The Effect of Bucket Size Tuning in the Dynamic Hybrid GRACE Hash Join Method

Masaru KITSUREGAWA† Masaya NAKAYAMA‡ Mikio TAKAGI†

† Institute of Industrial Science, The University of Tokyo
7-22-1 Roppongi, Minato-ku, Tokyo 106, Japan

‡ Toyohashi University of Technology
1-1 Tempaku-cho, Toyohashi 440, Japan

Abstract

In this paper, we show detailed analysis and performance evaluation of the Dynamic Hybrid GRACE Hash Join Method (DHGH Method) when the tuple distribution in buckets is unbalanced.

The conventional Hash Join Methods specify the tuple distribution in buckets statically. However it may differ from estimation since join operations are applied with selection operations. When the tuple distribution in buckets is unbalanced, the processing cost of join operation becomes more costly than the ideal case when you use Hybrid Hash Join Method (HH Method). On the other hand, when you use the DHGH Method, the destaging buckets are selected dynamically, gives the same performance as the ideal case even if the tuple distribution in buckets is unbalanced such as Zipf-like distributions.

We analyze the total I/O cost of a join operation at various number of buckets. The result shows that we have to determine the number of buckets based on the tuple distribution in buckets rather than the size of the source relation. It is shown that we had better partition the source relation using a large number of small buckets instead of the smaller number of buckets almost filling the whole main memory adopted in the HH Method.

1 Introduction

The *join* is one of the most expensive relational algebra operations and many join methods have been proposed [Sto76,Kit83,Bra84,DeW84,Yam85]. Among them, '*split based hash-partitioned join methods*', such as GRACE Hash Join Method, present good performance for large scale database [Kit83,DeW84,Sha86]. In those join methods, the source relations are partitioned into number of "*buckets*" by a "*split function*" when the size of the smaller relation exceeds that of the available staging memory. The number of buckets is determined by the size of the source relation. In the Hybrid Hash Join Method (HH Method), which combines the good features of GRACE Hash Join Method and Simple Hash Join Method, the number of buckets is determined by the relative sizes of available staging memory and relation; it is computed so as to make the aggregated size of buckets are almost same to the size of staging memory, excepting for the first processing bucket called "R_1 *bucket*". So, if the split function does not suit for the tuple distributions in the source relations, some buckets exceed the memory size. Such buckets, named "*overflown buckets*" in this paper, diminish the performance of the HH Method and you cannot expect to achieve optimal performance [Nak88]. In order to minimize the overflown buckets, we divide the source relations into a large number of buckets. We name this approach as "*Dynamic Hybrid GRACE Hash Join Method*" (DHGH Method), because we dynamically select the first processed buckets and establish a scheduled processing sequence of buckets. In this join method, the unbalanced tuple distribution in buckets does not diminish the performance as shown in [Nak88].

The purpose of this paper is twofold. First, we evaluate the processing cost of the DHGH Method at various number of buckets to examine the minimal occurrence of overflown buckets and to identify the guideline for determining the number of buckets when the tuple distribution in buckets is unbalanced.

If we divide the source relations into a large number of buckets, the size of each bucket may become too small to fill up one page. We call such short-filled pages as "*fragment pages*". The I/O operations for them diminish the performance. Therefore, in our join algorithm, many of these short-filled buckets are put together to a larger bucket in accordance with the available main

Proceedings of the Fifteenth International Conference on Very Large Data Bases

Amsterdam, 1989

memory. This technique is called "*bucket size tuning*" as in [Kit83]. Second purpose of this paper is to evaluate this strategy. We show the detailed evaluation of this strategy and exhibit its effectiveness.

In section 2, we define the notations used in this paper and explain the environment of our performance evaluation.

In Section 3, we analyze the DHGH Method and show certain requirements for its algorithm. Roughly speaking, there are two conditions: the first condition is for eliminating overflown buckets, and the second one is for keeping the first processing bucket in the available staging memory.

Section 4 gives the performance results of three kinds of tuple distributions. Performance was evaluated by using various number of buckets. When we treat large sized relations, all results indicate that the elimination of overflown buckets is the key condition to attain good performance. They also show that large enough number of buckets is effective to minimize overflown bucket.

Section 5 shows the performance results for medium sized relations. In this section, we show the effect of bucket size tuning strategy. Bucket size tuning for the first processing buckets gives advantages when the number of buckets is small. On the other hand, bucket size tuning for other buckets is effective when the number of buckets is large because this tuning is also effective for I/O reduction of fragment pages.

Section 6 concludes this paper. The results of all evaluations show that we should divide relations into larger number of buckets than conventional suggestions. The conventional join method determines the number of buckets based on the relative size of the source relation and the available main memory, but we determine it based on the tuple distribution in buckets.

2 Overview of the Dynamic Hybrid GRACE Hash Join Method

Previous researches in hash-partitioned join methods did not concern themselves with the unbalanced tuple distribution in buckets and assumed optimal tuple distributions. Our proposed join method can dynamically handle most of distributions [1]. Our previous paper [Nak88] shows an evaluation for a triangular tuple distribution in buckets (Figure 1) as an example with good results. In this paper, we show a more detailed evaluation of the DHGH Method including the tuple distribution used in [Nak88]. In addition, Zipf-like distribution is selected for another example of unbalanced tuple distribution as is used in AS^{3}AP benchmark set [Tur87,Tur88].

[1] If there is a bucket which exceeds the available main memory size with same valued join attributes, we have to use nested loop manner for achieving join operation. This is an exceptional example for our algorithm.

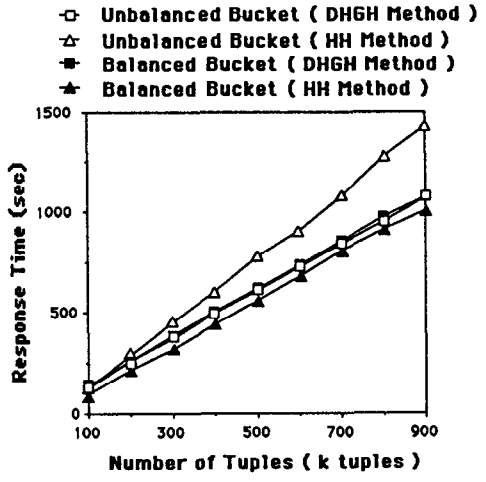

Figure 1: Comparison of processing time between our join method and conventional join method

2.1 The notations and definitions used in this paper

The two source relations of join operation are named R and S (R $\leq$ S). The result relation is named RES. When the size of relation R is smaller than that of the available main memory, Simple Hash Join Method [DeW84,DeW85] may be applied. While staging the whole relation R into the available main memory, we apply a **hash function** to each tuple and make a hash table. Then each tuple in relation S is staged and applied to the same hash function to achieve the join processing. In short, one scanning of each relation is enough to achieve join operation in this case.

On the other hand, when the size of relation R is larger than that of the available main memory, we have to divide it into sets of subrelations. We call these subrelations '**buckets**' and describe them as R_i ($1 \leq i \leq H_s$) in this paper. Here, H_s means the number of splitting buckets. The function to divide the source relations into buckets is called '**split function**'. When the bucket size is smaller than that of the available main memory, join operation for this bucket can be achieved by one scanning as described above. It means that join operation can be achieved by using two phase algorithms. The first phase is to divide the source relations into many buckets, and the second phase is to process the join operation for each bucket. The former phase is called '**split phase**' and the latter '**probe phase**'. Figure 2 shows the outline of join processing.

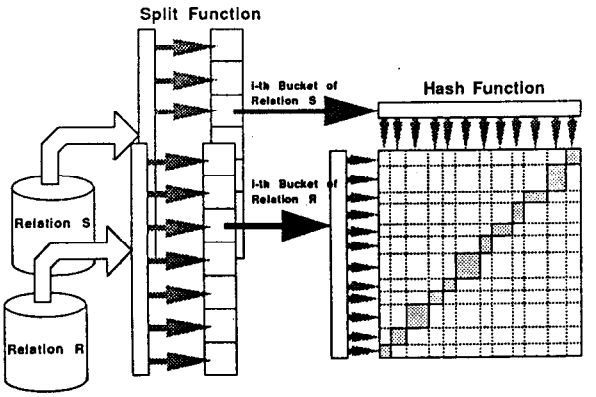

Figure 2: Two kinds of hash functions used by split based join methods

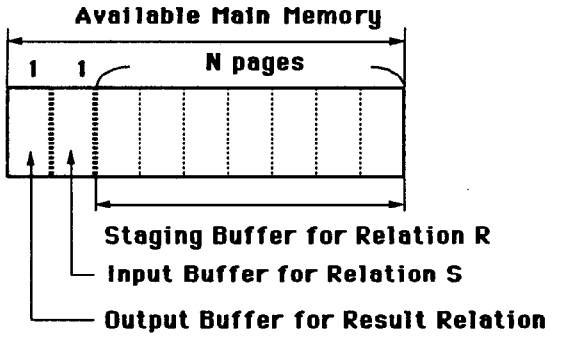

Figure 3: Memory usage for processing a join operation in our experiments

In the split phase, the CPU cost (the tuple splitting cost and the tuple movement cost) is negligible in comparison with I/O cost. So we treat buckets in main memory by page unit and tuples in the same bucket are placed closely together to save the I/O cost. As shown in Figure 3, main memory contains N pages for staging relations and 2 pages for input and output buffers.

When we partition relations into H_s buckets, $N - H_s$ pages are not used in the split phase if each phase is fully isolated (GRACE Hash Join Method). Instead of this method, the "**Hybrid Hash Join Method**" uses these free pages for staging R_1 bucket. This technique saves the I/O cost for R_1 bucket in join processing. In [DeW85,Sha86], it is shown that this method always gives the best performance among all of join methods when the following formula is satisfied.

$$H_s = \left\lceil \frac{|R| - |M|}{|M| - 1} \right\rceil + 1 \quad (1)$$

$$|R_1| = |M| - H_s + 1 \quad (2)$$

$$|R_i| = \frac{|R| - |R_1|}{H_s - 1} \quad (2 \le i \le H_s) \quad (3)$$

In the formula above, $|M|$ means the number of pages of available staging memory and $|R|$ means the number of pages of relation R. The notation $|M|$ is equivalent to N in our environment. In addition to this notation, we use $\{R\}$ to describe the number of tuples of relation R.

When the above formula is not satisfied for any reason, additional processing cost is required. If the size of R_1 bucket is small, the saving of the I/O cost is not significant. In such a case, other buckets get more tuples than the estimation and some buckets may exceed N pages (overflown buckets). These buckets must be recursively split into many **sub-buckets** and only then join operations can be performed. If the i-th bucket exceeds N pages and is partitioned into H'_s sub-buckets, we describe them as R_{ij} $(1 \le j \le H'_s)$. On the other hand, if the size of R_1 bucket exceeds the available staging memory, we have to re-split the R_1 bucket dynamically.

These additional overheads are necessary because of the statical decision of the R_1 bucket. In the DHGH Method, however, we dynamically determine the R_1 bucket and split the source relations into a large number of buckets to prevent overflown buckets. We discuss the method to determine the number of buckets in section 4.

When we divide the source relations into a large number of buckets, each bucket may have fewer tuples than $\{M\}$ (fragment pages). Such fragment pages can be put together into a bucket for further processing: i.e. a set of buckets, in effect, can be processed at a time. This strategy is called '**bucket size tuning**' and there are two kinds of size tuning. The size tuning for the buckets whose join processing are overlapped with split phase is called '**R_1 bucket size tuning**'. The other bucket size tunings is called '**R_i bucket size tunings**'. The advantages of size tuning are discussed in section 5.

2.2 Environment for performance evaluation

In this paper, we evaluate the performance for the Zipf-like distribution as is used in the AS^{3}AP benchmarks [Tur87,Tur88], This distribution is the generalization of the Zipf distribution [Knu73]. A power of the rank of an item is inversely proportional to its frequency:

$$i^z \times f_i = \frac{1}{constant^{(z)}} \quad (1 \le i \le n)$$

In this formula, z is the decay factor and $constant^{(z)}$ is the n-th harmonic number of order z. If $z = 0$, the distribution becomes uniform. When $z = 1$, it shows

the Zipf distribution. And also, G. K. Zipf found the distribution of personal income when $z = 0.5$.

We assume that tuple distribution of each bucket follows this distribution. For instance, i-th bucket of relation R has such number of tuples as shown in the following formula:

$$\{R_i\} = \frac{\{R\}}{i^z \cdot \sum_{j=1}^{H_s} \frac{1}{j^z}} \qquad (1 \le i \le H_s) \tag{4}$$

Though each bucket has an unbalanced distribution, we assume that each join attribute has a unique value in the relation in order to get the same total I/O cost for join operation for all distribution of buckets.

3 Performance analysis of the Dynamic Hybrid GRACE Hash Join Method

The algorithm of new join method was shown in [Nak88]. In this section, we show two conditions to ensure the good performance of the DHGH Method. As illustrated in the last section, the HH Method determines the number of buckets as small as possible. When the tuple distribution in buckets is unbalanced, performance becomes worse because of the overflown buckets. In the DHGH Method, we determine the number of bucket as large as we can for preventing overflown buckets; we divide the relation into number of buckets to satisfy the following formula:

$$\max|R_i| \le |M| \tag{5}$$

As for the size of R_1 bucket, it is determined by the formula (2) in the HH Method. In our method, however, the minimum sized bucket is dynamically selected for the R_1 bucket. So the condition is denoted as the following formula:

$$\min|R_i| \le |M| - H_s + 1 \tag{6}$$

Also, when we apply a join operation to small sized relations, we may collect a number of buckets to the first processing bucket. This schedule is called R_1 bucket size tuning. If you give the bucket identifier k in accordance with the size sequence, the previous condition (6) can by rewritten as follows. In this case, we assume that t buckets are selected as the first staging buckets.

$$\sum_{k=1}^{t}|R_k| \le |M| - H_s + t \tag{7}$$

In this paper, we analyze the I/O cost for join operation changing the number of buckets. If we can stage t buckets in the main memory at the end of split phase, we can denote the I/O cost of join operation, $C(R,S)$, as follows.

$$C(R,S) = |R| + |S| + |RES| \qquad (|R| \le |M|) \tag{8}$$

$$C(R,S) = |R| + |S| + \sum_{k=1}^{t}|RES_k| + \sum_{k=t+1}^{H_s}(|R_k| + |S_k| + C(R_k,S_k)) \qquad (|M| < |R|) \tag{9}$$

When the size of relation R is larger than that of available memory, total I/O cost of join operation is recursively defined. If we can divide the source relation into a number of buckets to satisfy the condition (5), $C(R,S)$ can be denoted as follows.

$$\begin{aligned} &C(R,S) \\ &= |R| + |S| + \sum_{k=1}^{t}|RES_k| \\ &\quad + \sum_{k=t+1}^{H_s}\{|R_k| + |S_k| + (|R_k| + |S_k| + |RES_k|)\} \\ &= |R| + |S| + |RES| + 2\sum_{k=t+1}^{H_s}(|R_k| + |S_k|) \end{aligned} \tag{10}$$

When all those buckets between $u+1$ and H_s are overflown buckets and thus further divided into $H_s^{(k)}$ sub-buckets satisfying the condition (5), total I/O cost becomes as follows.

$$\begin{aligned} &C(R,S) \\ &= |R| + |S| + \sum_{k=1}^{t}|RES_k| + \sum_{k=t+1}^{H_s}(|R_k| + |S_k|) \\ &\quad + \sum_{k=t+1}^{u}(|R_k| + |S_k| + |RES_k|) \\ &\quad + \sum_{k=u+1}^{H_s}\{|R_k| + |S_k| + |RES_k| \\ &\qquad + 2\sum_{l=t^{(k)}+1}^{H_s^{(k)}}(|R_{kl}| + |S_{kl}|)\} \\ &= |R| + |S| + |RES| + 2\sum_{k=t+1}^{H_s}(|R_k| + |S_k|) \end{aligned}$$

$$+2 \sum_{k=u+1}^{H_s} \sum_{l=t^{(k)}+1}^{H_s^{(k)}} (|R_{kl}| + |S_{kl}|) \qquad (11)$$

In this formula, we assume that $t^{(k)}$ sub-buckets can be collected into the first staging sub-bucket in split phase of k-th bucket. Both $t^{(k)}$ and $H_s^{(k)}$ are dependent on the tuple distribution in the k-th bucket and $t^{(k)}$ is also dependent on the size of k-th bucket. Here, we assume that the tuple distribution in sub-buckets is same distribution as in buckets and assume that $H_s^{(k)}$ $(u < k \leq H_s)$ is equal to the H_s value.

In this remaining section, we analyze the I/O cost of join operations for the same two relations. As described before, each tuple has a unique value at join fields in the relation R and the amount of result relation becomes twice the source relation ($|RES| = 2|R|$): Formula (10) and (11) are rewritten as follows.

$$C(R,R) = 4|R| + 4 \sum_{k=t+1}^{H_s} |R_k| \quad (formula\ 10) \qquad (12)$$

$$C(R,R) = 4|R| + 4 \sum_{k=t+1}^{H_s} |R_k| + 4 \sum_{k=u+1}^{H_s} \sum_{l=t^{(k)}+1}^{H_s} |R_{kl}| \quad (formula\ 11) \qquad (13)$$

Now, we can get information about extra I/O cost for overflown buckets. For detailed analysis, we have to fix the tuple distribution in buckets. We choose the triangular distributions for this analysis. The tuple distribution in buckets can be described as follows.

$$\{R_k\} = \frac{2\{R\}}{H_s(H_s+1)} \cdot k \qquad (1 \leq k \leq H_s)$$

Here T denote the minimum number of buckets to satisfy the condition (5). When t buckets can be staged into the staging memory at the same time, C_T can be described as follows.

$$C_T = 4|R| + 4 \sum_{k=t+1}^{T} \frac{2|R|}{T(T+1)} \cdot k$$

When we use $T-1$ buckets in the split phase, the largest bucket (R_{T-1} bucket) becomes overflown bucket and the I/O cost of join operation is described as follows. (Appendix A shows the reason why we get such a formula.)

$$\begin{aligned} C_{T-1} &= 4|R| + 4 \sum_{k=t+1}^{T-1} |R_k| + 4|R_{T-1,T-1}| \\ &= 4|R| + 4 \sum_{k=t+1}^{T-1} \frac{2|R|}{T(T-1)} \cdot k + 4\frac{4|R|}{T^2} \end{aligned}$$

The extra I/O cost for processing the overflown bucket can be calculated as follows.

$$\begin{aligned} C_{extra} &= C_{T-1} - C_T \\ &= 8|R| \cdot \left(\frac{2}{T^2} - \frac{t(t+1)}{T(T-1)(T+1)} \right) \end{aligned}$$

This formula shows that an overflown bucket leads about $2/T^2$ of I/O cost disadvantages at most [2]. When we treat large relations, the occurrence of the overflown bucket is not so serious a problem because T is large. When we treat the small relations, however, we have to carefully determine the number of buckets to avoid the overflown bucket. The following two sections show these phenomena separately.

4 Performance results of three kinds of the bucket distributions

As shown in Figure 1, the HH Method does not suit for unbalanced tuple distribution in buckets. The processing cost for unbalanced buckets is about 1.4 times worse than that for balanced buckets. The reason for this phenomenon is not clearly discussed in [Nak88]. In this section, we evaluate the join performance of three kinds of tuple distribution in buckets as we change the number of buckets and the size of source relations.

We choose the triangular distribution, the Zipf-like distribution and the uniform distribution as the three kinds of tuple distribution in buckets. The former two distributions are used to evaluate the join performance for unbalanced buckets. The uniform distribution is used to get the base join performance.

In each distribution, we calculate the I/O cost during a join operation as we change the size of source relations and the number of buckets. We assume that the available staging memory has 100 pages and each page can contain 32 tuples in it. In this case, H_s can be changed between 2 and 99 (= $|M| - 1$). When the size of relation R is greater than that of the available staging memory, we need at least 2 buckets for achieving join operation. And the Dynamic Destaging Strategy, used in the DHGH Method, determines the upper bound of H_s. When we have no free pages in the staging memory at the split phase, destaging bucket is selected from the set of buckets which has more than one page. So the number of buckets must be less than $|M|$.

[2] For splitting the source relation into buckets, twice I/O operations (read the source relation and write to the temporal relation) are needed for each source relation. To probe join operation, another read and write operations are needed. Totally, 8 I/O operations are needed for basic join operation.

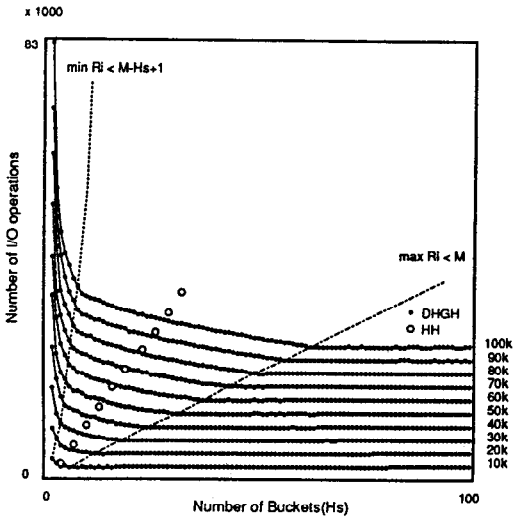

Figure 4: The performance results for triangular distribution

For all performance evaluations in this section, the number of tuples in source relations is changed every 10,000 tuples from 10,000 tuples to 100,000 tuples. As described above, at most 3,200 tuples can be staged in the available staging memory at a time. Most of the relations used for performance evaluation have more than five times larger tuples than the available staging memory contains. They are called "large sized relations" in [Nak88]. The performance results of "medium sized relations" are shown in the next section.

4.1 Performance results of the triangular distributions

First, we show the results of triangular distribution. This distribution is chosen in [Nak88] and we evaluate more detailed phenomena in this paper. Figure 4 shows the number of I/O operation for each number of bucket and for each size of the source relation.

In this figure, two different dotted lines show the boundary conditions of equation (5) and (7). The white points show the results of the HH Method and the black points show the results of the DHGH Method for each size of relation.

If the size of each bucket does not exceed the size of the available staging memory, H_s does not affect the performance that much. These phenomena are shown in the area whose H_s value is greater than the condition (5) line.

When some buckets exceed the available memory size, extra I/O cost is required to perform join operation for such overflown buckets. However, it is not so large on amount. These phenomena are shown in the area whose H_s value is between two condition lines.

When all the buckets exceed the size of available memory, the performance becomes extremely worse. These phenomena are shown in the area whose H_s value is smaller than the condition (7) line. In this situation, the maximum size of sub-bucket of $|R_{H_s}|$ bucket exceeds the size of $|R_1|$ bucket as follows.

$$\begin{aligned}
&\max|(\max|R_k|)_t| - \min|R_k| \\
&= |R_{H_s,H_s}| - |R_1| \\
&= \frac{2}{H_s+1}\cdot\frac{2|R|}{H_s+1} - \frac{2|R|}{H_s(H_s+1)} \\
&= 2|R|\cdot\frac{H_s-1}{H_s(H_s+1)^2} \quad > \quad 0
\end{aligned}$$

It means that sub-buckets whose size exceed the available staging memory need much more re-splitting operation to process the join operation. This is the reason why the performance of join operation becomes so much worse.

4.2 Performance results of the uniform distributions

In this subsection, we show the results of uniform distributions. Similar to the triangular distributions, we evaluate the I/O cost of join operation at various size of the source relation and the number of buckets. Figure 5 shows the results. Two different dotted lines, the white points and the black points have the same meaning as Figure 4.

Unlike the triangular distribution, the condition (5) is more severe than the condition (7). When all buckets satisfy both conditions, the I/O cost can be described by using formula (10) as follows:

$$\begin{aligned}
C(R,R) &= 4|R| + 4\sum_{k=t+1}^{H_s}|R_k| \\
&= 4|R| + 4\frac{|R|}{H_s}\cdot(H_s - t) \\
&= 8|R| - 4\frac{|R|}{H_s}\cdot t \qquad (14)
\end{aligned}$$

When T denotes the minimum number of the buckets which satisfies the condition (7), the I/O cost can be described as follows.

$$C_T = 8|R| - 4\frac{|R|}{T}$$

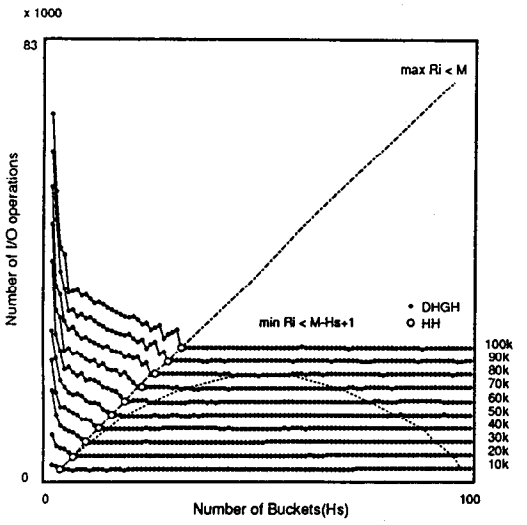

Figure 5: The performance results for uniform distribution

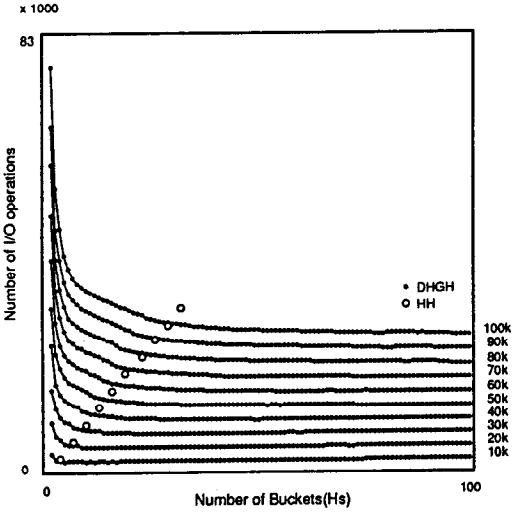

Figure 6: The performance results for Zipf-like distribution ($z = 0.5$)

When the condition (7) is not satisfied, the total I/O cost can be calculated as $t = 0$ in the formula (14).

$$C_{T-1} = 8|R|$$

The extra cost for the R_1 bucket overflown is described as follows. And the disadvantage becomes about $1/2T$.

$$C_{extra} = C_{T-1} - C_T = 8|R|\frac{1}{2T}$$

When the tuple distribution in buckets is balanced, each bucket has the same number of tuples. If the condition (5) is not satisfied, all destaging buckets become overflown buckets. The total I/O cost is described as follows.

$$\begin{aligned} C(R,R) &= 4|R| + 4\sum_{k=1}^{H_s}|R_k| + 4\sum_{k=1}^{H_s}\sum_{l=t+1}^{H_s}|R_{kl}| \\ &= 8|R| + 4\frac{|R|}{H_s}\cdot(H_s - t) \end{aligned}$$

In this formula, t is a constant value for all buckets. When T denotes the minimum number of the buckets to satisfy the condition (5), the I/O cost of join operation can be described as follows.

$$C_T = 8|R|$$

$$C_{T-1} = 8|R| + 4\frac{|R|}{T-1}\cdot(T-1-t)$$

And the extra I/O cost can be calculated as follows.

$$C_{extra} = C_{T-1} - C_T = 4|R|\cdot\frac{T-1-t}{T-1}$$

In this formula, both value of T and t are dependent on $|R|$. When $|R|$ is small, $(T-1-t)/(T-1)$ also becomes small and the extra I/O cost does not affect the performance. On the other hand, when $|R|$ is large, $(T-1-t)/(T-1)$ becomes large and the extra I/O cost will affect the performance.

4.3 Performance results of the Zipf-like distributions

To evaluate the performance for actual databases, we use the Zipf-like distributions in this subsection. The Zipf-like distribution has a parameter, the decay factor z. We evaluate the performance when z is 0.5 and 1.0. Figure 6 and Figure 7 show the results respectively. The white points show the results of the HH Method and the black points show the results of the DHGH Method.

We get similar results to those of the triangular distributions. However the results of Zipf-like distributions are not as good as that of the triangular distributions. Because Zipf-like distribution is more severe than the triangular distributions for the condition (5). In the triangular distributions, the largest bucket contains $2\{R\}/H_s$ tuples at most. However, in the Zipf-like distributions, it contains $\{R\}/(\sum_{j=1}^{H_s} 1/j^z)$ tuples. If we treat the Zipf dis-

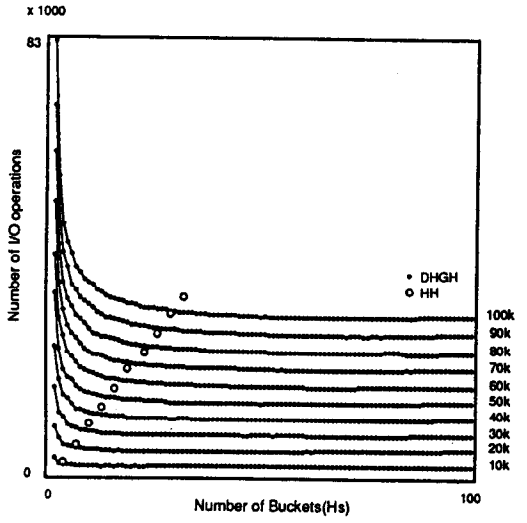

Figure 7: The performance results for Zipf-like distribution ($z = 1.0$)

tribution (z=1.0), sum of the inverse numbers becomes $\ln H_s$ and the largest bucket contains $\{R\}/\ln H_s$ tuples. It means that the source relation which contains more than 15k tuples produces overflown buckets even though we use dynamic destaging strategies. In this case, additional I/O cost for processing join operation is needed and the performance is degraded. This phenomenon is observed in the income distribution (z=0.5). If the relation contains more than 60k tuples, overflown bucket is occurred though the amount is little in such cases.

The following figure shows the total I/O cost ratio in each join method to the basic I/O cost for join operation ($8|R|$).

This figure clearly shows that we need additional I/O to apply join operation for Zipf-like distributions. It also shows that medium sized relation for all distribution does not have overflown buckets.

5 Discussion of bucket size tuning

We show the results for medium sized relations which has triangular distribution and discuss the effect of bucket size tuning in this section.

As described in [DeW84,DeW85,Sha86,Ger86], the advantage of R_1 bucket overlap processing is remarkable when the source relation is a small size as the available staging memory. In our environment, 3,200 tuples can be staged at a time. And we evaluate the performance of 5,000 tuples and 10,000 tuples relations. As illustrated

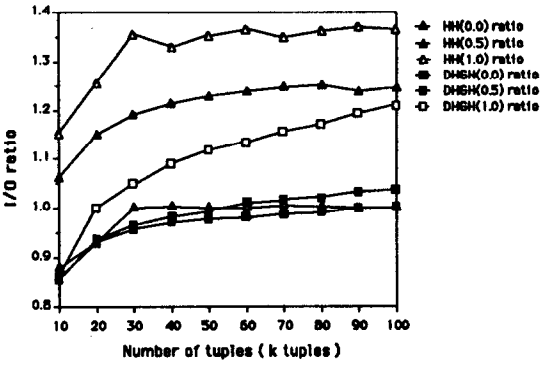

Figure 8: The total I/O ratio to the basic I/O operation

before, we have two kinds of schedules about bucket size tuning, R_1 bucket size tuning and R_i bucket size tuning. To evaluate the detailed performance of bucket size tuning, we use following four kinds of algorithms.

- The algorithm 1 is applied neither to R_1 bucket size tuning and to R_i bucket size tuning.
- The algorithm 2 is applied only to R_1 bucket size tuning.
- The algorithm 3 is applied only to R_i bucket size tuning.
- The algorithm 4 is applied both to R_1 bucket size tuning and to R_i bucket size tuning.

Algorithm 1 includes the result of the HH Method. It determines the join processing sequence of buckets statically. The algorithm 2 only schedules a set of buckets in the R_1 bucket using the Dynamic Destaging Strategy. And the algorithm 3 schedules the join processing sequence of buckets by their sizes. A number of buckets are collected to fill up the available memory. This schedule can reduce the I/O operations for fragment pages. Lastly, the algorithm 4 denotes the DHGH Method itself. Figure 9 and Figure 10 show the result for 5,000 tuples relation and 10,000 tuples relation respectively.

These figures show that R_1 bucket size tuning is effective when the number of buckets is small. That is to say, this strategy is effective when the available staging areas for first processing bucket has a large number of pages. And the advantages of R_1 bucket size tuning decrease as the number of buckets increase. However, this overhead is much less than that of overflown buckets.

When the number of buckets is large enough to avoid the overflown buckets in this situation, each bucket has very small number of tuples in it. In this environment,

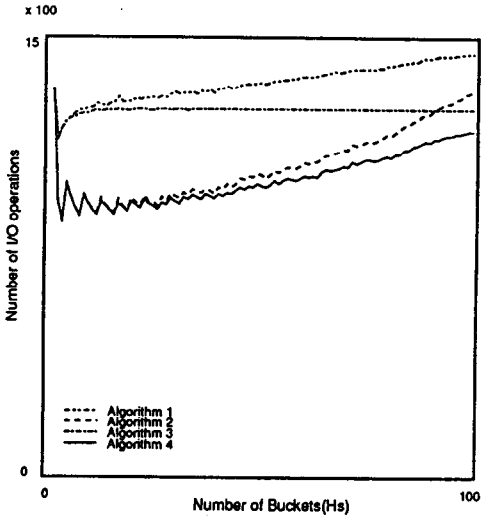

Figure 9: The I/O performance for 5000 tuples relation

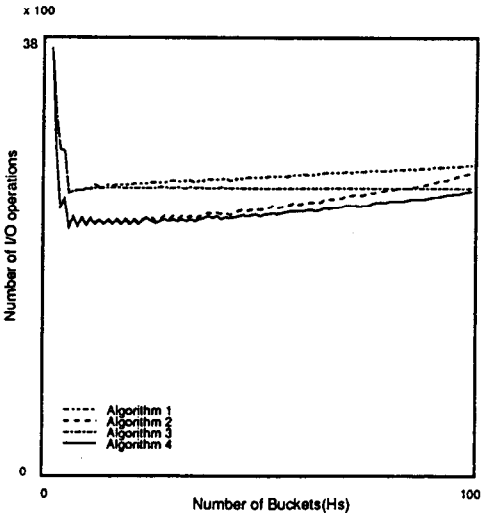

Figure 10: The I/O performance for 10000 tuples relation

the performance becomes worse because of the I/O operations for fragment pages. The result of algorithm 1 and algorithm 3 show these phenomena. So the R_i bucket size tuning is effective to reduce such overheads.

Difference between the result of 5,000 tuples relation and that of 10,000 tuples relation shows the fact that the effect of bucket size tuning is large when the source relation size is small.

6 Conclusions

Conventional split based hash-partitioned join methods determine the processing sequence of each bucket statically. They assume that the size of each bucket is almost same size. However, such cases are actually rare because join operations generally follow some selection/restriction operations and there is no guarantee that the split function is suitable for any kinds of tuple distribution in the source relation. Unbalanced tuple distribution in the buckets diminishs the performance in comparison with the estimation in the conventional join method. We extends the GRACE hash algorithms to obtain high performance than Hybrid Hash algorithms. We name it the Dynamic Hybrid GRACE Hash algorithms.

In this paper, we evaluate the join performance of three kinds of tuple distribution in buckets at various number of buckets and the size of source relations to investigate the phenomenon. The three distributions are; the uniform, the triangular and the Zipf-like distribution. Performance results show that we have to determine the number of buckets to minimize the overflown buckets. There are two conditions which describe the boundary conditions of overflown bucket:

$$\max |R_i| \leq |M|$$

$$\min |R_i| \leq |M| - H_s + 1$$

And the minimum number of buckets which satisfys such conditions depends on the distribution not the size of source relation. In our join method, if we cannot get information about tuple distribution in buckets before processing join operation, the number of buckets is chosen as the maximum number to minimize the overflown buckets.

Also, in this paper, we check the effect of bucket size tuning strategy using the medium sized relation. The R_1 bucket size tuning is efficient when the number of buckets is small and the amount of staging area for R_1 bucket can be taken largely. On the other hand, R_i bucket size tuning works together some fragment pages to reduce the I/O cost of them. So this tuning is efficient when the size of source relation is small and the number of buckets is large.

Appendix A How to lead the I/O cost formula with overflown bucket

When we partition the source relation R into H_s buckets and each bucket follows the triangular distributions, the tuple distribution in buckets becomes as follows.

$$\{R_k\} = \frac{2\{R\}}{H_s(H_s+1)} \cdot k \qquad (1 \le k \le H_s)$$

Here, we denote the minimum number of buckets which satisfys the condition (5) as T.

$$\begin{aligned} \max|R_k| &= \frac{2|R|}{T(T+1)} \cdot T \quad < \quad |M| \\ |M|T &> 2|R| - |M| \end{aligned} \tag{15}$$

And if we use $T-1$ buckets to partition the source relation R, the largest bucket (R_{T-1} bucket) becomes overflown bucket in this case.

$$\begin{aligned} |R_{T-1}| &= \frac{2|R|}{T(T-1)} \cdot (T-1) \quad > \quad |M| \\ 2|R| &> |M|T \end{aligned} \tag{16}$$

Using these conditions, we can lead the fact that the R_{T-2} bucket does not exceed the size of available staging memory.

$$\begin{aligned} & |M| - |R_{T-2}| \\ &= |M| - \frac{2|R|}{T(T-1)} \cdot (T-2) \\ &= \frac{1}{T(T-1)} [|M|T(T-1) - 2|R|(T-2)] \\ &> \frac{1}{T(T-1)} [(2|R| - |M|)(T-1) - 2|R|(T-2)] \\ & \qquad\qquad (\textit{Condition (15)}) \\ &= \frac{1}{T(T-1)} [2|R| - |M|(T-1)] \\ &> \frac{1}{T(T-1)} [|M|T - |M|(T-1)] \\ & \qquad\qquad (\textit{Condition (16)}) \\ &= \frac{|M|}{T(T-1)} \quad > \quad 0 \end{aligned} \tag{17}$$

For all situations when T is larger than 2, only (T-1)-th bucket exceeds the size of available staging memory. It means that both u and $t^{(T-1)}$ in the formula (13) become $T-2$. And the I/O cost with the overflown bucket can be described as follows.

$$C(R,R) = 4|R| + 4\sum_{k=t+1}^{H_s} |R_k| + 4\sum_{k=u+1}^{H_s}\sum_{l=t^{(k)}+1}^{H_s} |R_{kl}|$$

$$= 4|R| + 4\sum_{k=t+1}^{T-1} |R_k| + 4|R_{T-1,T-1}|$$

References

[Bra84] K. Bratbergsengen. "Hashing Methods and Techniquues for Main Memory Database Systems". In *Proc. of the 10th Int. Conf. on VLDB*, pp. 323–333, 1984.

[DeW84] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. R. Stonebraker, and D. Wood. "Implementation Techniques for Main Memory Database Systems". In *ACM SIGMOD '84*, pp. 1–8, 1984.

[DeW85] D. J. DeWitt and R. Gerber. "Multiprocessor Hash-Based Join Algorithms". In *Proc. of the 11th Int. Conf. on VLDB*, pp. 151–164, 1985.

[Ger86] R. H. Gerber. "Dataflow Query Processing Using Multiprocessor Hash-partitioned Algorithms". Technical Report 672, University of Wisconsin, 1986.

[Kit83] M. Kitsuregawa, H. Tanaka, and T. Moto-oka. "Application of Hash to Data Base Machine and Its Architecture". *New Generation Computing*, Vol. 1, No. 1, pp. 66–74, 1983.

[Knu73] D. E. Knuth. *"The Art of Computer Programming"*, volume 3. Addison-Wesley Pub. Co., 1973.

[Nak88] M. Nakayama, M. Kitsuregawa, and M. Takagi. "Hash-Partitioned Join Method Using Dynamic Destaging Strategy". In *Proc. of the 14th Int. Conf. on VLDB*, pp. 468–478, 1988.

[Sha86] L. D. Shapiro. "Join Processing in Database Systems with Large Main Memories". *ACM Transactions on Database Systems*, Vol. 11, No. 3, pp. 239–264, 1986.

[Sto76] M. Stonebraker, E. Wong, P. Kreps, and G. Held. "The Design and Implementation of INGRES". *ACM Transactions on Database Systems*, Vol. 1, No. 3, pp. 189–222, 1976.

[Tur87] C. Turbyfill. *"Comparative Benchmarking of Relational Database Systems"*. PhD thesis, Cornell University, September 1987.

[Tur88] C. Turbyfill, C. Orji, and D. Bitton. "AS^{3}AP – An Ansi Sequel Standard Scalable and Portable, Benchmark for Relational Database Systems". University of Illinois Technical Report, December 1988.

[Yam85] Y. Yamane. "A Hash Join Technique for Relational Database Systems". In *Proc. on Foundations of Data Organization*, pp. 388–398, 1985.