The Impact of Data Placement Strategies on Reorganization Costs in Parallel Databases

Kiran J. Achyutuni Edward Omiecinski Shamkant B. Navathe

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
e-mail: {kiran,edwardo,sham}@cc.gatech.edu

October 29, 1994

Technical Report No. GIT-CC-95/02

Abstract

In this paper, we study the data placement problem from a reorganization point of view. Effective placement of the declustered fragments of a relation is crucial to the performance of parallel database systems having multiple disks. Given the dynamic nature of database systems, the optimal placement of fragments will change over time and this will necessitate a reorganization in order to maintain the performance of the database system at acceptable levels. This study shows that the choice of data placement strategy can have a significant impact on the reorganization costs. Until now, data placement heuristics have been designed with the principal purpose of balancing the load. However, this paper shows that such a policy can be beneficial only in the short term. Long term database designs should take reorganization costs into consideration while making design choices.

1 Introduction

Declustering is a technique that partitions a file and spreads it across many disks [5], [11]. Performance studies have shown that, except under extremely high utilization conditions, declustering is consistently a better approach than placing the entire relation on a single disk [9], [10]. After a relation has been declustered, decisions must be made regarding the number of disks over which the relation will be distributed, and the particular disks on which to place the data [1]. In *full declustering*, a relation is declustered across all the available disks, whereas in *no declustering*, the entire relation is placed on a single disk. In many situations, less than full declustering outperforms both no declustering and full declustering. In this case, data placement is a non-trivial problem [1]. Effective data placement is crucial to the performance of database systems having multiple disks. A data placement is effective when the I/O load is distributed as uniformly as possible across all the disks.

The number of disks on which to place the relation fragments, called the degree of allocation, is determined based on a number of criteria: response time and throughput requirements of transactions, the access characteristics of the relation, the sizes of the relations, the number of disks available in the system etc. However, database systems are dynamic in that relations can grow in size, the access characteristics can change, and more disks and processors can be added. All of these changes can change the degree of allocation of the relation, and with it, there is an opportunity to improve the response time and throughput of queries and transactions. However, this opportunity can be fully exploited *only after* reorganizing and moving data to reflect the new allocation.

In this paper, we study the data placement problem from a reorganization point of view. More specifically, we address the following problem: given that the degree of allocation of a relation changes, what impact does a data placement strategy have on the consequent reorganization costs? Ideally, we seek a data placement strategy that not only does a good job of balancing the load across disks, but also minimizes the reorganization costs in the event of a reorganization. Minimization of reorganization costs is an important goal especially when the reorganization has to be performed on-line or concurrently with usage. When reorganization is performed without taking the database off-line or quiescing the transactions, it is called concurrent reorganization or on-line reorganization. On-line reorganization has been identified as a challenging problem by the database community [14, 17, 2, 15, 16]. The single most compelling reason to do on-line reorganization is the availability of the database during the reorganization. The conventional approach to reorganization is to take the database offline. Any business enterprise that relies on 24-hour availability such as reservation systems, global finance, hospitals, police etc. cannot afford to go offline to perform reorganization.

A fundamental parameter that directly controls the performance of online reorganization is the amount of data to be reorganized. In this paper, we seek the relationship between data placement strategies and the amount of data to be reorganized. We consider four data placement strategies - classical Bubba placement strategy [1], an optimization to the classical Bubba called Bubba Opt1, a hash placement strategy, and a hybrid hash placement strategy. The data placement strategies are compared with the following metric: the amount of data to be reorganized (also referred to as the reorganization cost). The reorganization cost plays a direct role in determining the magnitude of the performance impact on the transactions during on-line reorganization. Thus a lower reorganization cost of a data placement strategy translates to a smaller performance impact on transactions.

The paper is structured as follows: in section 2, we present the related work. In section 3, we motivate the problem by showing that, when the sizes of the relations change significantly, it pays to change the degree of allocation of relations and re-place the relations using a data placement strategy. This motivates us to examine the relationship between data placement strategies and the reorganization costs. We use the TPC-C benchmark to make a simulation study. In section 4, the various data placement strategies are discussed. In section 5, an experimental study of the the relationship between the data placement

strategies and the amount of data to be reorganized is presented. In section 6, we develop and validate the analytical models that describe the relationship. Finally, the conclusions are made in section 7.

2 Related Work

In [1], the authors introduce the Bubba placement strategy, which places relation fragments in descending order of their weights, and selecting the least loaded disk in every step. The initial placement for the relations is determined by applying the Bubba placement strategy for all the relations simultaneously. If the disks become imbalanced after the initial placement, they re-place the relations causing the imbalance. To determine the new placement for the relation under consideration, they use the Bubba placement strategy, except that all relations other than the relation under consideration are already placed. But the degree of allocation for the relation remains unchanged. In this paper, we follow a similar method, i.e., we apply the placement strategy to one relation at a time, but systematically vary the degree of allocation and measure the reorganization cost.

In [4], the authors present an adaptive data placement scheme to rebalance the disks when the disks become imbalanced due to insertions and deletions. Their scheme tries to minimize the data movement by retaining as many fragments on the disks as possible. Their technique is useful only when the degree of allocation of the relation remains unchanged.

In [13, 12], the authors describe an adaptive method for data allocation and load balancing in disk arrays that responds to evolving access patterns. Their methods place more emphasis on maintaining the load balance rather than maintaining the degree of allocation. It is important to maintain the degree of allocation of relations because it has a direct bearing on the response time of transactions. The methods in their paper can unintentionally cause the degree of allocation of relations to change. This is not a desirable feature.

In [19], the authors present a method to determine the degree of allocation of a relation. Their method assumes the system is operated in single user mode, and all read and write requests are for the entire file, rather than for fractions of files as in a database environment. The authors also propose dynamic placement strategies. We do not examine the relationship between dynamic placement strategies and reorganization costs, because dynamic placement strategies are suitable in a file system environment, where files are created and deleted often. In a database environment, all the relations are known at database design time, and it is very rare that new relations are added. New relations are typically added when the database schema is changed. Hence we focus on static data placement strategies.

3 Motivation

Is it necessary to change the degree of allocation of a relation? To answer this question, we will make a simulation study to determine if a change in the degree of allocation can actually improve the performance of queries and transactions. For the simulation study, we chose the

Table Name	Initial Size	Size after 3 months
WAREHOUSE	0.089	0.089
DISTRICT	0.950	0.950
CUSTOMER	19650	19650
HISTORY	1380	45450
ORDER	720	23850
NEW-ORDER	72	72
ORDER-LINE	16200	238482
STOCK	30600	30600
ITEM	8200	8200

Figure 1: TPC-C relations and their sizes (in 1000 bytes).

TPC (Transaction Processing Council) Benchmark C (TPC-C) [3], which models a medium complexity online transaction processing workload. The TPC-C is patterned after an order-entry workload, with multiple transactions types ranging from simple transactions that are comparable to the simple debit-credit transaction to medium complexity transactions that have two to fifty times the number of calls of the simple transactions. The workload also includes a join query. In [8], the authors make a modeling study of the TPC-C benchmark and determine the access skew for the relations, and the access pattern of the queries for the various relations.

There are 9 relations and 5 query types in the TPC-C benchmark. The benchmark specifies the initial sizes of the relations, their growth rate, and the frequency of queries. Figure 1 shows the relations and their sizes, and Figure 2 shows the workload mix as specified in the TPC-C benchmark and workload mix we used in our simulation study. Observe that the fraction of *Stock Level* transaction is 0.3%. This is because of the following: in the TPC benchmark, the transaction *Stock Level* has a join between ORDER-LINE and STOCK, and it also allows indexes on the tables. So the join specified in the benchmark can be computed using the indexes and thus the join can be relatively insensitive to the size of the relations involved. However, in a real transaction processing workload, there are ad-hoc queries where join processing can be computed only by scanning the relations involved. In order to make our simulation reflect a real-life workload, we compute the join using only table scans. Therefore, the frequency of *Stock Level* transaction is set to 0.3%, because a higher frequency can overwhelm the disks.

The simulator uses a closed-queueing model. The TPC-C benckmark specified a multi-programming level of 10 for each warehouse in the WAREHOUSE table. It is assumed that there is only one warehouse in the database (the sizes of relations in figure 1 is for one warehouse), 8 disks in the system, and the average I/O time for disk access is 25 msec. The page size is assumed to be 4K bytes. We determine the degree of allocation for each of the

Transaction	Minimum %	Simulation %
New Order	$\star$	45.7
Payment	43	45.0
Order Status	4	4.0
Delivery	4	5.0
Stock Level	4	0.3

Figure 2: Workload mix as specified in TPC-C and as used in our simulation

Relation	New Order	Payment	Order Status	Delivery	Stock Level
WAREHOUSE	1	1			
DISTRICT	1	1			1
CUSTOMER	1	2	2	10	
STOCK	10				Entire
ITEM	10				
ORDER	1		1	10	
NEW-ORDER	1			10	
ORDER-LINE	10		10	100	Entire
HISTORY		1			

Figure 3: Number of items requested by various queries.

relations using a method to be described a little later in this section. No assumptions are made of the declustering techniques. However, it is assumed that after the fragments of the declustered relation have been placed on the chosen set of disks (equal to the degree of allocation of that relation), random requests to the relation will translate to uniform access to the set of disks. This is a reasonable assumption even when the relations have skew, because one of the goals of a placement strategy is to balance the load on disks in the presence of skew. After the degree of allocation for each of the TPC-C relations has been determined, they are placed on the disks using the Bubba placement strategy [1]. When queries make requests for data, requests are queued at the disks which hold the relation fragments. Table 3 shows the number of items requested by each query type from various relations.

The following method is used to determine the degree of allocation for each relation in the TPC-C benchmark. Earlier methods to determine the degree of allocation [19] were geared towards a single user system and for one relation at a time. This method is geared towards multiple user system and can determine the degree of allocation for all the relations in the database schema at the database design time. It is designed to minimize the response times for the mixed workload. A comparison of this method and earlier ones is beyond the

relation	Initial		3 months later	
	Utilization	Deg. of Alloc.	Utilization	Deg. of Alloc.
WAREHOUSE	0.025	1	0.014	1
DISTRICT	0.025	1	0.007	1
CUSTOMER	0.054	1	0.030	1
STOCK	0.811	4	0.406	1
ITEM	0.125	1	0.069	1
ORDER	0.028	1	0.015	1
NEW-ORDER	0.027	1	0.015	1
ORDER-LINE	0.642	3	2.70	7
HISTORY	0.013	1	0.014	1

Figure 4: Utilization of relations and the degree of allocation as determined by our method. Number of disks in the system = 8.

scope of this paper. The method involves the following steps:

1. Assume that there are as many disks as relations, so that each relation is placed on a separate disk.

2. Execute the query workload using the simulator.

3. Let D_i be the utilization of disk i. If the utilization of any disk is greater than 90%, then partition the relation and store it on two or more disks. This is because a disk with an utilization of greater than 90% becomes the bottleneck in the system, and reduces the utilization of other disks (because queries need data from more than one disk). Therefore, to get a true indication of the utilization of various relations, it is necessary that no disk has a utilization greater than 90%.

4. Let S be the cumulative utilizations of all the disks. Let N be the number of actual disks in the system. If the relations were placed on N disks, then each disk should theoretically have an average utilization of $A = S/N$.

5. The degree of allocation for relation i is given by D_i/A, rounded off to the nearest integer.

Figure 4 shows the utilizations and the degree of allocation of the relations. For ease of presentation, we will refer to the degree of allocation of relation in column three in Figure 4 as INITIAL DEGREE OF ALLOCATION, and the degree of allocation of relations in column five of Figure 4 as FINAL DEGREE OF ALLOCATION.

The simulation results are shown in Figure 5, which shows the response times for the various query types as well as the mixed workload. The second column is the response times with INITIAL DEGREE OF ALLOCATION for relations. The third column in

Transaction	Inital	After 3 months	
		Before Reorganization	After Reorganization
Mixed	2.0	14.60	8.22
New Order	2.0	22.85	8.32
Payment	1.76	1.49	6.80
Order Status	0.84	22.58	4.61
Delivery	2.60	21.99	7.49
Stock Level	50.89	525.78	210.947

Figure 5: Response times (in seconds) of the TPC-C query workload before and after reorganization.

Figure 5 is the response times after 3 months with INITIAL DEGREE OF ALLOCATION for relations. The fourth column is the response times of the queries after reorganizing relations with FINAL DEGREE OF ALLOCATION. Notice the improvement in response times after reorganization. The mixed workload shows about 43% improvement whereas *New Order* shows about 63% improvement. Except the query *Payment*, all the queries show significant performance improvement.

We also considered the case where the degree of allocation of relations in the initial database is the same as the degree of allocation of the relations after 3 months (i.e., the degree of allocation for relations was FINAL DEGREE OF ALLOCATION). Figure 6 shows the response times of the queries. Clearly, this results in inferior performance.

The simulation results shows that, for the initial database, the INITIAL DEGREE OF ALLOCATION provides better performance. After three months, because of the growth in the relation sizes, the FINAL DEGREE OF ALLOCATION provides a better performance. The simulation results provide conclusive evidence that the degree of allocation changes with the growth of the relations. Moreover, there is no one degree of allocation that is best for all sizes of the relation. Thus the degree of allocation of a relation is a dynamic parameter that changes with the size of the relation.

4 Data Placement Strategies

In this section, we describe four data placement strategies: the classical Bubba placement strategy, an optimization of classical Bubba called Bubba Opt1, hash placement, and a hybrid hash placement strategy called HUBBA. For the purpose of this paper, we assume that the relation is hash declustered into M buckets, and there are N disks in the parallel database system. Each hash bucket i is associated with a weight w_i, which is equal to the number of tuples in bucket i. (It is important to note that it is not essential for a relation to be hash declustered for the study in this paper to be valid.) It is reasonable to assume that the hash bucket weights have a Zipf-like distribution with parameter θ [6] (see Figure

Transaction	with INITIAL DEGREE OF ALLOCATION	with FINAL DEGREE OF ALLOCATION
Mixed	2.0	6.33
New Order	2.0	11.94
Payment	1.76	0.486
Order Status	0.84	1.24
Delivery	2.60	1.73
Stock Level	50.89	191.3

Figure 6: Response times (in seconds) of the TPC-C query workload for the initial database with two different degree of allocations.

7).

1. Classical Bubba Bubba's placement strategy [1] uses the following simple algorithm: place hash buckets on disks in the descending order of their weights, and choose the disk with the least cumulative weight in every step.

2. Bubba Opt1 placement strategy: With this strategy, the first time a relation is placed on the disks, the classical Bubba strategy is used. Every subsequent time the relation has to be re-placed, the placement for the heaviest N (equal to the number of disks) buckets is unchanged. But the remaining buckets are placed with the classical Bubba placement strategy.

3. Hash Placement strategy: In this strategy, the weights of the buckets are not used. Instead the hash buckets are placed on disks using another hash function of the type $K mod D$, where K is the hash bucket number, and D is the degree of allocation for the relation.

4. HUBBA placement strategy: This placement strategy exploits the Zipf-like distribution of the hash bucket weights. Depending upon a tunable parameter α, it uses the Bubba Opt1 placement strategy to place the first α% of the **heaviest** buckets, and the hash placement strategy to place the remaining (1-α)% of the buckets.

   The value of α can be varied from 0% to 100%. With α equal to 0%, HUBBA is completely hash; with α being 100%, HUBBA is completely Bubba Opt1. However, the parameter α should be selected such that α times the number of buckets is well *beyond* the knee of the Zipf-like distribution. Therefore, α can be expected to never exceed e^{-1} (i.e., 0.36) for any skew value and any number of buckets.

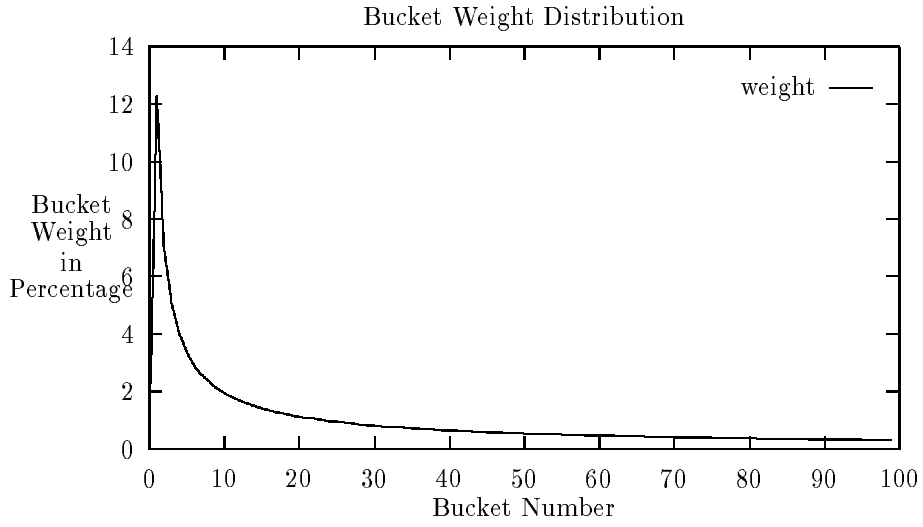


Figure 7: An example hash bucket weight distribution for $\theta = 0.2$ and $M = 100$.

5 Experimental Comparison of data placement strategies

In this section, we compare the data placement strategies using two metrics: the ability to load balance, and the reorganization cost.

5.1 Load Balance

We begin with by comparing the standard deviations in the load due to each of the data placement strategies. The standard deviations in the general case, as per the skew distribution in 7, is shown in figure 8. We also considered a bad case scenario for Zipf-like bucket weight distribution, wherein multiples of a certain bucket have the largest weights. In our experiments, we chose multiples of bucket 7 to have the largest weights. The bad case load variance is shown in 9.

To put these results into perspective, we quote Livny, Khoshafian, and Boral from their paper [9]: "In the recent International Workshop on High Performance Transaction Processing, a United Airlines representative reported a differential of only 1% between the busiest and most idle disks. This is achieved by hand tuning the data placement from day to day". We relax this a little bit and define a data placement strategy to be effective if it produces a load deviation of no more than 5%. By this standard, Bubba and Bubba Opt1, and HUBBA are all effective data placement strategies. However, hash placement is not quite effective. In fact, when the number of disks is increased, the number of buckets decreased, but the skew is increased, the effectiveness of the hash placement strategy decreases. This is also reflected in the effectiveness of HUBBA because hash constitutes (1-α)% of HUBBA. The excessive dependency on hash is one of the limitations

Number of Disks	Mean disk wt.	Standard Deviation			
		Classical	Bubba Opt1	Hash	HUBBA
4	25.0	0.00	0.00	4.70	0.60
5	20.0	0.13	0.13	4.42	0.26
6	16.6	0.16	0.16	4.28	0.78
7	14.2	0.09	0.09	4.08	1.38
8	12.5	0.13	0.13	3.96	1.81
9	11.1	0.46	0.46	3.70	1.97
10	10.0	0.80	0.80	3.61	1.93
11	9.09	1.06	1.06	3.48	2.35
12	8.33	1.25	1.25	3.50	2.35

Figure 8: Normal case load deviation in percent: $\theta = 0.2$, buckets=100, α=0.3

of HUBBA. This is to be expected because the hash placement does not use the weights of the hash buckets. Note also that in the bad case scenario, the variance due to the hash placement strategy is disastrously high. When the number of disks was 7, the standard deviation is as much as 12% and almost as much as the mean itself. This is because there is a strong correlation between the weights of buckets which are multiples of 7. The hash placement strategy $K \bmod 7$ placed all these heavy buckets on the same disk, and hence the large variance. However, HUBBA does well even in the bad case because it uses Bubba Opt1 to place the heaviest weights on different disks. Note that the values in figures 8 and 9 correspond to the case with small number of buckets, high skew, and progressively increasing number of disks. When a real-life application relation is hash declustered, the number of buckets is usually much larger than 100, in which case the load deviation would be much lower. Although not reported here, HUBBA showed a deviation of less than 1% with 1000 buckets.

Observe that the bad case distribution of weights does not affect the effectiveness of load balancing of classical Bubba and Bubba Opt1. This is to be expected because for these placement strategies, the bad case is just another permutation of the input weights. Since these strategies select weights in decreasing order, the input permutation does not matter.

5.2 Reorganization Cost

We now compare the four placement strategies with respect to the reorganization cost, which is computed as follows:

1. Let the initial degree of allocation of a relation be N. Without loss of generality, the relation hash buckets can be placed on disks numbered 0 through $N - 1$.

2. Apply each of the four data placement strategies to determine the initial placement of buckets to disks.

Number of Disks	Mean disk wt.	Standard Deviation			
		Classical	Bubba Opt1	Hash	HUBBA
4	25.0	0.00	0.00	5.99	1.31
5	20.0	0.13	0.13	6.40	1.93
6	16.6	0.16	0.16	4.28	2.90
7	14.2	0.09	0.09	14.67	2.12
8	12.5	0.13	0.13	3.96	2.97
9	11.1	0.46	0.46	3.33	3.10
10	10.0	0.80	0.80	3.41	3.03
11	9.09	1.06	1.06	3.33	3.80
12	8.33	1.25	1.25	3.62	3.86

Figure 9: Bad case load deviation in percent: $\theta = 0.2$, buckets = 100, $\alpha = 0.3$.

3. Change the degree of allocation to N'. Relation hash buckets can be placed on disks numbered 0 through $N' - 1$.

4. Apply each of the four data placement strategies to determine the new placement of buckets to disks.

5. For each data placement strategy, compare the allocation in steps 2 and 4 to determine the hash buckets that are assigned to different disks. These hash buckets have to be reorganized, i.e., moved across disks. The cumulative weights of these hash buckets gives the reorganization cost for the respective data placement strategies.

In our experiments, we assume that there are 16 disks in the system. The initial degree of allocation of the relation in consideration is 4. The degree of allocation is progressively changed from 5 to 16. Figures 10 and 11 show the reorganization cost for each of the four placement strategies. In these figures, we only show the results for the case where the new degree of allocation is greater than the old degree of allocation, i.e., $N' > N$. Interestingly, the interchanging of N and N' has no effect on the reorganization cost. In other words, given a pair of N and N', it does matter whether we consider N as the old degree of allocation and N' as the new degree of allocation; or vice versa; the reorganization cost is identical in both cases. We can make a number of observations from figures 10 and 11:

The hash and HUBBA placement strategies show sharp depressions at certain points in these graphs. At these points, the reorganization cost is as low as 48%, i.e., less than 50% percent of the data belonging to the declustered relation has to be moved across the disks to achieve effective load balance once again. Comparative reorganization costs for other placement strategies is as much as 95% at these points. The reason hash and HUBBA display such a behavior is explained in more detail in the section 6.2. Essentially, with hash placement, the movement of data displays a certain pattern when the degree of of allocation is changed. This pattern is best seen if we represent the reorganization under

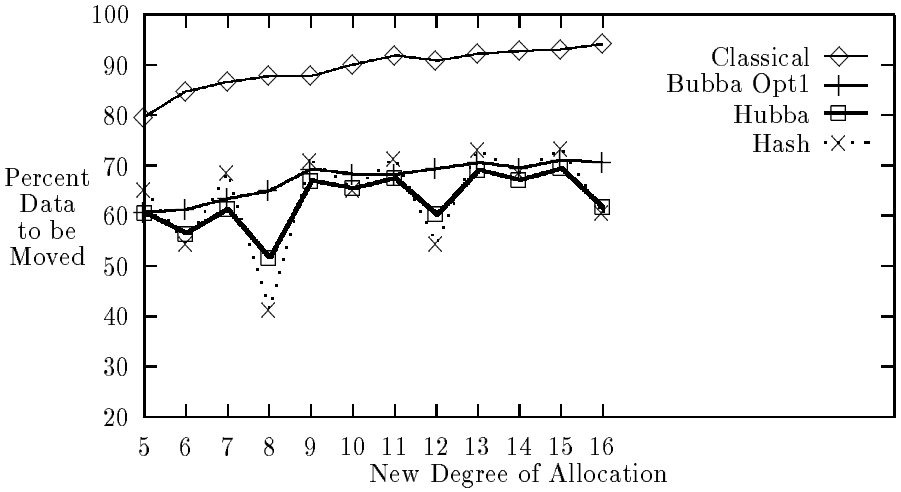


Figure 10: Percent of Data reorganized. $\theta = 0.2$, $N = 4$, Buckets $= 100$, $\alpha = 0.3$.

hash placement as bipartite graph. The two sets of nodes represent the initial set of N disks, and the new set of N' disks respectively. The edges represent the movement of at least one logical bucket from the source node to the destination node. The bipartite graph displays connected components under the hash placement strategy (see Section 6.2). For example, if $N = 4$, and $N' = 8$, then there would be 4 connected components in the bipartite graph. The number of connected components is equal to the greatest common divisor of N and N'. Thus, the greater the number of connected components in the bipartite graph, the lower the reorganization cost. Thus the reorganization cost is the lowest when the new degree of allocation is twice the old degree of allocation. Thus the depression in the figures 10 and 11 is the largest when $N' = 8$ (the bipartite graph has 4 connected components).

The reorganization costs of classical Bubba and Bubba Opt1 are represented by almost parallel lines. The difference between them constitutes the fraction of data in the heaviest N buckets that is not moved by Bubba Opt1. When the skew decreases from $\theta = 0.0$ to $\theta = 1.0$, the difference between the parallel lines narrows because the fraction of data in the heaviest N buckets decreases progressively. Observe in figure 11 that there is a small but finite gap between the two parallel lines. This gap is about 4%.

In summary, classical Bubba and Bubba Opt1 placement strategies are effective in balancing the load, especially when the variance among the bucket weights is large; however, they do not have a low reorganization cost. On the other hand, Hash is not effective in balancing the load when the variance is large; however, it possesses the low reorganization cost property. With the hybridization, HUBBA inherits the good properties of both Bubba and Hash: effective load balancing property from Bubba and low reorganization cost from hash.

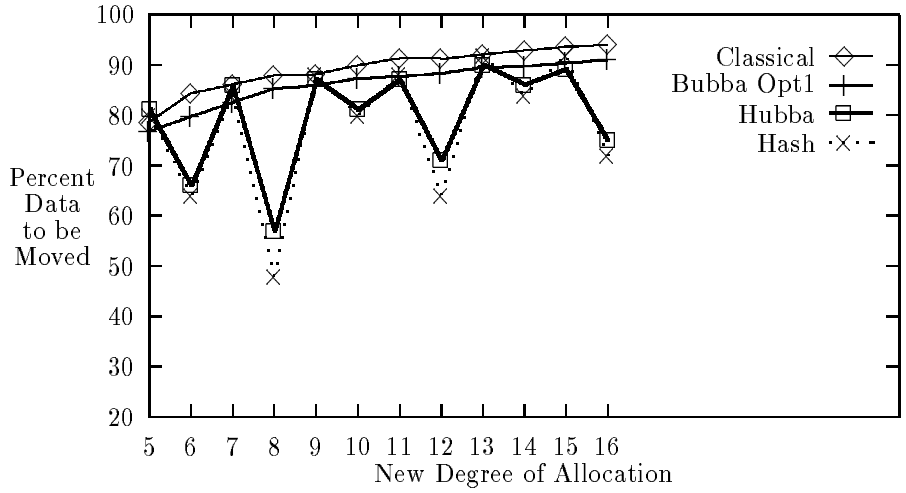


Figure 11: Percent of Data reorganized. $\theta = 1.0$, $N = 4$, Buckets $= 100$, $\alpha = 0.3$.

In order to provide a deeper and fuller understanding of the relationship between the data placement strategies and the reorganization cost, we will derive analytical models for the data placement strategies in the next section.

6 Analytical Modeling

6.1 Analytical model for Classical Bubba and Bubba Opt1 strategies

Without loss of generality, assume that the buckets are arranged in a decreasing order of weights. Denote by $p(M, N, i, \theta)$ the probability that a particular bucket would be placed on disk i when the bucket skew is θ and the number of buckets is M, and the number of disks is N. The approximation we make is that the number of buckets placed on a particular disk is inversely proportional to the heaviest bucket placed on that disk. Since the bucket weights have a Zipf-like distribution, we have, after normalization,

$$p(M, N, i, \theta) = \frac{1}{Z(M, i, \theta) * \sum_{k=1}^{N} \frac{1}{Z(M,k,\theta)}} \tag{1}$$

where $Z(M, k, \theta)$ is the weight of bucket k, and belongs to the Zipf-like distribution is given by [7]

$$Z(M, k, \theta) = \frac{1}{H_M^{(1-\theta)} * k^{1-\theta}} \tag{2}$$

where $H_M^{(1-\theta)}$ is the harmonic number of order $(1-\theta)$. When the the degree of allocation is changed to N' disks, the probability that a particular bucket would be placed on disk

i is $p(M, N', i, \theta)$. The probability that a particular bucket is not moved (i.e., reassigned to the same disk) during the reorganization is $p(M, N, i, \theta) * p(M, N', i, \theta)$. Therefore the probability that a bucket will be moved is

$$P^{\theta}_{moved} = 1 - \sum_{j=1}^{N} p(M, N, j, \theta) * p(M, N', j, \theta) \quad (3)$$

The fraction of data moved for classical Bubba is also P^{θ}_{moved}. However, the fraction of data moved for Bubba Opt1 is given by

$$P^{\theta}_{moved} * (1 - \sum_{k=1}^{N} Z(M, k, \theta)) \quad (4)$$

This is because the heaviest buckets assigned to disks 1 through N are not moved.

Validation of Equations 3 and 4

In table 1, we compare the errors in the analytical predications using equation 3 with respect to the experimental results for the fraction of the data moved by classical Bubba. The numbers in the tables were obtained as follows: equation 3 was used for each pair of numbers (N, N'), where N was set to 4, and N' was varied from 5 through 16. The result of equation 3 was compared with the respective experimental data and the error was noted. The *mean error* in the tables represents the error over the entire range of N' values. The *max error* was the maximum error for some particular value of N', and so was *min error*. Table 1 shows the errors when the number of buckets is 500 Note that the maximum error is only 8%. When the number of buckets is 100, the maximum error is only 6%. Due to lack of space, we do not include the table for this case here. Therefore, Equation 3 reasonably approximates the behavior of the classical Bubba data placement strategy.

In tables 2 and 3, we compare the analytical predictions using equation 4 and the experimental results for the fraction of data moved by Bubba Opt1 placement strategy. Table 3 presents the results for 100 buckets. With the exception of skew cases 0.7-0.9, the errors are within 13%. We have not been able to satisfactorily model the skew case 0.7 through 0.9. One of the reasons for the errors being large in these cases in that the number of buckets is quite small, in this case only 100. When the number of buckets is increased to 500, the errors are well within 14% for all the cases without exception as shown in table 2.

6.2 Analytical model for Hash placement

We now model the behavior of the hash placement strategy. As mentioned earlier in this section, with hash placement, the movement of data displays a recognizable pattern when the degree of of allocation is changed. This pattern is best seen if we represent the reorganization under hash placement as a bipartite graph. The bipartite graph has two sets of vertices S_1 and S_2. Set S_1 corresponds to the *old* set of disks $0 \ldots N-1$, and S_2 to the *new* set of disks $0 \ldots N'-1$. The edges in the bipartite graph are defined as follows: an edge between a vertex $i \in S_1$ and a vertex $j \in S_2$ exists if at least one record in old disk

Bucket skew (θ)	mean error (in %)	max error (in %)	min error (in %)
0.0	3.5	6.4	0.0
0.1	4.7	8.2	0.0
0.2	3.4	5.6	-2.8
0.3	3.5	4.9	0.0
0.4	2.6	4.4	0.0
0.5	2.6	4.1	0.0
0.6	2.2	3.6	0.0
0.7	2.3	3.5	0.0
0.8	1.2	3.0	-0.3
0.9	0.6	1.4	0.0
1.0	-0.3	0.3	-0.8

Table 1: Errors in the analytical modeling of classical Bubba. Number of buckets is 500.

Bucket skew (θ)	mean error (in %)	max error (in %)	min error (in %)
0.0	-0.2	2.8	-3.1
0.1	1.0	3.8	-1.3
0.2	0.4	4.8	-2.7
0.3	0.9	3.3	-1.0
0.4	0.3	6.9	-9.0
0.5	1.9	3.4	0.0
0.6	2.1	5.5	-0.4
0.7	3.9	11.2	-0.9
0.8	3.8	13.9	-0.9
0.9	0.5	5.5	-2.5
1.0	0.2	1.8	-1.9

Table 2: Errors in the analytical modeling of Bubba Opt1. Number of buckets is 500.

Bucket skew (θ)	mean error (in %)	max error (in %)	min error (in%)
0.0	-0.5	17.1	-5.1
0.1	0.1	7.3	-3.3
0.2	-1.9	0.0	-5.6
0.3	-1.1	3.2	-4.3
0.4	-2.6	4.4	-13.8
0.5	0.9	3.6	-1.4
0.6	1.5	13.7	-4.1
0.7	4.2	24.5	-3.2
0.8	10.5	57.7	-3.6
0.9	2.1	19.4	-3.9
1.0	0.1	3.7	-5.1

Table 3: Errors in the analytical modeling of Bubba Opt1. Number of buckets is 100.

i will have to be moved to the new disk j. With this representation, the patterns of data movement can be seen as connected components in the bipartite graph. All data movement is within the component and never across components. This data movement pattern can be used to explain the reorganization cost of the hash placement strategy. In the Appendix, we will give a rigorous proof (as theorem 1) for the existence of connected components in the bipartite graph. We now discuss its implications.

Theorem 1 *The bipartite graph is partitioned into equivalence classes E_k such that $(i,j) \in E_k$ if and only if $i \equiv j$ modulo $\gcd(N,N')$, where $i \in S_1$, $j \in S_2$, $|S_1| = N$, $|S_2| = N'$, and $k = 0, \ldots, \gcd(N,N') - 1$.*

Corollary 2 *If $(i,j) \in E_k$, then $i \bmod \gcd(N,N') = j \bmod \gcd(N,N') = k$, where $0 \leq k < \gcd(N,N')$.*

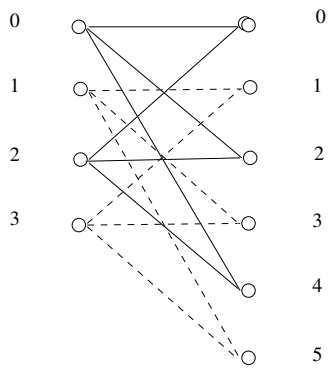
Corollary 3 *The number of equivalence classes is $\gcd(N,N')$.*

Corollary 4 *Each equivalence class is a fully connected bipartite graph.*

Corollary 5 *All equivalence classes have equal number of vertices from the sets $S1$ and $S2$. The $N' - N$ extra disks belonging to $S2$ are equally distributed among all the classes.*

Figure 12 shows an example of Theorem 1 when $N = 4$ and $N' = 6$. The $gcd(N,N') = 2$. Hence there are two connected components E_0 and E_1. Note that corollary 2 can be used to quickly determine the members of each component.

Theorem 1 can be used to explain the behavior of hash placement strategy. By corollary 5, all equivalence classes have equal number of vertices from both the sets $S1$ and $S2$. Therefore, the larger the number of components, the fewer the number of vertices in each



gcd(4,6) = 2

$E_0 = \{\{0, 2\}, \{0, 2, 4\}\}$

$E_1 = \{\{1, 3\}, \{1, 3, 5\}\}$

Figure 12: Illustration of Theorem 1

component. The fewer number of vertices from $S2$ implies that a larger percentage of the data on any disk belonging to the set $S1$ need not be reallocated. For example, let $N = 4$ and $N' = 6$. In this case, there are 2 connected components in the bipartite graph. Then the class E_0 is given by $\{\{0, 2\}, \{0, 2, 4\}\}$. In this case, data from disk 0 has to be relocated to disks 2 and 4. Therefore, two-thirds of the logical buckets allocated to disk 0 will now have to be moved. If $N' = 8$, then there are 4 connected components in the bipartite graph. Then E_0 is given by $\{\{0\}, \{0, 4\}\}$. This implies that only one-half of the logical buckets assigned to disk 0 will have to be moved to disk 4. Hence the larger the number of connected components, the lower the cost of reorganization. This is the reason for the occurrence of the sharp depressions in figures 10 through 11. Observe that larger the $\gcd(N, N')$, lower the fraction of data moved.

The fraction of data moved when the system is scaled up from N disks to N' disks can be computed by the fact that a bucket i does not move if $i \bmod N = i \bmod N'$. Therefore we have:

$$F(N, N') = 1 - \sum_{j=0}^{limit\_1} \sum_{k=j*limit\_2}^{limit\_3} Z(M, k, \theta) \tag{5}$$

where

$$limit\_1 = \left\lceil \frac{M * \gcd(N, N')}{N * N'} \right\rceil$$

$$limit\_2 = \frac{N * N'}{\gcd(N, N')}$$

$$limit\_3 \quad = \quad \min(j * limit\_2 + N - 1, M - 1)$$

Validation: We found a perfect agreement between equation 5 and the experimental data for the hash placement strategy, i.e., the mean, min and max errors were all 0%!.

6.3 Analytical model for HUBBA placement strategy

In this section, we will present the analytical model for the HUBBA placement strategy. Since HUBBA is a hybrid of hash and Bubba Opt1 placement strategies, the fraction of data moved is given in terms of equations 4 and 5. Define $\mathcal{BS}$ to be the set containing the first $\alpha * M$ heaviest buckets. HUBBA uses Bubba Opt1 to place buckets in $\mathcal{BS}$. Let $h(i)$ be a function that returns the ith heaviest bucket. Then the fraction of data moved is

$$F(N, N') = \sum_{i=1, i \notin \mathcal{BS}}^{M} Z(M, i, \theta) - \sum_{j=0}^{limit\_1} \sum_{k=j*limit\_2, k \notin \mathcal{BS}}^{limit\_3} Z(M, k, \theta) + f_{bubba} \qquad (6)$$

where

$$f_{bubba} = \begin{cases} P^{\theta}_{moved} * \sum_{i=N+1}^{\alpha * M} Z(M, h(i), \theta) & \text{if } \alpha * M > N \\ 0 & \text{otherwise} \end{cases}$$

The first term in equation 6 represents the fraction of data to be placed using the hash component of the HUBBA placement strategy. The second term, similar to equation 5, represents the fraction of the data not moved by the hash placement strategy. The third term, similar to equation 4, is the fraction of the data moved using the Bubba Opt1 component of the HUBBA placement strategy. Observe that hash placement is applied to the $(1 - \alpha)\%$ of the buckets, and Bubba Opt1 to $\alpha\%$ of the buckets.

Validation: Table 4 show the correspondence between the analytical computation of the fraction of the data moved using equation 8 and the experimental data for the HUBBA placement strategy. When the number of buckets is 100, the errors are less than 13% for most cases, and about 18% in a couple of cases. Due to lack of space, we do not include a figure here. When the number of buckets is 500 and $\alpha = 0.1$, the errors are less than 9% as shown in table 4. The errors are essentially due to equation 4. This shows that equation 6 is a fair approximation of the reorganization cost of the HUBBA placement strategy.

7 Conclusions

Data placement in parallel database systems is a critical factor in determining the performance of the system. Given the dynamic nature of database systems, the optimal placement of relations will change over time and this will necessitate a reorganization in order to maintain the performance of the database system at acceptable levels. The objective of this paper has been to study the impact of data placement strategies on the reorganization costs. In this paper, we focused on one particular aspect, i.e., given that the degree of allocation of a

Bucket skew (θ)	mean error (in %)	max error (in %)	min error (in %)
0.0	-1.0	4.8	-4.9
0.1	-0.8	3.2	-2.3
0.2	-1.4	0.6	-4.1
0.3	-0.4	2.2	-2.0
0.4	-1.4	0.4	-5.5
0.5	-0.1	1.2	-1.0
0.6	0.5	3.5	-1.0
0.7	1.0	8.6	-1.2
0.8	1.3	8.7	-0.7
0.9	0.5	3.9	-0.6
1.0	0.3	1.4	-0.9

Table 4: Errors in the analytical modeling of HUBBA. Number of buckets is 500, $\alpha = 0.1$.

relation changes, what impact does a data placement strategy have on the consequent reorganization costs? This is an important issue because the amount of data to be reorganized directly determines the impact on transactions during on-line reorganization.

An important conclusion we can make from this study is that the choice of a data placement strategy can have a significant impact on the reorganization costs, with associated implications on on-line reorganization. So far, data placement heuristics were designed with the express purpose of balancing the load. However, this paper shows that such a policy can be beneficial only in the short term. Long term database designs should factor in reorganization costs while making design choices.

A Appendix

A.1 Analytical Model for Hash Placement

Let us denote the hash placement function by $h(K;N) = K \bmod N$. When the degree of allocation changes from N to N', the hash placement function changes from $h(K;N)$ to $h(K;N')$. Then the data hashed into $0 \ldots N-1$ disks by $h(K;N)$ will have to be reorganized into $0 \ldots N' - 1$ disks to reflect the data placement due to $h(K;N')$. We can model the reorganization problem as a bipartite graph with two sets of vertices S_1 and S_2. Set S_1 corresponds to the *old* set of disks $0 \ldots N-1$, and S_2 to the *new* set of disks $0 \ldots N' - 1$. The edges in the bipartite graph are defined as follows: an edge between a vertex $i \in S_1$ and a vertex $j \in S_2$ exists if at least one record in old disk i will have to be moved to the new disk j.

Before we proceed to the main result (Theorem 1) of the subsection, we will need the following discussion. Suppose the hashing function is changed from $h(K;n)$ to $h(K;m)$, where n and m are relatively prime integers with $n < m$. Observe that $K = h(K;n) +$

$n\lfloor\frac{K}{n}\rfloor = h(K;m) + m\lfloor\frac{K}{m}\rfloor$. Rewriting this, we get $h(K;m) - h(K;n) = n\lfloor\frac{K}{n}\rfloor - m\lfloor\frac{K}{m}\rfloor$. We can interpret $h(K;m) - h(K;n)$, abbreviated as $f(K)$, as the "distance" between the new disk and the old disk for a record with key K. A non-zero value of the "distance" for a record with key K signifies that the record needs to be reorganized. The "distance" has no other physical significance. Therefore, we have:

$$f(K) = n\lfloor\frac{K}{n}\rfloor - m\lfloor\frac{K}{m}\rfloor$$

For example, if K is 5, and n is 3 and m is 8, then old hashing function $h(K;3)$ places this record on disk 2, whereas the new hashing function $h(K;8)$ places it on disk 5. The distance between these disks is 3. If K is 17, then the old hashing function places it on disk 2, and the new hashing function on disk 1. Therefore, the distance is -1 or, more appropriately, 7 (because -1 mod 8 is 7). Note that the distance can also be computed using $f(K)$.

We can ask two questions: (a) given a record with key K, what is its distance? (b)given a distance, which keys have this distance? (These questions have direct relevance in proving the necessary and sufficient conditions of theorem 1). The first question can be easily answered using the above equation. To answer the second question, the reader is referred to the example in figure 13, in which n is 3, and m is 8. The example in figure 13 tabulates keys and their distance. We can make the following observations:

1. Consecutive keys can be grouped into blocks of 3 (i.e., n), starting at 0. Keys in a block have the same distance. For example, keys 3 through 5 belong to a block and have a distance 3.

2. Blocks of keys differing in their distance by 1 are separated by 3 blocks, (or 9 keys). For example, the block with distance 1 begins with key 9, and the block with distance 2 begins with key 18. There are 9 keys between them, or equivalently, 3 blocks. This separation is called *shift* and is measured in blocks.

3. The distance for a key K is the same as the distance for the key K mod 24 (i.e., K mod $n * m$). For example, key 25 has the same distance as key 1.

4. The values for distance are between 0 and 7 (i.e., $0 \leq$ distance $\leq m - 1$).

These observations can be used to answer the second question we posed earlier. For example, to find a block of keys with distance 3, we can use observation 2: the block with distance 3 is $3*$*shift* blocks away from the block with distance 0, i.e., keys in the range 27 through 29 have a distance 3. (From observation 3, the keys in the range 3 through 5 also have a distance 3). This fact is generalized and expressed succinctly as lemma 1 below. Lemma 1 is used in theorem 1 to prove a necessary and sufficient condition.

As we can see from the above discussion, *shift* plays a critical role, for which we now derive a general expression. Let K_1 and K_2 be two keys whose distance differs by 1. Without

Key	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Distance	0	0	0	3	3	3	6	6	6	1	1	1	4	4	4

Key	15	16	17	18	19	20	21	22	23	24	25	26	27	$\cdots$
Distance	7	7	7	2	2	2	5	5	5	0	0	0	3	$\cdots$

Figure 13: An Example leading to lemma 1. Here $n = 3$, and $m = 8$.

loss of generality, let the distance for K_1 be greater than that of K_2. Therefore,

$$\begin{aligned} f(K_1) - f(K_2) &= 1 \\ n\lfloor\frac{K_1}{n}\rfloor - m\lfloor\frac{K_1}{m}\rfloor - n\lfloor\frac{K_2}{n}\rfloor + m\lfloor\frac{K_2}{m}\rfloor &= 1 \\ n(\lfloor\frac{K_1}{n}\rfloor - \lfloor\frac{K_2}{n}\rfloor) - m(\lfloor\frac{K_1}{m}\rfloor - \lfloor\frac{K_2}{m}\rfloor) &= 1 \\ nS - mr &= 1 \end{aligned}$$

where S is $\lfloor\frac{K_1}{n}\rfloor - \lfloor\frac{K_2}{n}\rfloor$, and r is $\lfloor\frac{K_1}{m}\rfloor - \lfloor\frac{K_2}{m}\rfloor$. We call S as the *shift* and is given by $(1+rm)/n$. Since *shift* is measured in blocks, and the block length is n, we require $(1+rm)$ to be an integral multiple of n, i.e., $(1 + rm) \bmod n = 0$. In lemma 1, we show that this is restriction is necessary.

We next present lemma 1. For ease of presentation, we assume that records with keys between 0 through shift$*n * m$ exist.

Lemma 1 *Let n and m be a pair of relatively prime integers with $n < m$. Let S be an integer called the* Shift *given by $S = (1+rm)/n$ where r is the smallest positive integer such that $(1+rm) \bmod n = 0$. Let $f(K) = n\lfloor\frac{K}{n}\rfloor - m\lfloor\frac{K}{m}\rfloor$. Then for any distance q, $0 \le q < m$, every K in the interval $nSq \le K < n(Sq+1)$ satisfies $f(K) = q$.*

Proof: Let $K = Sqn + i$, where $0 \le i < n$. Then

$$\begin{aligned} f(K) &= n\lfloor\frac{Sqn+i}{n}\rfloor - m\lfloor\frac{Sqn+i}{m}\rfloor \\ &= Sqn - m\lfloor\frac{Sqn}{m}\rfloor \\ &= Sqn - (Sqn - Sqn \bmod m) \\ &= (1+rm)q \bmod m \\ &= q \end{aligned}$$

We require that $(1 + rm) \bmod n = 0$. We show that if this is not the case, then $f(K) \neq q$ for some K in $[nSq, n(Sq+1))$. Let S' be the integer part of S. Then the intervals $[nSq, n(Sq+1))$ and $[n(S'q+1), n(S'q+2))$ overlap. Let $K = n(S'q+1)$ since it belongs to both the intervals. Then,

$$
\begin{aligned}
f(K) &= n\lfloor\frac{n(S^{'}q+1)}{n}\rfloor - m\lfloor\frac{n(S^{'}q+1)}{m}\rfloor \\
&= n(S^{'}q+1) - [n(S^{'}q+1) - (n(S^{'}q+1)) \bmod m] \\
&= S^{'}qn \bmod m + n \bmod m \\
&= q + n \bmod m \\
&\neq q
\end{aligned}
$$

□

In the example of figure 13, let $n = 3$ and $m = 8$. Then the distance q can take values in the range 0 through 7, i.e., $m - 1$. From the lemma, the shift is $S = 3$. Given a value of q, every K in the interval $[9q, (9q + 3))$ satisfies $f(K) = q$.

We next proceed to the main result (theorem 1) of this subsection. Theorem 1 shows the existence of the connected components in the bipartite graph and gives an easy method to compute the members of each of the connected components. Each connected component is an equivalence class in which there is an edge between every pair of nodes, each belonging to the two different node sets of the bipartite graph. The notation $i \equiv j$ modulo Z means that $i \bmod Z = j \bmod Z$. The implications of theorem 1 are discussed in section 3.2.1.

For ease of presentation, we assume that records with keys between 0 through shift$*N*N^{'}$ exist. This is neither a stringent requirement nor a crucial assumption. This purpose of the assumption is to allow the readers to clearly see the underlying structure of the bipartite graph.

Theorem 6 *The bipartite graph is partitioned into equivalence classes E_k such that $(i, j) \in E_k$ if and only if $i \equiv j$ modulo $\gcd(N, N^{'})$, where $i \in S_1$, $j \in S_2$, $|S_1| = N$, $|S_2| = N^{'}$, and $k = 0, \ldots, \gcd(N, N^{'}) - 1$.*

Proof: Without loss of generality, let $N < N^{'}$.

1. *Necessary Part:* Let $(i, j) \in E_k$, i.e., there exists an edge between vertices i and j. To show $i \equiv j$ modulo $\gcd(N, N^{'})$, i.e., $|i - j|$ is an integral multiple of $\gcd(N, N^{'})$. Since an edge exists, there exists a record with key K such that $j = i + N\lfloor\frac{K}{N}\rfloor - N^{'}\lfloor\frac{K}{N^{'}}\rfloor$. Write $N = \gcd(N, N^{'}) * N_1$, and $N^{'} = \gcd(N, N^{'}) * N_1^{'}$, where N_1 and $N_1^{'}$ are integers. Then

$$
\begin{aligned}
|j - i| &= |N\lfloor\frac{K}{N}\rfloor - N^{'}\lfloor\frac{K}{N^{'}}\rfloor| \\
&= \gcd(N, N^{'}) * |N_1\lfloor\frac{K}{N}\rfloor - N_1^{'}\lfloor\frac{K}{N^{'}}\rfloor| \\
&= \gcd(N, N^{'}) * p
\end{aligned}
$$

where p is an integer equal to $|N_1\lfloor\frac{K}{N}\rfloor - N_1'\lfloor\frac{K}{N'}\rfloor|$.

2. *Sufficient Part:* Let $i \equiv j \mod \gcd(N, N')$. Therefore $|j-i| = p * \gcd(N, N')$, where p is an integer such that $p * \gcd(N, N') < N'$ (because $0 \leq |j-i| < N'$). We need to find a record with key K such that this record moves between disks i and j, and the value of $|j-i|$ (i.e., the 'distance' for the record with key K) is constrained to be an integral multiple of $\gcd(N, N')$.

   Let $N = Q * n$ and $N' = Q * m$ where n and m are relatively prime, and Q is the product of the prime factors common to both N and N'.

$$\begin{aligned} |j-i| &= p * \gcd(N, N') \\ &= |n\lfloor\frac{K'}{n}\rfloor - m\lfloor\frac{K'}{m}\rfloor| * Q \\ &= (n\lfloor\frac{K'}{n}\rfloor - m\lfloor\frac{K'}{m}\rfloor) * Q \end{aligned}$$

   where $K' = K/Q$. Setting $q = |j-i|/Q$, we can use function $f(K)$ of Lemma 1. Therefore, records with keys K in the interval $[SqnQ, nQ(Sq+1))$ move between disks i and j where $|j-i|$ is constrained to be $p * \gcd(N, N')$.

□

References

[1] G. Copeland, W. Alexander, E. Boughter, and T. Keller. Data placement in Bubba. In *ACM SIGMOD*, 1989.

[2] D. J. DeWitt and J. Gray. Parallel database systems: The future of high performance database systems. *Comm. ACM*, 35(6):85–98, June 1992.

[3] J. Gray. *The Benchmark Handbook for Database and Transaction Processing Systems.* Morgan Kaufmann Publishers, Inc, 1991.

[4] K. A. Hua and C. Lee. An adaptive data placement scheme for parallel database computer systems. In *Proceedings of the 16th VLDB Conference*, pages 493–506, 1990.

[5] M.Y. Kim. Synchronized disk interleaving. *IEEE Trans. Computers*, C-35(11), 1986.

[6] M. Kitsuregawa and Y. Ogawa. Bucket spreading parallel hash: A new, robust, parallel hash join method for data skew in the super database computer. In *Proc. 16th Intl. Conf. VLDB*, pages 210–220, August 1990.

[7] D. E. Knuth. *The Art of Computer Programming: Volume 3. Sorting and Searching.* Addison-Wesley, 1973.

[8] S. T. Leutenegger and D. Dias. A modeling study of the tpc-c benchmark. *ACM SIGMOD 1993 Conference Proceedings*, 1993.

[9] M. Livny, S. Khoshafian, and H. Boral. Multi-disk management algorithms. In *ACM SIGMETRICS*, pages 69–77, 1987.

[10] A. L. N. Reddy and P. Banerjee. An evaluation of multiple-disk I/O systems. *IEEE Trans. Computers*, 38(12):1680–1690, december 1989.

[11] K. Salem and H. Garcia-Molina. Disk striping. In *Proc. 2nd Intl. Conf. on Data Engg.*, 1986.

[12] P. Scheuermann, G. Weikum, and P. Zabback. Adaptive load balancing in disk arrays. In *Proceedings of the 4th Intl. Conference on Foundations of Data Organization and Algorithms (FODO)*, 1993.

[13] P. Scheuermann, G. Weikum, and P. Zabback. Disk cooling in parallel disk systems. *Bulletin of the Technical Committee on Data Engineering*, 17(3):29–40, September 1994.

[14] P. G. Selinger. Predictions and challenges for database systems in the year 2000. In *Proc. 19th Intl. Conf. VLDB*, pages 667–675. Morgan Kaufmann Publishers, August 1993.

[15] G. Sockut and R. Goldberg. Database reorganization - principles and practice. *ACM Computing Surveys*, 11(4):371–395, Dec 1979.

[16] G. H. Sockut and B. R. Iyer. Reorganizing databases concurrently with usage: A survey. Technical Report TR 03.488, IBM, Santa Teresa Laboratory, San Jose, CA, June 1993.

[17] M. Stonebraker, R. Agrawal, U. Dayal, E. J. Neuhold, and A. Reuter. DBMS research at crossroads: The vienna update. In *Proc. 19th Intl. Conf. VLDB*, pages 688–692. Morgan Kaufmann Publishers, August 1993.

[18] P. Valduriez. Parallel database systems: Open problems and new issues. *Distributed and Parallel Databases: An International Journal*, 1(2):137–166, April 1993.

[19] G. Weikum, P. Zabback, and P. Scheuermann. Dynamic file allocation in disk arrays. In *Proceeding of the ACM SIGMOD Conference*, 1991.