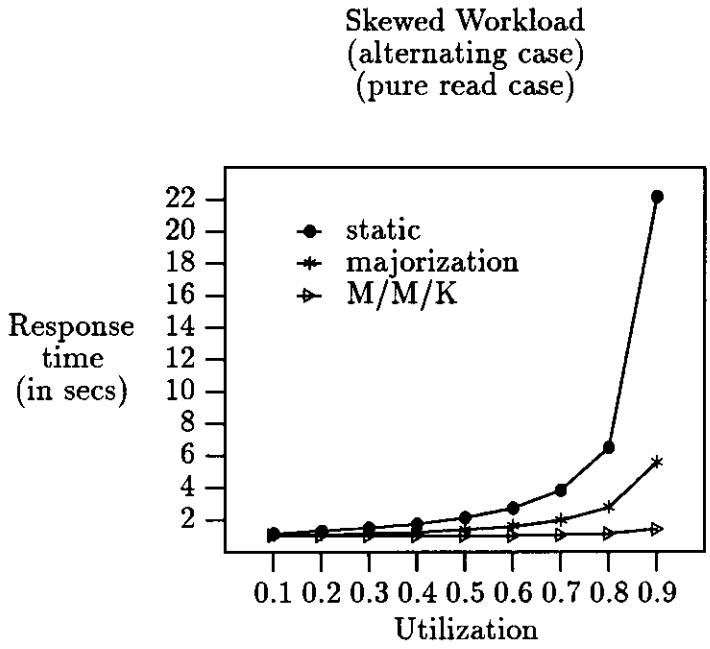

Figure 8: Skewed Workload (type 1) – Pure Read Case

bound model. Both static and dynamic scheduling schemes asymptotically saturate at the same point. In the case of a failed disk and/or skewed access, their saturation occurs at lower utilizations than in the case of the M/M/K lower bound model. This is due to the relaxation of constraints in the service discipline. However, an important difference which was pointed out earlier, is that the shape of the response time curve for the dynamic scheme is much like that of the M/M/K model, i.e., it remains flat almost until the point of saturation. This is not true in the case of the static scheme.

Since both dynamic and static schemes saturate at the same level of workload, let us, for ease of illustration, consider the static case. We present the following condition which characterizes the workload that can be sustained by our system without reaching saturation.

Let λ_i be the arrival rate to queue i, $i = 0, 1, \ldots, D-1$. The system is stable if for any i and any k, $k = 1, 2, \ldots, D-1$, the following holds:

$$\sum_{j=0}^{k-1} \lambda_{(i+j) \bmod D} \; < \; k+1 \qquad \textit{and} \tag{3}$$

$$\sum_{j=0}^{D-1} \lambda_j \; < \; D \tag{4}$$

We would like to offer the following intuition for the this condition. Any set of N consecutive queues can be serviced by $N+1$ disks (except in the case where $N = D$). We constrain the combined arrival rate to the N queues to be less than the combined service rate of the $N+1$

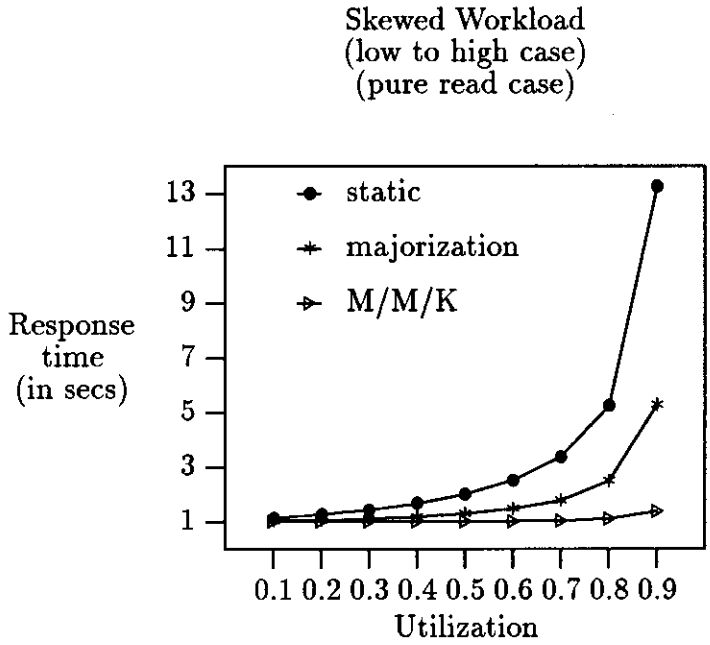

Figure 9: Skewed Workload (type 2) – Pure Read Case

disks. This is the condition for stability for a simple queueing system (see [8]).

As a simple example of the stability problem, consider a system which starts out with a uniform workload. First, change the arrival pattern as follows: (a) increase (slightly) one λ_i, and (b) decreasing all the other λ_j so as not to change the total workload; i.e., create a "spike" in the workload. Then allow this "spike" to grow, while keeping the total workload constant. If we now obtain the system response time using both dynamic and static schemes and plot it as a function of the height of the "spike", then we should observe the following: (1) the two schemes should saturate at the same point, and (2) the dynamic scheme's curve should grow a lot more gradually than the static scheme's curve.

Referring to Figure 11, we begin with a uniform workload, with the total load being 8.0, and then allow the frequency of access to data fragment A, i.e., queue 0, to grow (or become a "hot spot"). While the "spike" height is small both the static and the dynamic schemes are able to balance the load, and the response time curves remain almost flat. But, as the height of the "spike" continues to grow, the static scheme looses its ability to optimally balance the load, i.e., keep all the disks equally loaded; this occurs approximately around $\lambda_0 = 1.3$. Beyond this point, the static scheme's response time curve increases sharply. The response time curve of the dynamic scheme continues to grow fairly gradually. At $\lambda_0 = 2.0$, the system becomes unstable, since the load on disks 0 and 1 exceeds their service capacity.

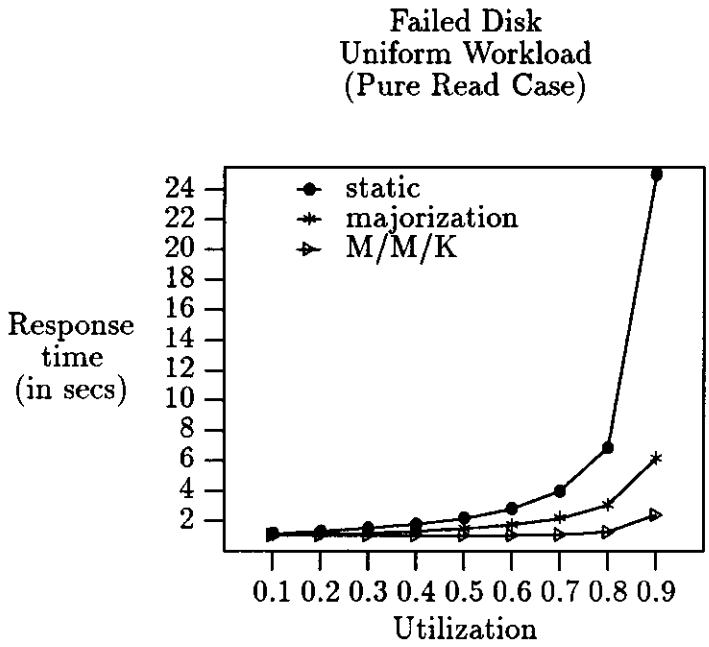

Figure 10: Failed Disk – Uniform Workload – Pure Read Case)

4 Conclusion

We have set out to study the effects of dynamic scheduling on I/O subsystems with chained declustering scheme of data replication applied at the physical level. We have illustrated its performance, at various workloads, for fully functional systems as well as systems with a single failure. We have showed that a dynamic scheduling scheme is significantly better than a static scheme at balancing the load among the disks and much more robust to skewness in the workload as well as disk failure. We conclude that at the physical level these benefits compensate for the overhead and complexity associated with dynamic scheduling schemes. Whether this remains to be true at the logical level is the topic of our current research.

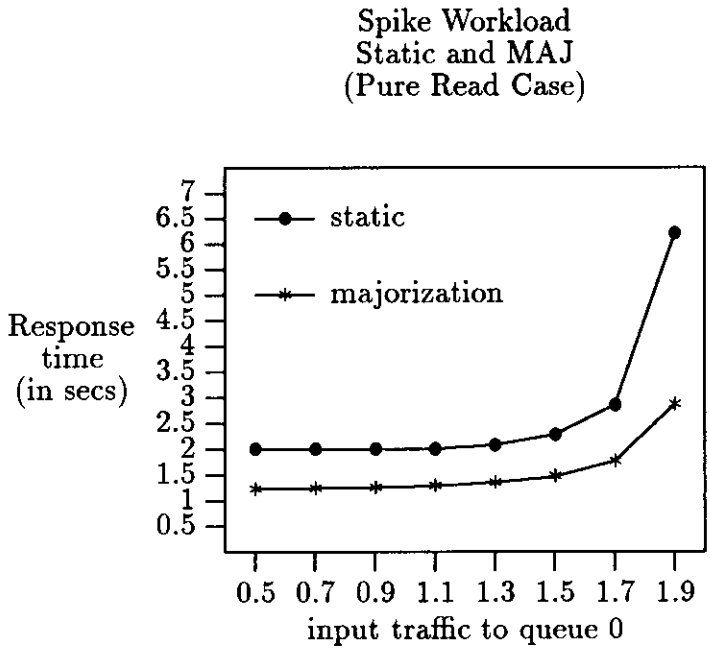

Figure 11: Saturation of Dynamic and Static Schemes

References

[1] NonStop SQL, A Distributed, High-performance, High-reliablity Implementaion of SQL. Technical Report No. 82317, Tandem Database Group, March,1987.

[2] DBC/1012 database computer system manual release 2.0. Technical Report Document No. C10-0001-02, Teradata Corporation, Nov 1985.

[3] D. Bitton and J. Gray. Disk shadowing. *VLDB*, pages 331–338, 1988.

[4] S. Chen and D. Towsley. Performance of a mirrored disk in a real-time transaction system. *ACM Sigmetrics 1991*, pages 198–207, 1991.

[5] David J. Dewitt, R. Gerber, G. Graefe, M. Heytens, K.Kumar, and M.Muralikrishna. Gamma : A high performance dataflow database machine. *VLDB Conference*, pages 228–240, 1986.

[6] Jim Gray, Bob Horst, and Mark Walker. Parity striping of disc arrays: Low-cost reliable storage with acceptable throughput. *VLDB Conference*, pages 148–172, 1990.

[7] H. Hsiao and D. J. DeWitt. Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines. *Proc. of Data Engineering*, pages 456–465, 1990.

[8] L. Kleinrock. *Queueing Systems, Volume II: Computer Applications.* Wiley-Interscience, 1976.

[9] A. W. Marshall and I. Olkin. *Inequalities: Theory of Majorization and Its Application.* Academic Press, 1979.

[10] Richard R. Muntz and John C.S. Lui. Performance analysis of disk arrays under failure. *VLDB Conference*, pages 162–173, 1990.

[11] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). *ACM SIGMOD Conference*, pages 109–116, 1988.

[12] D. Towsley, S. Chen, and P.S. Yu. Performance analysis of a fault tolerant mirrored disk system. *Proceeding of Performance '90*, pages 230–254, 1990.

[13] Philip S. Yu and Asit Dan. Effect of system dynamics on coupling architectures fro transaction processing. Technical Report RC 16606, IBM T.J. Watson Research Division, Feb 1991.

[14] Philip S. Yu and Asit Dan. Impact of affinity on ther performance of coupling architectures for transaction processing. Technical Report RC 16431, IBM T.J. Watson Research Division, Jan 1991.