Automatically Declustering Relations In A Multi-Disk Database*

B. Srikanth Oracle Corporation, California
sbellamk@us.oracle.com

S. Seshadri Bell Labs, Murray Hill, NJ 07974
seshadri@lucent.com

ABSTRACT

Declustering relations across multiple disks is a standard and popular mechanism for improving the performance of database systems. Range partitioning is a very commonly used technique for declustering relations in a multi-disk environment. Severe load imbalances on disks may arise if partition boundaries corresponding to the ranges of a relation are not chosen carefully. In this paper, we address the problem of automatically determining the boundaries of a partition corresponding to a range of the relation so that the load on disks is balanced. This greatly reduces the human administration required and is yet another small step towards "knobs-free" operation of database systems.

We use observed access frequencies on tuples or a range of tuples to guide us to a solution that balances load across the disks. We propose an optimal algorithm (that minimizes the load imbalance) for the above problem. However, the optimal algorithm can be very expensive for large instances of the problem. Therefore, we propose two cheaper heuristic solutions. Our experiments demonstrate that these heuristics provide solutions that are close to optimal unless the access frequencies are highly skewed.

We also consider the allied problem of migrating from an already partitioned database to a new partitioned database with minimum data movement. We present an efficient optimal algorithm for the above problem.

1. INTRODUCTION

Declustering is a very popular technique that exploits I/O parallelism to improve the performance of a database system. Many declustering strategies have been proposed for parallel or multi-disk database systems - range, hash, round-robin, Hybrid-Range Partitioning Strategy (HRPS) [5], and multi-attribute declustering techniques like BERD and MAGIC [6]. However to maximize the performance improvement, the load needs to be balanced across the multiple disks. Surprisingly, there has been very little work done on choosing partitions that would result in the above techniques balancing the load uniformly.

In this paper, we address the problem of automatically detecting partition boundaries for range and HRPS partitioning strategies based on the access frequencies of tuples. Range partitioning is extensively used by commercial database vendors (Tandem, Oracle etc.) and to the best of our knowledge, currently, there do not exist good tools that can automatically balance the load. Database administrators therefore often have to guess the partitioning values. A tool that could detect these automatically would enable databases to "auto-tune" themselves.

Our algorithms use access frequencies of tuples[1]. These access frequencies can be obtained from page access frequencies which are easy to maintain in a system. Let us call the sum of access frequencies of the tuples mapped to a partition, the *heat* of the partition. Let us assume that we wish to decluster a relation into R partitions. We consider two metrics for evaluating the performance of a partitioning scheme:

1. variance in the heats of R partitions,
2. skew in the heats of R partitions (or maximum heat partition amongst the R partitions)

The variance metric reflects the average variation in the heats of partitions while skew reflects the maximum variation. So, in essence, they try to minimize the deviation of the load caused by a partition from the average load. For each of these metrics, we develop an optimal algorithm (that minimizes the metric) that automatically detects partition boundaries for range partitioning. For a given metric, there could be more than one optimal solution. Therefore, a combined metric that considers both these metrics in some order is also useful. We considered the problem of finding amongst all the minimum variance partitionings the one with minimum skew. This basically would try to minimize variance and having minimized variance, then minimize skew to the extent possible. We outline an algorithm that minimizes variance and then minimizes skew amongst the partitions

*Work was done while authors were at IIT Bombay and was funded by a grant from Tandem Computers

[1]Access frequencies on a coarser granularity like a range of tuples will also suffice for all the techniques outlined in this paper.

ADVANCES IN DATA MANAGEMENT 2000
Krithi Ramamritham, T.M. Vijayaraman (Editors)
Tata McGraw-Hill Publishing Company Ltd.

with minimum variance. A symmetric problem of minimizing variance amongst all minimum skew partitionings is also considered and an algorithm for finding such a partition is presented.

Optimal algorithms for either individual metrics or combined metrics could however be expensive in their computational requirements and hence, we also propose two heuristics called the Greedy algorithm and the Cooling algorithm that are cheaper to compute. We conduct a performance study which indicates that unless the access frequencies are highly skewed, the variance and skew in the heats of partitions obtained by these heuristics are very close to those obtained by optimal algorithms. Our performance study also indicates that the computational requirements of the optimal algorithm are significantly more than those required by the heuristic solutions.

We also consider a variant of range partitioning strategy called Hybrid Range Partitioning Strategy (HRPS) and propose algorithms that automatically detect partitioning boundaries for HRPS strategy. The basic HRPS strategy range partitions a relation into a constant (say c) times the number of disks (also called fragments) and then uses round-robin assignment to assign exactly c of these fragments to a disk. The algorithm, being an hybrid of range and round-robin partitioning schemes, retains the advantages of range partitioning while inheriting the better load balancing and intraquery parallelism features of round-robin strategy. We will call the above strategy HRPS/RR to distinguish it from our variant of HRPS called HRPS/OPT. HRPS/OPT also range partitions the relation into c times the number of disks, but, instead of using round-robin assignment, it attempts to find the best way to distribute these fragments to disks so that load is perfectly balanced. We show that the problem of deciding the fragment assignment to disks to minimize variance or skew is NP-complete. We, therefore, present heuristic solutions to this allocation problem. Our experiments indicate that HRPS/OPT is able to balance load better than pure range partitioning strategies and HRPS/RR, thus indicating that HRPS/OPT is a viable choice in multi-disk database systems.

We also consider a related problem that of *repartitioning*, which aims to repartition an already partitioned relation with minimal data movement. Repartitioning may be needed for several reasons such as changes in user's access patterns, updates to the database and the addition of new disks. What constitutes an optimal partitioning for the present workload may not be the optimal one in future. This significantly affects the response time and throughput of the system. Updates to the database may result in a skewed distribution i.e., an imbalanced load and hence, a deteriorated query performance. Repartitioning is the viable solution in this scenario. Also, addition of new disks may call for repartitioning to exploit the increased parallelism available and to reduce the response times of the queries. The repartitioning problem can be decoupled into two parts:

- Determining the new configuration: Here the goal is to find new partitions that balance the load across disks. Any of our previously mentioned techniques can be used for this purpose.

- Optimal way of shifting to the new configuration: Assignment of newly found partitions to disks incurring minimum data movement is the problem out here. A solution to this problem is also presented in this paper.

The rest of the paper is organized as follows: We describe our system model and the assumptions in Section 2. In Section 3, we consider range partitioning strategy and develop algorithms that partition the relation in an optimal (or near optimal) manner. In Section 4, we consider HRPS strategy. The problem of assigning newly found partitions to disks in an optimal way is dealt in Section 5. We describe the results of our performance study in Section 6. We discuss related work in Section 7 and conclude in Section 8.

2. PROBLEM STATEMENT AND DEFINITIONS

In this section, we state the problem we wish to solve formally and introduce some definitions that will be used by our algorithms. The informal problem statement for the problem we solve is: Given a relation and access frequencies on the tuples of the relation and the number of disks on which the relation needs to be declustered, find an assignment of tuples to disks that balances the load across the disks. We do not address the problem of how to determine the degree of declustering of a relation as well as the identity of the disks on which a relation is to be declustered. This is an orthogonal problem addressed, for example, in [2, 8]. Therefore, we assume throughout that a relation is to be declustered across disks numbered 1 through R (in other words the degree of declustering is R).

We assume that access frequencies (or their estimates), also called *heats*, of tuples are available. The access frequencies could however be at a coarser granularity, for example, a range of tuples. Our algorithms are independent of the granularity of the ranges on which access frequencies are available. However, the granularity determines the effectiveness of our solutions to the partitioning problem. The finer the granularity, the better will be the load balancing at the expense of greater overheads incurred in collecting and using the access frequencies. We denote by a unit, the set of tuples (or a single tuple) on which access frequencies are available. We assume throughout that the units are numbered 1 through N.

We model the units as a finite ordered set U and the access frequency of a unit as a function *heat*, $heat(u) \in Z_0^+$, for each $u \in U$. We want to partition U into R disjoint sets $U_1, U_2, \ldots, U_R$. The heat of a partition is defined as the sum of heats of units it contains, i.e.,

$$heat(U_i) = \sum_{u \in U_i} heat(u)$$

The optimum heat of a partition, *OptHeat*, used later by our algorithms, is defined as

$$OptHeat = \frac{\sum_{u \in U} heat(u)}{R}$$

Note that $U_1, U_2, \ldots, U_R$ as defined above captures any particluar partitioning strategy. Range partitioning additionally requires that for any i, j $\in U$, if i, j $\in U_m$ and $i < j$, then for any k, $i < k < j$, $k \in U_m$.

We consider two metrics for determining the efficiency of a partitioning strategy. The first one is the variance in the heats of partitions as given by the formula:

$$\frac{\sum_{i=1}^{k} (heat(U_i) - OptHeat)^2}{R}$$

The second metric is skew in the access frequencies of partitions which we define as the maximum value amongst the heats of the partitions $U_1, U_2, \ldots, U_R$.

3. AUTOMATIC DETECTION OF PARTITION BOUNDARIES FOR RANGE PARTITIONING

We present in this section, three algorithms for determining partition boundaries when the relation is to be range partitioned. First, an algorithm based on *dynamic programming* that is optimal with respect to minimizing variance is presented. This algorithm is then modified to find optimal partitions minimizing the skew. The dynamic programming solution is however computationally very expensive and therefore we propose two heuristic algorithms - *Greedy* and *Cooling* algorithms. The greedy algorithm, in fact, finds optimal partitions with skew being the optimization metric, but is not optimal when variance is the optimization measure. The *Cooling* algorithm is computationally the least expensive one but is not optimal for either of the metrics. However, as we will see in Section 6, partitions produced by the heuristic solutions are almost as good as those produced by the optimal algorithm.

3.1 Dynamic Programming Algorithm

The problem of finding an optimal range partitioning of U can be solved by employing the *dynamic programming* strategy. The solution involves making a sequence of decisions - with each decision determining the boundaries of a partition. Let $C_{1,m,r}$ denote the minimum cost of partitioning units from 1 to m into r partitions. This cost is the contribution of r partitions to the overall variance or skew. Stated formally,

- when *variance* is the optimization criterion

$$C_{1,m,r} = \min_{1 \le k \le m-1} \{C_{1,k,r-1} + C_{k+1,m,1}\}, r > 1 \quad (1)$$

$$C_{i,j,1} = \left(\sum_{t=i}^{j} heat(t) - AveHeat \right)^2, with j \ge i \quad (2)$$

- when *skew* is the optimization criterion

$$C_{1,m,r} = \min_{1 \le k \le m-1} \{\max(C_{1,k,r-1}, C_{k+1,m,1})\}, r > 1 \quad (3)$$

$$C_{i,j,1} = \sum_{t=i}^{j} heat(t), j \ge i \quad (4)$$

Equations 1 and 3 when solved for $C_{1,N,R}$ result in partitions that are optimal with respect to variance and skew respectively. We first compute $C_{i,j,1}$ for each possible i, j values and use them to compute $C_{i,j,2}$ using equations 1 or 3 depending on the optimization metric. Then, using $C_{i,j,2}$ and $C_{i,j,1}$, we find $C_{i,j,3}$. We repeat this process till we get $C_{1,N,R}$. To construct the optimal partitioning, we store the decision (the value of k) made at each step.

Computing $C_{i,j,1}s$ takes $O(N^2)$ time as i and j range from 1 to N. Computing $C_{1,j,k}, 1 < k \le R$ requires $O(RN^2)$ time. Hence, the complexity of this dynamic programming solution becomes $O(RN^2)$. While $O(N^2)$ space is needed to store $C_{i,j,1}s$, $O(RN)$ space is needed for $C_{1,m,r}$. Assuming, $R < N$, the algorithm requires $O(N^2)$ space.

Our experimental results indicate that algorithms that minimize variance also tend to have small skew while the converse is not always true. So, the administrator may choose variance as the metric. Further more, there could be more than one optimal solution for a given metric (say variance). Therefore, we considered the problem of finding amongst all minimum variance partitionings, the one with minimum skew. This basically would try to minimize variance and having minimized variance, minimize skew to the extent possible.

We now outline how the problem of finding a partitioning with minimum skew amongst all partitionings having minimum variance can be solved. The reason several partitionings can have same variance is that there may exist more than one k value that results in same contribution to the overall variance during the computation of $C_{1,j,r}$, for r>2 using equation 1. We can obtain such a minimum skew partitioning of the relation by adding the following rule to Equation 1: Choose k to minimize $\max\{C'_{1,k,l-1}, C'_{k+1,m,1}\}$, where $C'_{i,j,l}$ gives the maximum heat of a partition when units from i to j are partitioned into l ranges.

The symmetric problem of minimizing variance amongst all minimum skew partitionings can also be considered and a solution analogous to the solution above can be used.

3.2 Greedy Algorithm

The optimal algorithm based on dynamic programming strategy is very expensive in terms of both space and time as N can be very large. Our experiments also confirm this fact as can be seen in Section 6. This motivated us to develop faster algorithms. Here, we present one that is based on *greedy* strategy. Greedy approach does not provide an optimal solution when variance is the optimization metric but does provide an optimal solution when the optimization metirc is skew.

When skew is the optimization metric, we can restate the problem as: Find a partition of U into disjoint sets $U_1, U_2, \ldots, U_k$, such that

$$heat(U_i) \le H, \forall 1 \le i \le k \quad (5)$$

where H is the least possible such value. H can be interpreted as the maximum heat a partition can have. The basic idea of this algorithm is to consider tuples in the increasing order of partitioning key values and to pack as many tuples as possible into a partition before proceeding to the next partition. An *ideal* partitioning is one in which all partitions have equal heat (which is *OptHeat*) as it balances load equally among the disks of the system. But, an *optimal* partitioning for a given work load may not always result in skew equal to *OptHeat*. We first try to find an ideal partitioning i.e., start at $H = OptHeat$. If we are successful (condition 5 is satisfied), we have achieved perfect load balancing. If not, we increment H and repeat the process. We do this till we

find partitions satisfying the condition 5. We prove later that the partitions thus obtained are optimal when skew is the optimization metric.

The above strategy can be optimized by using binary search in the range [OptHeat, TotalHeat], where OptHeat is the optimum heat of a partition in the ideal case and TotalHeat is the sum of the heats of all the units. Note that condition 5 can be trivially satisfied with $H = TotalHeat$ and $OptHeat$ is the least H value that can satisfy this optimality condition. Rather than incrementing H by 1 each time we fail to partition the relation satisfying (5), we double H and try again. The algorithm (given in Figure 1) is recursive in nature and needs to be invoked as Greedy(optHeat, TotalHeat) to find the optimal solution.

Algorithm: Greedy(leftLimit, rightLimit)

```
begin
    if (leftLimit == rightLimit) then
        return
    H ← (leftLimit+rightLimit)/2
    foundPartitions ← AccumulateTuples(H)
    if (foundPartitions) then
        call Greedy(leftLimit, H)
    else
        call Greedy(H, rightLimit)
end
```

Figure 1: Greedy Algorithm

The function $AccumulateTuples(H)$ used in the algorithm is outlined in Figure 2. This function stuffs as many tuples as possible into a partition till its heat exceeds H. For the simplicity of presentation, we assume that the heat of a single tuple does not exceed H. If it does, we can change the function $AccumulateTuples$ to return FALSE. It returns *true* if partitions satisfying the condition 5 can be obtained or else, returns *false*. As searching the whole range

---

Algorithm: AccumulateTuples(H)

```
begin
    for i = 1 to R-1 do
        push as many tuples as possible into partition
        P_i without exceeding its heat beyond H
    put remaining tuples in P_R
    if (heat(P_R) > H)
        return FALSE
    else
        return TRUE
end
```

Figure 2: The AccumulateTuples algorithm

[OptHeat, TotalHeat] is computationally expensive, we reduce the search space using the following approach: Start finding partitions at $H = OptHeat$. If we are successful, we don't have to proceed further as we have achieved perfect load balancing. If not, try with $2H$, then $4H$ and so on till we are successful in obtaining partitions. We then limit our search space to [last_unsuccessful_Hvalue, current_successful_Hvalue] and invoke the Greedy algorithm as Greedy(last_unsuccessful_Hvalue, current_successful_Hvalue).

We now prove that the greedy algorithm is indeed optimal when skew is the optimization metric.

3.2.1 Proof of Optimality of Greedy Strategy

Before proving the optimality of GREEDY algorithm, we state and prove the following lemma.

LEMMA 3.1. *If there exists a range partitioning of the relation with skew less than or equal to x, then AccumulateTuples(x) will find a partitioning with skew less than or equal to x.*

PROOF. Let $A = A_1, A_2, \ldots, A_k$ be the partitioning of the relation with skew less than or equal to x and let $B = B_1, B_2, \ldots, B_k$ be the partitioning obtained by AccumulateTuples(x). We show that the heat of each partition B_i is less than or equal to x.

AccumulateTuples function ensures that the heats of partitions from 1 through $k-1$ are less than or equal to x. We, then, have to show that the heat of the last partition B_k is also bounded by x. Since AccumulateTuples() packs as many tuples as possible into a partition before proceeding to next partition, the right boundary of each partition in A is less than or equal to the corresponding one in B i.e., $right(A_i) \leq right(B_i)$ where $right(P)$ denotes the right boundary of a partition P. This is trivially true for $i = 1$. Now, consider A_2 and B_2. The right boundary of partition A_2 can never be greater than that of B_2. If it were, then we could have packed tuples from $right(B_2)$ to $right(A_2)$ into B_2 without increasing its heat beyond x. This is because, the left boundary of B_2 is greater than or equal to that of A_2 and $heat(A_2) \leq x$. As AccumulateTuples packs as many tuples as possible into a partition before moving to next partition, it will not miss out tuples to be packed in the current partition. Therefore, $right(A_2) \leq right(B_2)$. By repeating the same argument, it is evident that the right boundary of A_{k-1} is less than or equal that of B_{k-1}. Hence, the last partition B_k will be a subset of A_k making its $heat \leq x$ since $heat(A_k) \leq x$, thus proving the lemma. □

THEOREM 3.1. *Greedy algorithm is optimal when skew is the performance metric.*

PROOF. We also prove this theorem by contradiction. Suppose that Greedy algorithm terminates at $H = M_G$ and that there exists an optimal partitioning of the relation with $H = M_G - k$. Since greedy strategy uses binary search in the search space, it would have tried to partition the relation with $H = M_G - 1$ and was unsuccessful. That is, AccumulateTuples would have been called with $M_G - 1$ as argument and it returned FALSE. Since an optimal partitioning of the relation exists with skew less than or equal to $M_G - 1$, by **Lemma 3.1**, AccumulateTuples($M_G - 1$)

would have been successful – resulting in a contradiction. Hence proving that Greedy algorithm is optimal when the performance metric is skew. □

3.3 Cooling Algorithm

We now present another heuristic called "Cooling Algorithm". The basic idea of this algorithm is to start from some random partitioning and refine it by cooling *hotter* ranges. Cooling a partition involves moving tuples from its boundaries to the neighboring partitions.

The initial partitioning (range) can be chosen arbitrarily. The partitions of this initial partitioning may not be well balanced with respect to load. We then adjust the boundaries of partitions such that the heats of partitions are approximately the same. This is achieved by cooling hotter partitions. Partitions are considered for cooling in the decreasing order of their heats. Let the hottest partition be A. For cooling this partition, we have to decide the direction (in other words the neighbor to the left or right, if they exist) in which tuples of this partition should be sent first. We calculate the average heats of partitions to the left and right of A, if they exist and the tuples are first sent to the direction having smaller average heat. Tuples can then be sent to the other direction depending on the configuration.

The configuration can be one of the three cases as shown in figure 3. A is the current partition being cooled. In the first case, tuples are moved to the side with smaller average heat until its average exceeds OptHeat. Then, we move tuples to the other side and repeat the algorithm for the two subranges. In Case(2) of Figure 3, we move the tuples to the left (since its average heat is smaller than OptHeat) until its heat exceeds OptHeat or heat of A becomes zero. In the former case, the algorithm is repeated for the two subranges while in the latter case, the algorithm is repeated on the entire set of partitions. The last case is similar to the second one except for the reversal of directions. Boundary conditions (when A is the extreme left or right partition) are ignored for simplicity but they can be handled easily. The algorithm Cooling(I, J) which cools partitions from I to J is given in Figure 4. To find partitioning of the relation, we invoke it as Cooling(1, R).

It is assumed that the function Average(left, right) returns the average heat of partitions in [left, right]. The optimum heat, $OptHeat$, is obtained by calling Average(1, R). A call to the function moveTuplesToLeft(I, K) moves tuples from a given partition K to its left neighbor. It returns when the average heat of partitions to the left of K i.e., [I, K-1] goes beyond OptHeat or the heat of K becomes zero. moveTuplesToRight(J,K) works similarly.

The worst case complexity of this algorithm is $O(RN)$. This is because, in the worst case, each unit is considered at most R times and there are N units.

Algorithm: Cooling(I, J)

```
find the hottest partition in [I, J] and let it be A
leftAve ← Average(I, A-1)
rightAve ← Average(A+1, J)
if (leftAve < OptHeat) and (rightAve < OptHeat)
                    /** case (1) of figure 3 **/
    moveTuplesToLeft(I, A)
    moveTuplesToRight(J, A)
    Cooling(I, A-1)
    Cooling(A+1, J)
else if (leftAve < OptHeat) and (rightAve > OptHeat)
                    /** case (2) of figure 3 **/
    moveTuplesToLeft(I, A)
    leftAve ← Average(I, A-1)
    if leftAve > OptHeat then
        Cooling(I, A-1)
        Cooling(A, J)
    else
        Cooling(I, J)
else the same steps as in case (2) except a reversal
    in the direction of tuple movement
                    /** case (3) of figure 3 **/
```

Figure 4: The Cooling Algorithm

4. THE HYBRID-RANGE PARTITIONING STRATEGY

In this section, we consider HRPS strategy which has the advantages of range partitioning in focusing point and range queries to minimum number of disks and load balancing/intra-query parallelism characteristics of hash and round-robin declustering strategies. It has been proved in [5] that HRPS balances load better than conventional partitioning strategies. HRPS range partitions the relation into large (some multiple of degree of declustering) number of fragments (or ranges) and distributes them across disks in a round-robin fashion. Instead of applying simple range partitioning, we use the algorithms described in previous sections to obtain the fragments. For this, we just have to run those algorithms with the degree of declustering parameter set to a constant times the actual degree of declustering desired. We do not consider in this paper the question of how to determine that constant. In [5], the authors propose to assign fragments to disks in a round-robin manner. We call this approach HRPS/RR. We also consider another approach in which the assignment of fragments to disks is done optimally by considering all possible assignments. We call this approach HRPS/OPT. We will now show that optimally assigning fragments to disks is NP-complete and then outline, how well known heuristics can be used for HRPS/OPT.

4.1 Some NP-Complete Results

It turns out that the assignment of fragments to disks in an optimal way is NP-Complete under both definitions of optimality viz. minimizing skew and minimizing variance.

4.1.1 Minimizing the Skew

If the optimality condition were to minimize the skew, then the partitioning problem can be mapped to the well-known NP-Complete *multi-processor scheduling* problem [4] in which, given a set of processors and a set of tasks of a certain length, the question is to schedule these tasks on processors such that all tasks are completed by a deadline. Note that minimizing the skew is equivalent to minimizing

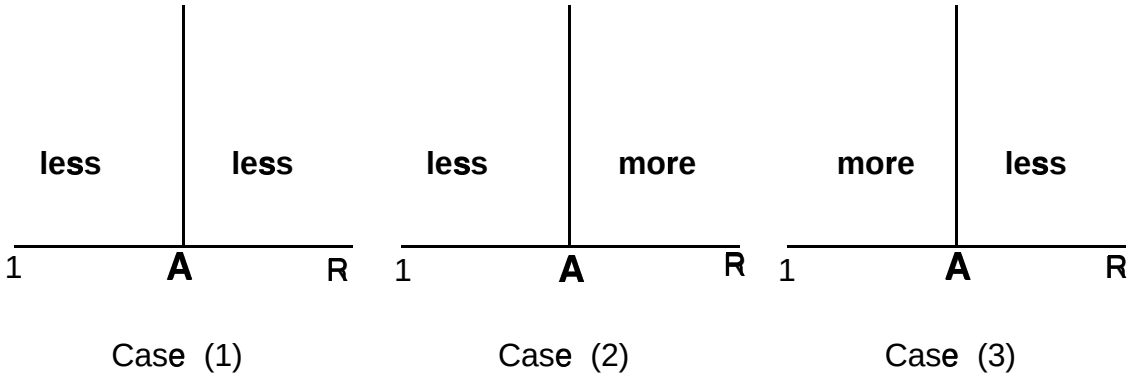
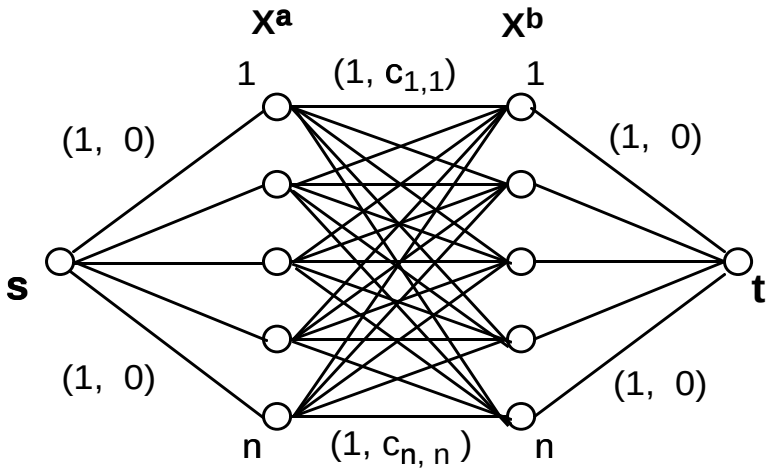


Figure 3: Three possibilities during Cooling Algorithm



(i, j) :: (capacity, cost)

$c_{r,p}$: cost of assigning partition r to processor p

Figure 5: Underlying graph of the Assignment Problem

the maximum heat of a partition. Tasks in the scheduling problem correspond to logical fragments and their length to the heat of corresponding fragment. The number of processors to which these tasks are assigned corresponds to the *degree of declustering* of the relation. Finally, the deadline corresponds to the maximum heat a partition can have. It is clear from the above mapping that the partitioning problem is same as multi-processor scheduling problem and hence is NP-Complete.

4.1.2 Minimizing the Variance

This problem can be mapped to *minimum sum of squares* problem which is also known to be NP-Complete [4]. The formal description of minimum sum of squares problem is as follows:

4.1.2.1 Instance: .

Finite set A, a size $s(a) \in Z^+$ for each $a \in A$, positive integers $K \leq \mid A \mid$ and J.

4.1.2.2 Question: .

Can A be partitioned into K disjoint sets $A_1, A_2, \ldots, A_K$ such that

$$\sum_{i=1}^{K} \left(\sum_{a \in A_i} s(a) \right)^2 \leq J \qquad (6)$$

We reduce the minimum sum of squares problem to our partitioning problem as:

elements of set A	$\rightarrow$	logical fragments
size of an element a	$\rightarrow$	heat of the corresponding fragment
K	$\rightarrow$	the degree of declustering

Our goal is to minimize the *variance* in the heats of partitions. The decision version[2] of this problem is: Find partitions such that

$$\sum_{i=1}^{K} \left(\sum_{a \in A_i} heat(a) - AveHeat \right)^2 \leq H.$$

We will transform the above equation to match the equation 6 by expanding the l.h.s. and determining the constant J. This constant J can be $H - H'$ where H' is obtained from the following reduction:

Let us denote $\sum_{a \in A_i} heat(a)$ by v. Then, l.h.s. of the above equation becomes

$$\begin{aligned}
&\equiv \sum_{i=1}^{K} (v - AveHeat)^2 \\
&\equiv \sum (v^2 + AveHeat^2 - 2 * AveHeat * v) \\
&\equiv \sum v^2 + \sum AveHeat^2 - 2 * AveHeat * \sum v \\
&\equiv \sum v^2 + \sum AveHeat^2 - 2 * AveHeat * K * AveHeat \\
&\equiv \sum v^2 + H', \text{where } H' \text{ is the constant given by} \\
&\quad H' = \sum AveHeat^2 - 2 * AveHeat * K * AveHeat
\end{aligned}$$

From this mapping it is clear that, given an optimal algorithm for partitioning problem, we can construct one for minimum sum of squares problem. Given that minimum sum of squares problem is NP-Complete, partitioning problem is also NP-Complete.

4.2 Heuristics for NP-Complete Problems

We now present some heuristic solutions for the above partitioning problems. When the optimization metric skew minimization, well-known LPT heuristic [4] can be used. In this heuristic, the fragments are sorted in descending order of their heats, first. Then, they are considered in that order and each fragment is assigned to one of the k partitions. The rule is to assign the current fragment to a partition having minimum heat. The heat of a partition is the sum of heats of fragments it contains. LPT strategy provides better solution compared to other known heuristics for the multiprocessor scheduling problem.

When the optimization measure is variance, we can either use *best fit*, or *best fit decreasing* heuristic [4]. In the *best fit* strategy, a fragment is assigned to a partition where it fits the best[3]. In the *best fit decreasing* strategy, we consider fragments in decreasing order of their heats and as in *best fit* strategy, a fragment is assigned to the best fit partition. We will evaluate the efficiency of these heuristics in Section 6.

5. THE REPARTITIONING PROBLEM

In this section, we turn our attention to the problem of migrating a table from an existing partitioning to a new partitioning with minimal data movement.

Let R=$\{1, 2, 3, \ldots, m\}$ denote the set of new partitions and P=$\{1, 2, 3, \ldots, n\}$ be the set of disks and let $m = n$. We assume existence of a cost function COST defined as:

$$COST : R \times P \rightarrow Z_0^+$$

i.e. $COST(r, p)$ gives the cost of assigning partition r to processor p. This cost which is the data movement required is a function of the number of tuples of partition r not contained at disk p (as per the current partitioning).

The repartitioning problem is to find a permutation π, defining an assignment $i \rightarrow \pi_i$ (partition i to disk π_i), with

[2] We have omitted denominator from the expression for variance as it is a constant

[3] The best fit partition is one which has $\mid heat\_of\_partition - OptHeat \mid$ minimum.

Algorithm → Distribution ↓	Greedy	Cooling	DP_SK	DP_VAR
Uniform	1250.00	1250.00	1250.00	1250.00
Normal	1252.00	1253.00	1252.00	1252.00
Zipfian (skew=1)	1305.00	1392.00	1305.00	1305.00
Zipfian (skew=5)	3859.00	4549.00	3859.00	3859.00

Table 1: Comparison of the skew in heats of partitions for 4 disks 1000 units

Algorithm → Distribution ↓	Greedy	Cooling	DP_SK	DP_VAR
Uniform	0.00	0.00	0.00	0.00
Normal	2.00	5.00	2.00	2.00
Zipfian (skew=1)	7821.00	28462.00	7925.00	2466.00
Zipfian (skew=5)	2330345.00	3636275.00	2304920.00	2270833.00

Table 2: Comparison of the variance in heats of partitions for 4 disks 1000 units

minimum cost as given by

$$\sum_{i=1}^{n} COST(i, \pi_i).$$

The problem of optimally assigning new partitions to disks to minimize data movement can be modeled as a *matching* problem in *bipartite graphs*. Given an undirected bipartite graph $G = (X^a \cup X^b, A)$, and a cost c_k associated with each edge $a_k \in A$, we want to find a perfect matching[4] of G with minimum cost. This problem is also referred to as the Assignment Problem in the literature. The sets X^a and X^b correspond to partitions and disks respectively. The cost of an arc (r, p), $r \in X^a \& p \in X^b$, is the data movement required when partition r is assigned to disk p. When $\mid X^a \mid = \mid X^b \mid = n$ and the graph is complete, which holds in our case, the matching problem can be solved using *network flow* methods of [3].

The underlying graph[5] for the assignment problem is shown in figure 5. An artificial source vertex s with out-edges $(s, x_i), \forall x_i \in X^a$, having unit capacity and zero cost, and sink vertex t with in-edges $(x_i, t), \forall x_i \in X^b$, also with unit capacity and zero cost are added to the given bipartite graph. Arcs of A have unit capacity. The maximum-flow minimum-cost path from s to t would have a flow value n and a nonzero flow (of value 1) in those arcs of A with minimum summed cost. These arcs would form a minimum cost perfect matching of G. The algorithm for solving this assignment problem [1] is referred to as *Hungarian* method. It has a polynomial complexity of $O(n^3)$.

Though X^a and X^b are assumed to be equal in size for simplicity of exposition, it is easy to extend the algorithm to the case where they are not. Thus, we can also change the degree of declustering of the relation.

The costs of edges in A can be determined given the initial and final partitioning of the relation and the old assignment. This is easy if the initial and final partitioning strategy is range partitioning. In this case, it boils down to the question of determining the number of tuples of a partition are already present at a particular site (or the non-overlapping part), which is nothing but the selectivity of a range query. This can be estimated using a sampling based approach as in [7].

[4]A *perfect matching* is a matching in which every vertex is matched to some other vertex

[5]The edges are assumed to be directed from left to right though they are not shown in the figure.

6. EXPERIMENTS

To compare the relative performance of algorithms proposed, we implemented them as stand-alone programs in 'C' language and ran the experiments on Sun SPARCstation running Solaris. As the performance of these algorithms critically depends on the skew in access frequencies of tuples, we synthetically generate various skewed distributions and test the algorithms. We considered the following distributions:

- Uniform,
- Normal, and
- Zipfian with skew parameters skew=1 and skew=5. The zipfian distribution with skew=5 is more skewed than one with skew=1.

We varied the number of units on which access frequencies are available (N) from 1000 to 10,000 and the number of disks (P) from 4 to 64. The algorithms are compared on the basis of following parameters:

- skew in the heats of partitions,
- variance in the heats, and
- time taken by the algorithms.

First, we compare the performance of algorithms for range partitioning. Later, we give the performance study when the relation is partitioned using hybrid-range partitioning strategy.

6.1 Performance of the Algorithms for Range Partitioning

Algorithms *Greedy, Cooling, DP_SK,* and *DP_VAR* stand for greedy, cooling, dynamic programming (with skew as optimization measure) and dynamic programming (with variance as optimization measure) algorithms described in Section 3 respectively.

Algorithm → Distribution ↓	Greedy	Cooling	DP_SK	DP_VAR
Uniform	80.00	80.00	80.00	80.00
Normal	80.00	84.00	80.00	80.00
Zipfian (skew=1)	381.00	412.00	381.00	381.00
Zipfian (skew=5)	3859.00	4310.00	3859.00	3859.00

Table 3: Comparison of the skew in heats of partitions for 64 disks 1000 units

Algorithm → Distribution ↓	Greedy	Cooling	DP_SK	DP_VAR
Uniform	121.00	5.00	5.00	5.00
Normal	98.00	5.00	4.00	4.00
Zipfian (skew=1)	21897.00	14846.00	12732.00	1887.00
Zipfian (skew=5)	237199.00	284772.00	234072.00	227077.00

Table 4: Comparison of the variance in heats of partitions for 64 disks 1000 units

6.1.1 Study on the Skew of the algorithms

Table 1 gives the skew in heats of partitions produced with number of disks P=4 and number of units N=1000. As expected, the skew in heats of partitions obtained increases as skew in access frequencies increases. Furthermore, Greedy, DP_SK and DP_VAR produce identical results. Greedy and DP_SK are provably optimal for this metric, but surprisingly, DP_VAR also produced optimal results. As we will see later, it does not always do so but, will be very close to optimal. This suggests that minimizing the variance has a benevolent side effect of minimizing the skew most of the time. The Cooling algorithm produces partitions close to optimal for small skews but, produces inferior partitions for higher skew.

6.1.2 Study on the Variance of the algorithms

Table 2 shows the variance in the heats of partitions produced with number of disks P=4 and number of units N=1000. In this case, DP_VAR is optimal while Greedy and DP_SK are close to DP_VAR for low skew, much worse for medium skew (Zipfian with skew =1) and again, pretty close when the skew is very high. The running times of Greedy and Cooling algorithms are significantly smaller than those of DP_SK and DP_VAR (by a factor of 100 to 1000). We also ran the above experiments by varying N and P. Table 3 gives the skew in the heats of partitions obtained when P=64 and N=100, and Table 4 shows the variance for the same configuration. An interesting thing to note here is that Cooling algorithm performs much closer to the optimal DP_VAR algorithm than Greedy algorithm does when variance is the optimization metric.

6.1.3 Study of the Time taken by algorithms

The time taken by various algorithms when number of disks P is 16 and number of units N is 10000 is tabulated in Table 5. It is evident from the table that dynamic programming algorithms take more time than greedy and cooling algorithms. Cooling algorithm is the fastest of all with the Greedy algorithm a close second. This makes them suitable in dynamic environments where reorganizations should not be time consuming.

6.2 Performance of algorithms for HRPS

We now compare the performance of hybrid-range partitioning strategies with plain range partitioning strategies to see whether they improve the variance and skew over plain range partitioning strategies. The range partitioning algorithm considered was *DP_VAR* as it is optimal when the optimality metric is *variance* and is close to optimal even when *skew* is the optimization metric. We also consider a partitioning strategy that partitions in any manner (not necessarily range), but tries to minimize variance and skew. This is the *LPT* algorithm (an heuristic algorithm since the optimal assignment is known to be NP-complete) that partitions the relation in a non-range way and produces near optimal solutions. The number of logical fragments was taken to be five times the number of disks. Logical fragments are obtained through optimal range partitioning algorithm, namely *DP_VAR*. Two variants of HRPS were used in the comparison. The first one is the conventional hybrid-range partitioning strategy of [5] where logical fragments are assigned in a round-robin fashion, whereas in the second variant, we assign the fragments using *LPT* heuristic. We denote them by *HRPS/RR* and *HRPS/OPT* respectively.

We observed from experimental results that the skew in heats of partitions obtained using *HRPS/OPT* algorithm is better than *DP_VAR* algorithm. Also, this skew is close to that obtained by using *LPT* algorithm. *HRPS/RR*'s performance is inferior to other algorithms. Table 6 shows the variance in heats of partitions when the number of disks is 4 and the number of units is 1000. *HRPS/OPT* algorithm performs better than *DP_VAR* algorithm and is close to *LPT* when skew is high. Again, the performance of *HRPS/RR* is inferior to other algorithms. Similar results are obtained when N and P are varied. Table 7 gives the variance in heats of partitions when P=16 and N=10000 and it confirms the observations made above. As expected, the ratio of the time taken by the HRPS algorithm to the *DP_VAR* algorithm is equal to the number of logical fragements divided by the degree of declustering (which is five in our experiments). This is because, for HRPS algorithms, the first step is to run *DP_VAR* algorithm on more number of ranges than the number of disks (5 times more in our case).

Algorithm → Distribution ↓	Greedy	Cooling	DP_SK	DP_VAR
Normal	0.19	0.13	727.74	3203.15
Uniform	0.12	0.12	726.14	3201.60
Zipfian (skew=1)	0.19	0.12	724.00	3198.76
Zipfian (skew=5)	0.17	0.13	630.83	2812.12

Table 5: Comparison of the time (in seconds) taken by the algorithms for 16 disks 10000 units

Algorithm → Distribution ↓	HRPS/OPT	HRPS/RR	DP_VAR	LPT
Uniform	0.00	0.00	0.00	0.00
Normal	0.00	5.00	2.00	0.00
Zipfian (skew=1)	1404.00	1831.00	2466.00	0.00
Zipfian (skew=5)	2269428.00	2668260.00	2270833.00	2268960.00

Table 6: Comparison of the variance in heats of partitions for 4 disks 1000 units

6.2.1 *Impact of varying the number of logical fragments*

We have also conducted experiments to analyze the impact of number of logical fragments on the performance of HRPS algorithms. In earlier experiments, the ratio of number of logical fragments to number of disks is 5. We repeated all experiments with a ratio of 10. Tables 8 - 11, give the comparison among HRPS algorithms as the number of logical fragments is increased. We have not included tables giving the time taken by these algorithms: the *dynamic programming* algorithm DP_VAR used to determine logical fragments takes time linear to the number of ranges (complexity $O(RN^2)$). Hence, the time taken when the number of logical fragments is 10 times the number of disks is approximately twice (slightly more due to allocation phase of LPT or RR heuristic) the earlier case where the ratio was 5.

From the Table 8, it is clear that increasing the ratio of logical fragments to disks to 10 has shown positive results for HRPS/OPT. For clarity, we show the ratio in "brackets" beside the algorithm name i.e., HRPS/OPT(10) stands for HRPS/OPT algorithm when the ratio is 10. The skew in the heats of partitions is same for both HRPS/OPT(5) and HRPS/OPT(10) for *normal*, *uniform*, and *generalized zipfian* distributions and for highly skewed distributions, HRPS/OPT(10) performs better than HRPS/OPT(5). This is because of the more uniform distribution of tuples as the number of logical fragments into which the relation is declustered is increased. This is also confirmed by smaller variances obtained for HRPS/OPT(10), as shown by the Table 9. But when the configuration contains 64 disks, the performance of HRPS/OPT is same in both cases. This may be due to the fact that 645 = 320 logical fragments are sufficient enough to capture the skew. Tables 10 and 11 show this result. The performance of HRPS/RR is rather erratic. In some cases HRPS/RR(10) performs better while in others, HRPS/RR(5) is better.

7. RELATED WORK

The work most closely related to ours was in the context of data placement in the Bubba Project [2] and file allocation in disk arrays [8]. They consider the more general problem of data allocation to multiple disks for a set of relations or a set of files. The general problem includes (i) finding the degree of declustering of each relation, (ii) the set of disks on which each relation is to be declustered and (iii) finally, the actual mechanism for declustering a relation once the degree of declustering and the identitiy of the disks is fixed. It is the last of the above problems that we consider in detail in this paper while the work in [2] and [8] concentrate on the first two problems. To the best of our knowledge, the last of the above problems has not been studied in detail although [2] mentions that each relation is declustered in a heat balanced manner in Bubba but does not however expand on how this is done. We also assume the first two problems have been solved and assume that for each relation we know the degree of declustering as well as the identity of the disks on which the relation is to be declustered. Therefore, our work is complementary to the work in [2] and [8].

In addition, we also consider the problem of moving from one partitioning to another with minimal data movement. Once again, to the best of our knowledge this problem has not been addressed in literature.

8. CONCLUSIONS

We first considered the problem of automatically determining partition boundaries of a relation in a system that uses partitioning. We devised algorithms that determine load balanced partitions using access frequencies on the tuples. We considered two performance metrics for the algorithms proposed - variance in the heats of partitions and skew in their heats. We developed three algorithms for automatically detecting partition boundaries while using range partitioning. The first one is based on *dynamic programming strategy* that finds optimal partitions under both definitions of optimality. However, this algorithm is computationally expensive in terms of both space and time. This motivated us to develop cheaper and faster algorithms. We proposed a *greedy* strategy that determines optimal partitions when the optimization metric is skew. We also proposed a heuristic solution called *Cooling* algorithm. The Greedy and the Cooling strategy are very fast but yet produce partitions which are very close to the optimal.

We then considered the HRPS strategy. In the traditional HRPS strategy, the relation is first partitioned into a num-

Algorithm → Distribution ↓	HRPS/OPT	HRPS/RR	DP_VAR	LPT
Uniform	0.00	0.00	0.00	0.00
Normal	0.00	3.00	2.00	0.00
Zipfian (skew=1)	45578.00	569465.00	53128.00	0.00
Zipfian (skew=5)	83810808.00	91676608.00	83814480.00	83807640.00

Table 7: Comparison of the variance in heats of partitions for 16 disks 10000 units

ber of logical ranges and then they are assigned to disks in round-robin fashion. We call this strategy HRPS/RR. We suggest a strategy called HRPS/OPT in which the assignment of fragments to disks is done in an optimal fashion. Since this optimal assignment is NP-Complete under both definitions of optimality viz. minimizing variance and minimizing the skew, we presented heuristic solutions. HRPs/OPT is one such heuristic that uses the well known LPT heuristic. Our experimental results showed that HRPS/OPT produces partitions with smaller variance than those produced by plain range partitioning as well as HRPS/RR.

Finally, we proposed an optimal solution to the problem of migrating from an already partitioned database to a new patitioned database with minimum data movement.

9. REFERENCES

[1] N. Christofides. *Graph Theory: An Algorithmic Approach.* Academic Press, New York, 1975.

[2] G. Copeland, W. Alexander, E. Boughter, and T. Keller. Data Placement in Bubba. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, pages 99–108, June 1988.

[3] L. Ford and D. Fulkerson. *Flows in Networks.* Princeton University Press, New Jersey, 1962.

[4] M. R. Garey and D. S. Johnson. *Computers And Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[5] S. Ghandeharizadeh and D. J. DeWitt. Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines. In *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 481–492, Brisbane, Australia, Aug. 1990.

[6] S. Ghandeharizadeh and D. J. DeWitt. A Performance Analysis of Alternative Multi-Attribute Declustering Strategies. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, pages 29–38, California, U.S.A., June 1992.

[7] R. J. Lipton, J. F. Naughton, D. A. Schneider, and S. Seshadri. Efficient sampling strategies for relational database operations. *Theoretical Computer Science*, 116, 1993.

[8] G. Weikum, P. Zabback, and P. Scheuermann. Dynamic File Allocation in Disk Arrays. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, pages 406–415, Colorado, U.S.A., May 1991.

Algorithm → / Distribution ↓	HRPS/OPT (5)	HRPS/OPT (10)	HRPS/RR (5)	HRPS/RR (10)
Uniform	1250.00	1250.00	1250.00	1250.00
Normal	1250.00	1250.00	1252.00	1251.00
Generalized Zipf	1251.00	1250.00	1255.00	1253.00
Zipfian (skew=0.9)	1312.00	1293.00	1318.00	1399.00
Zipfian (skew=0.95)	1359.00	1255.00	1515.00	1518.00
Zipfian (skew=3)	3330.00	3330.00	3957.00	3602.00
Zipfian (skew=5)	3859.00	3859.00	4079.00	4100.00

Table 8: Comparison of the skew in heats of partitions for 4 disks 1000 units

Algorithm → / Distribution ↓	HRPS/OPT (5)	HRPS/OPT (10)	HRPS/RR (5)	HRPS/RR (10)
Uniform	0.00	0.00	0.00	0.00
Normal	0.00	0.00	5.00	3.00
Generalized Zipf	0.00	0.00	20.00	9.00
Zipfian (skew=0.9)	1404.00	1498.00	1831.00	9654.00
Zipfian (skew=0.95)	4049.00	18.00	24793.00	38104.00
Zipfian (skew=3)	1442209.00	1442222.00	2443646.00	1880991.00
Zipfian (skew=5)	2269428.00	2268988.00	2668260.00	2709036.00

Table 9: Comparison of the variance in heats of partitions for 4 disks 1000 units

Algorithm → / Distribution ↓	HRPS/OPT (5)	HRPS/OPT (10)	HRPS/RR (5)	HRPS/RR (10)
Uniform	80.00	80.00	80.00	80.00
Normal	80.00	80.00	88.00	81.00
Generalized Zipf	88.00	80.00	135.00	133.00
Zipfian (skew=0.9)	381.00	381.00	431.00	424.00
Zipfian (skew=0.95)	536.00	536.00	591.00	573.00
Zipfian (skew=3)	3330.00	3330.00	3343.00	3344.00
Zipfian (skew=5)	3859.00	3859.00	3872.00	3873.00

Table 10: Comparison of the skew in heats of partitions for 64 disks 1000 units

Algorithm → / Distribution ↓	HRPS/OPT (5)	HRPS/OPT (10)	HRPS/RR (5)	HRPS/RR (10)
Uniform	5.00	5.00	5.00	5.00
Normal	3.00	3.00	9.00	7.00
Generalized Zipf	13.00	2.00	129.00	207.00
Zipfian (skew=0.9)	1848.00	1847.00	3443.00	3649.00
Zipfian (skew=0.95)	4276.00	4273.00	6252.00	6717.00
Zipfian (skew=3)	170483.00	170481.00	172075.00	172199.00
Zipfian (skew=5)	227077.00	227076.00	228689.00	228814.00

Table 11: Comparison of the variance in heats of partitions for 64 disks 1000 units