Database Reorganization in Parallel Disk Arrays with I/O Service Stealing

Peter Zabback* Ibrahim Onyuksel† Peter Scheuermann‡ Gerhard Weikum§

Abstract

We present a model for data reorganization in parallel disk systems that is geared towards load balancing in an environment with periodic access patterns. Data reorganization is performed by disk cooling, i.e., migrating files or extents from the hottest disks to the coldest ones. We develop an approximate queueing model for determining the effective arrival rates of cooling requests and discuss its use in assessing the costs versus benefits of cooling actions.

Index Terms:

database reorganization, load balancing, temporal access patterns, parallel disk systems, approximate queueing model, I/O service stealing

1 Introduction

Database reorganization plays an important role in the performance tuning of dynamic systems with evolving access patterns. In this environment it is highly desirable to invoke an on-line database reorganization scheme in which the reorganization actions are performed concurrently with regular transactions [2, 3]. Thus, in contrast to off-line reorganization, which is performed while disallowing regular transactions, on-line reorganization is usually performed incrementally as a lower priority transaction [7]. In a centralized database system reorganization is performed in order to reduce the access time [2, 3]. In contrast, the main objective of data reorganization in parallel database systems is load balancing.

We present a new model for database reorganization in parallel database systems which allows the system to determine at a given point in time whether a reorganization action is cost beneficial or not, given that the reorganization itself imposes an additional load on the system. Reorganization is performed dynamically

*Tandem Computers Incorporated, 10100 North Tantau Avenue, Cupertino CA 95014-2542, USA, e-mail: zabback@patch.tandem.com

†Northern Illinois University, Department of Computer Science, DeKalb, IL 60115, USA, e-mail: onyuksel@cs.niu.edu

‡Northwestern University, Department of Electrical Engineering and Computer Science, Evanston, IL 60208, USA, e-mail: peters@ece.nwu.edu

§University of the Saarland, Department of Computer Science, P.O. Box 151150, D-66041 Saarbruecken, Germany, e-mail: weikum@cs.uni-sb.de

by employing an incremental data migration procedure called disk cooling. Disk cooling migrates data from "hot" (i.e., heavily utilized) disks to "cold" (i.e., lightly utilized) disks.

Our model differs in a number of aspects from other studies proposed in literature for determining optimal reorganization points in centralized database systems[6, 8]. Although the cost of performing a reorganization was considered, reorganizations were viewed as occurring instantly, thus having no effect on the overall system load. In addition, these assumed that the cost of performing a regular transaction always increases in time if the database is not in a reorganized state.

Our reorganization model is geared towards a workload in which a substantial proportion of the transactions exhibit a periodic pattern of access characteristics. In such a case, it may be beneficial to postpone a reorganization for a later point in time when there are fewer regular transactions. In our system a data migration request consists of two phases, a read phase, where the hottest disk is accessed, and a write phase, where the coolest disk is accessed. However, the read phase of a migration action, and hence the entire cooling action, is not executed if the service queue of the source disk is not empty. Since the source disk carries the heaviest share of the load, scheduling a reorganization action would most likely increase the loadimbalance if the queue at the source disk is not empty. The model proposed here is a generalization of the earlier models introduced in [4, 5]. The model used in [5] did not consider periodic access patterns. In contrast, in [4] we considered explicitly periodical access patterns, but all reorganization actions were treated as lower priority requests.

We shall refer to the read requests of the reorganization actions under a no-enqueueing policy as I/O service stealing requests, given their analogy to cycle stealing operations in the CPU execution. As part of our reorganization scheme, we have developed an approximate queueing model for a system with two types of requests, namely regular requests and I/O service stealing requests. Using this model, we can determine the intervals in time when the read and write phases of the reorganization actions will be scheduled, given the moment in time that the load balance is observed.

2 Data Redistribution in Parallel Disk Systems

We have implemented an intelligent file manager, called FIVE, for parallel disk systems that can perform striping on a file-specific or global basis as desired by the application, and in addition achieves load balancing by judicious file allocation and redistribution of data [4, 5]. In order to perform load balancing, our file system keeps track of the following related statistics [5]:

- the heat of files (or extents, i.e., smallest units of data migration) and disks, where the heat is determined as the number of block accesses to a file or disk per time unit, as measured by statistical observation over a moving window of a certain length.
- the temperature of files (extents), which is defined as the ratio between heat and size.

An extent is a file fragment which consists of all the striping units of a file that are allocated to the same disk in a single allocation unit [7]. Note that the heat metric captures the number of block accesses due to regular requests, and thus we obtain the following relationship between the heat of a disk i, H_i, and the mean arrival rate of regular requests, λ_r:

$$\lambda_r = \frac{H_i}{\overline{R}} \tag{1}$$

where $\overline{R}$ is the average request size measured in terms of the number of blocks accessed.

The above formula assumes that the access patterns of files, hence disks, are fixed in time. In practice, we encounter many environments which exhibit periodical, predictable access patterns. In our model for database reorganization these periodic patterns can be incorporated by identifying a number of intervals such that the heat of a file stays constant accross an interval, but is allowed to vary across them. As in [4], we define the weighted heat of file k as:

$$WFH_k^* = \sum_{j=1}^{n} H_{k,j}^* \times (t_j - t_{j-1}) \tag{2}$$

where

n	is the numbers of intervals
$t_j - t_{j-1}$	is the length of interval j
$H_{k,j}^*$	is the heat of file k in interval j

Correspondingly, the weighted heat of disk i is defined as:

$$WDH_i = \sum_{j=1}^{n} H_{i,j} \times (t_j - t_{j-1}) \tag{3}$$

where $H_{i,j}$ is the heat of disk i in interval j. $H_{i,j}$ is computed as the accumulated heat in interval j of all files that reside on disk i. Note that arrival rate λ_r is also a function of the interval in time; we assume that $\overline{R}$, the average request size, is constant across all intervals, but as the heat of the disk changes, we obtain now $\lambda_{r,m} = \frac{H_{i,m}}{\overline{R}}$, for $m = 1, \ldots, n$.

2.1 Temporal Disk Cooling Algorithm

In order to perform dynamic heat redistribution we employ in our system a dynamic load balancing procedure called *disk cooling*. Basically, disk cooling is a greedy procedure which tries to determine the best candidate, i.e., file (or extent) to remove from the most utilized disk, i.e., the disk with the highest weighted heat, in order to minimize the amount of data being moved while obtaining the maximum gain. The (weighted) temperature metric is used as the criterion for selecting the files (extents) to be reallocated because temperature reflects the benefit/cost ratio of the reallocation. The file to be moved is reallocated to the disk with the lowest weighted

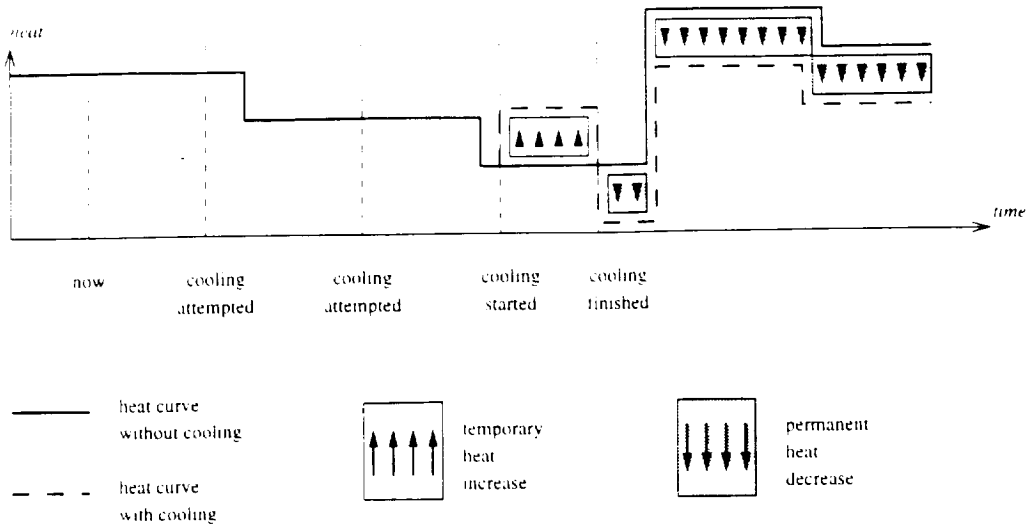

Figure 1: Impact of cooling on "hot" disk

heat. In the case of an extent, in order to facilitate intra-request parallelism, an additional constraint is observed, namely that the target disk should not already hold an extent of the corresponding file.

In our system the disk cooling procedure is implemented as a background daemon which is invoked at fixed intervals of time. The procedure checks first if a given trigger condition is satisfied or not [5]. If the trigger condition is false, the system is considered load balanced and no cooling action is performed.

A cooling action will be executed only if our estimate of its benefit exceeds its additional cost, with both measures taking into account this temporal access pattern. In order to estimate the cost/benefit of a cooling action we make use of the weighted disk heat variance (WDHV) as an explicit objective function [4]. WDHV is defined as follows:

$$WDHV(H) = \sum_{j=1}^{n} \sum_{i=1}^{D} (\bar{H}_j - H_{i,j})^2 \times (t_j - t_{j-1}) \tag{4}$$

where

D is the number of disks in the parallel system

$\bar{H}_j$ is the mean disk heat (over all disks) in interval j

The benefit of the cooling action is measured by examining the load balance of the system before and after the potential reorganization. This benefit, denoted by B, is computed as the difference $WDHV_{curr} - WDHV_{future}$, where $WDHV_{curr}$ is the weighted disk heat variance before the potential cooling process and $WDHV_{future}$ is the weighted disk heat variance that potentially would result if the extent were to be moved to the target disk. In order for the cooling to be scheduled, its benefit B must exceed the extra cost, denoted by E, introduced by the reorganization process itself. The cooling process is executed in two

steps, the first corresponding tothe *read* phase of the action, where the hot disk is accessed, and the *write* phase of the action, during which the cold target disk is accessed. The read(and write) phases introduce an additional amount of heat on the source and target disks which can be computed by dividing the size of the file (extent)to be moved by the corresponding duration of the phase. The read phases correspond to I/O stealing requests and, as discussed in Section 3, the response time ofan I/O stealing request is equal to its service time, denoted by $1/\mu_s$. Thus the duration of a read phase is estimated as $1/\mu_s$. Figure 1 illustrates the temporal heat changes on the source disk with and without cooling. The permanent heat reduction due to the read phase is already accounted for in the benefit B; on the other hand, to determine the extra cost (temporary heat increase in Figure 1) we also need to determine the interval in time when the cooling started.

2.2 Estimating the intervals of the cooling action

Assuming that the cooling daemon is invoked at time *now* an iterative procedure is invoked in order to determine the intervals in time, denoted by m and n, when the read and write phases will be actually scheduled and executed at the corresponding service queues. Let us assume that the cooling daemon is invoked during time interval j, i.e. $t_j - 1 \leq now \leq t_j$. Using the mean arrival rate of regular requests during interval j, $\lambda_{r,j}$, and the arrival rate of the disk cooling requests, λ_s, the approximate queueing model developed in Section 3 is used first to determine λ_{eff}, the effective arrival rate of the read actions of the corresponding cooling requests. We assume, for simplicity, that the trigger condition is always satisfied, i.e., some heat imbalance is always present. Thus, the mean arrival rate of disk cooling requests in our system, which correspond to the service stealing requests in the queueing model, is fixed and can be calculated as:

$$\lambda_s = \frac{1}{\text{time between successive daemon invocations}} \tag{5}$$

The interval m where the read part of the the cooling request would be scheduled is determined as by the following iterative procedure. Notice that this procedure may require to recompute the value of λ_{eff}.

compute λ_{eff} using equation (9);
`while` $(interval = NOT\_FOUND)$ `do`

- `Case 1:` $(t_{j-1} \leq (now + 1/\lambda_{eff}) \leq t_j)$:
  $m := j$; $interval := FOUND$;
- `Case 2:` $((t_j \leq (now + 1/\lambda_{eff}) \leq t_{j+1})$ *and* $(\lambda_{r,j-1} \leq \lambda_{r,j}))$:
  $m := j + 1$; $interval := FOUND$;
- `Else`:

Reiterate procedure to compute λ_{eff} using λ_{eff-1} and λ_s:

Recompute $now := now + (i * \lambda_s)$, where $i = \min\{k : now + (k * \lambda_s) \geq t_j\}$;

endwhile

The computation of the interval n, where the write phase of the cooling request would be scheduled, is substantially simpler. Since the target disk is cool we can schedule the corresponding reorganization request as soon aspossible. Thus, if the reading phase was executed in interval m, the write phase will be scheduled in the same interval m or in interval $m + 1$ [9].

Having determined the intervals m and n, we can compute E, the extra cost due to cooling, as follows. We add two "dummy" intervals to the load balancing cycle to account for the read and write phases of the cooling action. During such a dummy interval the heat of each disk, except for the disk which is the subject of the read or write, correspondingly, is taken to be the same as the heat of the disk during the time interval when the corresponding read or write phase started. Thus, the terms in E can be computed as follows:

$$E = \sum_{i=1}^{D} (\overline{H}'_m - H'_{i,m})^2 * read\_duration + \sum_{i=1}^{D} (\overline{H}''_n - H''_{i,n})^2 * write\_duration$$

where

$$H'_{i,m} = \begin{cases} H_{i,m} + extent\_size/read\_duration, & \texttt{if } i = s \\ H_{i,m}, & \texttt{otherwise} \end{cases}$$

$$H''_{i,n} = \begin{cases} H_{i,n} + extent\_size/write\_duration, & \texttt{if } i = t \\ H_{i,n}, & \texttt{otherwise} \end{cases}$$

More details of our temporal disk cooling procedure are given in [9].

3 An Approximate Queueing Model for I/O Service Stealing

I/O service stealing requests are issued periodically by the reorganization process whenever a load imbalance is observed, and they correspond to the read phases of the cooling actions. In this section we present an approximate queueing model for deriving the overall utilization and effective arrival rate, λ_{eff}, of I/O service stealing requests in a two class system consisting of regular and reorganization requests. The behavior of the two classes of requests is characterized as follows:

1. *regular requests*: these requests have a mean arrival rate λ_r. The interarrival time of these requests is assumed to be exponentially distributed. The mean service rate of these requests is given by μ_r.

2. *I/O service stealing requests*: these lower priority requests are issued periodically by an incremental reorganization process. We assume a constant interrarrival time $1/\lambda_s$ and a mean service rate μ_s for these requests.

For I/O service stealing requests two additional restrictions apply:

1. If an I/O service stealing request arrives and the service queue is not empty, the request is disregarded by the scheduler of the queue.

2. I/O service stealing requests are synchronous, i.e., a new I/O service stealing request is not enqueued until the execution of the previous one is finished. Thus, at any point in time there is at most one I/O service stealing request in the system.

From the discussion above it is clear that the response time of an I/O service stealing request is equal to its service time, i.e., $1/\mu_s$. We proceed now to derive the formulae for the effective arrival rate of I/O service stealing requests, λ_{eff}, as seen by the service center, and the overall system utilization, ρ.

The probability that an I/O service stealing request finds the service queue empty is given by $1 - \rho$. Thus, we obtain:

$$\lambda_{eff} = (1 - \rho) \times \lambda_s. \tag{6}$$

This is in effect a recursive formula since ρ depends on λ_{eff}. In order to eliminate the inherent recursion in formula (6), we adopt an approximation now and treat the system as a regular M/G/1 queue with two classes of prioritized requests [1] : regular requests have high priority, while the service stealing requests have low priority. Thus, we assume that the interarrival times for both regular requests and I/O service stealing requests are exponentially distributed. Note that in our actual system implementation the stealing requests have constant interrarrival times (see equation 5).

The utilization ρ_i due to requests with priority i in an M/G/1 queue with i priority classes is given by:

$$\rho_i = \lambda_i / \mu_i. \tag{7}$$

Furthermore, under the exponential interarrival times assumption, the overall utilization ρ of the system can be expressed as the sum:

$$\rho = \rho_r + \rho_s = \frac{\lambda_r}{\mu_r} + \frac{\lambda_{eff}}{\mu_s}. \tag{8}$$

Note that ρ depends only on the mean arrival and service rates of the two classes of requests, and is independent of the service time distributions. From equation (8) we obtain:

$$\lambda_{eff} = \left(\rho - \frac{\lambda_r}{\mu_r}\right) \times \mu_s. \tag{9}$$

Finally, substitution of equation (9) into equation (6) yields:

$$\rho = \frac{\lambda_s + \frac{\lambda_r}{\mu_r} \times \mu_s}{\lambda_s + \mu_s}. \tag{10}$$

In [9] we report on an experimental validation of this model and show that the maximum error of λ_{eff} ranges from 1% to 5% depending upon the arrival rates λ_r and λ_s of regular and I/O stealing requests.

Acknowledgment

Peter Scheuermann has been supported in part by NSF under grant IRI-9303583 and by NASA-Ames under grant NAG2-846.

References

[1] L. Kleinrock, *Queueing Systems* John Wiley & Sons, 1976.

[2] E. Omiecinski, L. Lee and P. Scheuermann, "Performance Analysis of a Concurrent File Reorganization Algorithm for Record Clustering," *IEEE Transaction on Knowledge and Data Engineering*, Volume 6, No.2, April 1994, pp. 265-357.

[3] B. Salzberg and A. Dimock, "Principles of Transaction-based On-line Reorganization," *Proceedings of the 18th International Conference on Very Large Databases*, 1992, pp. 511-520.

[4] P. Scheuermann, G. Weikum and P. Zabback, "Adaptive Load Balancing in Disk Arrays," *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms (FODO*, 1993, pp. 345-360.

[5] P. Scheuermann, G. Weikum and P. Zabback, "Disk Cooling in Parallel Disk Systems," *IEEE Data Engineering Bulletin* Vol.17 No.3, Sept. 1994, pp. 29-40.

[6] B. Shneiderman, "Optimum Data Base Reorganization Points," *Communications of ACM*, Volume 16, No. 6, June 1973, pp. 362-365.

[7] G. Weikum, P. Zabback and P. Scheuermann, "Dynamic File Allocation in Disk Arrays," *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1991, pp. 406-415.

[8] S.B. Yao, K.S. Das and T. Teorey, "A Dynamic Database Reorganization Algorithm," *ACM Transactions of Database Systems*, Vol. 1, No. 2, June 1976, pp. 159-174.

[9] P.Zabback, I. Onyuksel, P.Scheuermann and G. Weikum, "Temporal Database Reorganization with I/O Service Stealing," Technical Report, Northwestern University, Dept. of Electrical and Computer Engineering, 1996.