



ELSEVIER

Information Processing Letters 55 (1995) 283–289

Information  
Processing  
Letters

# Uniform partitioning of relations using histogram equalization framework: An efficient parallel hash-based join

Ung Kyu Park <sup>a</sup>, Hwang Kyu Choi <sup>b</sup>, Tag Gon Kim <sup>a,\*</sup>

<sup>a</sup> Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, 373-1 Kusong-Dong, Yusong-gu, Daejeon, 305-701, South Korea

<sup>b</sup> Department of Computer Engineering, Kangwon National University, Chunchon, Kangwon-Do, 200-701, South Korea

Communicated by K. Ikeda; received 20 May 1994; revised 16 November 1994

---

**Keywords:** Databases; Parallel join algorithm; Data skew; Histogram equalization

---

## 1. Introduction

In relational database systems, join operations are the most complex and time consuming ones which limit performance of such systems. Many parallel join algorithms have been proposed for parallel relational database systems [4,6,8]. Among them, the parallel hash-based join algorithm (PHJA) has been found to be superior to other join algorithms for the uniform distribution of data [4,8].

In real databases, it is often found that certain values for a given attribute occur more frequently than other values. This phenomenon is referred to as *data skew*. It is known that data distribution for many textual databases follows a variant of Zipf's law, representing skewed data distribution [5]. With such data distribution, the PHJA shows two major problems in performance: *load imbalance*

and *bucket overflow* [2,10]. This is because data skew can give rise to non-uniform distribution after hashing. Thus, the effectiveness of the PHJA depends on the degree of uniformity in data distribution. As pointed out in [4,9], most algorithms proposed for the PHJA limit exploiting parallelism as the skewness of data distribution becomes large.

Several algorithms have been proposed to overcome such limitations of the PHJA in data skew [2,3,10]. Algorithms proposed in [2] and [3] are based on bucket size tuning. The algorithms are proved to be effective with slightly skewed data. However, they can not remedy the problems of load imbalance and bucket overflow in highly skewed data. In [10], the proposed algorithm added an extra scheduling phase to the usual partitioning and joining phase to solve the two problems above. However, performance study in [9] indicated that, unless the data is highly skewed, the algorithm proposed in [10] becomes markedly worse than that of PHJA. This is because heuristics in the scheduling phase require many steps

---

\* Corresponding author.

for efficient join operations. Moreover, assumptions and approximations employed in the heuristic algorithm are not valid in slightly skewed or uniform data distributions.

This paper proposes an efficient algorithm, called *skewed resolution join algorithm* (SRJA), for parallel join operations with skewed data. We propose a methodology for partitioning relations evenly across all processors in a parallel database system. Using the histogram equalization technique, the framework transforms the histogram of skewed data to uniform distribution that corresponds to the relative computing power of node processors in the system. We performed simulation experiments under the Zipf-like distribution of hashed values for join attributes. Simulation study indicates that the proposed algorithm exhibits better performance than the conventional PHJA in the presence of data skew, with negligible overhead in the absence of data skew.

## 2. Data distribution framework

The following assumptions are made in the remainder of this paper. The parallel database system has  $P$  autonomous processors numbered by  $1, 2, \dots, P$ , each having its own memory and disk, which are linked through an interconnection network. There exist two joining relations labeled as  $R$  and  $S$  in the database, with  $R$  being the smaller one. Initially, both relations are horizontally partitioned into disjoint subsets of the tuples and evenly distributed across all the processors.

### 2.1. Histogram equalization methodology

Suppose that  $\bar{x}$  is a discrete random variable and  $g(x)$  is a monotonic transformation function of the discrete real variable  $x$ . Then the histogram equalization process can be considered as a transformation  $\bar{y} = g(\bar{x})$ . In the transformation, the input random variable  $\bar{x}$ , ranged over  $x_1, x_2, \dots, x_J$  ( $x_1 \leq x_2 \leq \dots \leq x_J$ ), is mapped into an output random variable  $\bar{y}$ , ranged over  $y_1, y_2, \dots, y_K$  ( $y_1 \leq y_2 \leq \dots \leq y_K$ ), such that the output probability density follows a uniform density.

Since a histogram of discrete random variables can be approximately by continuous random variables, we first obtain the transfer function in the continuous case. Because the transformation is monotonic, the fundamental theorem of random variable [7] follows that  $f_{\bar{y}}(y) = f_{\bar{x}}(x)/g'(x)$ , where  $f_{\bar{x}}(x)$  and  $f_{\bar{y}}(y)$  are the probability densities of  $\bar{x}$  and  $\bar{y}$ , respectively and  $g'(x)$  is the derivative of  $g(x)$ . Hence,  $\int_{y_1}^y f_{\bar{y}}(y) dy = \int_{x_1}^x f_{\bar{x}}(x) dx$ . The integral on the right is the cumulative distribution function  $F_{\bar{x}}(x) = P(\bar{x} \leq x)$  of the input variable  $\bar{x}$ . Thus  $\int_{y_1}^y f_{\bar{y}}(y) dy = F_{\bar{x}}(x)$ . In the special case for which the output density is forced to be a uniform density,  $f_{\bar{y}}(y) = 1/(y_K - y_1)$  for  $y_1 \leq y \leq y_K$ , the histogram equalization transfer function becomes

$$y = g(x) = (y_K - y_1)F_{\bar{x}}(x) + y_1. \quad (1)$$

Let us now return to the discrete case. Suppose that  $H_{\bar{x}}(x)$  for  $x = x_1, x_2, \dots, x_J$  represents the fractional number of occurrence frequencies

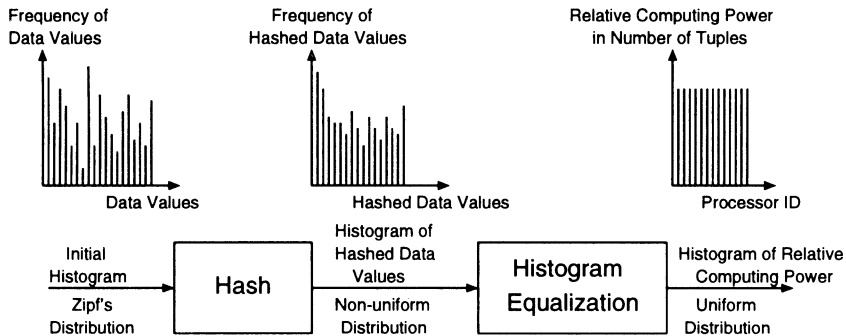


Fig. 1. Data distribution framework for parallel joins.

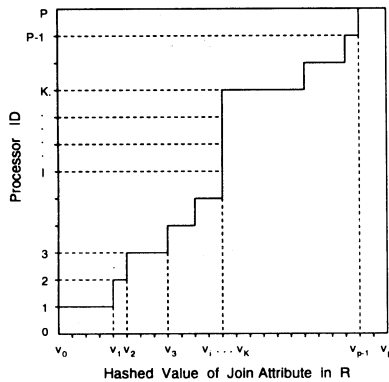


Fig. 2. Transfer function for histogram equalization.

of input values. Then the cumulative probability distribution of the input variable,  $F_{\bar{x}}(x)$ , is approximated by its normalized cumulative histogram as follows:  $F_{\bar{x}}(x) \approx \sum_{m=x_1}^x H_{\bar{x}}(m)$ . Hence Eq. (1) can be modified by

$$y = g(x) = (y_K - y_1) \sum_{m=x_1}^x H_{\bar{x}}(m) + y_1. \quad (2)$$

Our data distribution framework for parallel joins with the aim is shown in Fig. 1. Initially, we have a histogram of data values of the join attribute for a relation. Then, we hash the data values. Finally, the histogram of the hashed values is transformed into a uniformly distributed

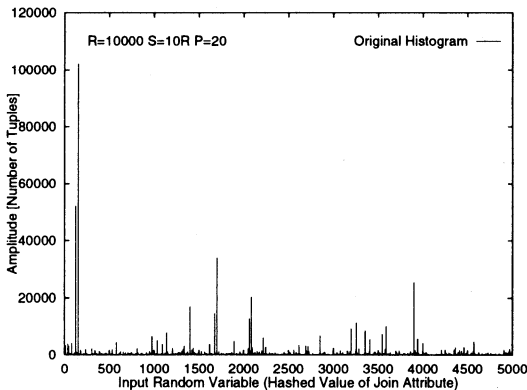
histogram using Eq. (2). Thus, given an arbitrary data histogram, we can obtain even distribution of data among processors. In our histogram equalization, an input random variable  $\bar{x}$  indicates the  $h$  distinct hashed values of the join attribute and an output random variable  $\bar{y}$  corresponds to the processor  $id$ , numbered by  $1, 2, \dots, P$ . Then the histogram equalization transfer function is obtained from Eq. (2) as

$$y = g(x) = \left\lceil P \sum_{m=0}^x H_{\bar{x}}(m) \right\rceil, \quad (3)$$

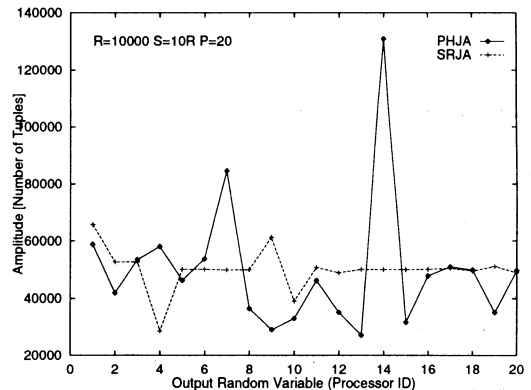
where  $\lceil \cdot \rceil$  is the ceiling function.

### 2.2. Finding upper boundary values for partitioning tuples

Since the transfer function in Eq. (3) is monotonically increasing function, an upper boundary value,  $x$ , for each partition is obtained by inverse transformation of Eq. (3). That is, the upper boundary value of the  $i$ th processor can be obtained by  $g^{-1}(i)$ . Let the  $P$  boundary values be  $v_1, v_2, \dots, v_P$  and the smallest hashed value of the join attribute of  $R$  be  $v_0$ . Then, a hashed value of the join attribute,  $a$ , falls into the range  $v_{i-1} < a \leq v_i$  for  $1 < i \leq P$ . Fig. 2 shows an example for determining the boundary values from a transfer function.



(a) Histogram before equalization.



(b) Histogram after equalization.

Fig. 3. Histograms in a heavy skewed case.

One complication arises in determining boundary values for partitioning in the presence of data skew. If  $g^{-1}(i) = g^{-1}(j)$  for  $i \neq j$ , the processor  $i$  ( $< j$ ) would have much more tuples than other processors and processor  $j$  would have no tuples at all. Fig. 2 shows such complication case where one hashed value maps into different processors. In the case, it is still possible to equalize the histogram of hashed values by uniformly distributing the tuples with the same hashed value to the processors into which the value is mapped. For example, suppose that the hashed values of a join attribute in a relation  $R$  with 9 tuples are {1, 2, 3, 3, 3, 3, 3, 3, 4}, and that we wish to partition over three processors 1, 2, and 3. By taking inverse transformation of equation (3), three boundary values are {3, 3, 4} (see Fig. 2). Thus, tuples {1, 2, 3, 3, 3, 3, 3, 3} are mapped into processor 1, the tuple {4} is mapped into processor 3, and no tuple is mapped into processor 2. However, this is not the case for an even distribution of tuples. To make the distribution to be even, we distribute  $1/6$  of the tuples {3, 3, 3, 3, 3, 3} into processor 1,  $1/2$  into processor 2, and  $1/3$  into processor 3. As will be shown in section 3, partitioning the tuples {3, 3, 3, 3, 3, 3} in such a way is possible because each processor has the cumulative histogram of the hashed values. Finally, for correct join operations, all the tuples of  $S$  with the value should be distributed to the processors in which associated tuples of  $R$  are assigned. Fig. 3 shows histograms for hashed values before and after equalization.

### 3. Parallel join algorithm

The basic procedure for our join algorithm consists of three major phases as follows.

#### Phase 1. Histogram evaluation phase

- 1.1. Each processor reads its portion of  $R$ , hashes the join attribute of each tuple and obtains a local cumulative histogram of the hashed values in parallel.
- 1.2. Each processor broadcasts its own local cumulative histogram, evaluates the global cu-

mulative histogram using received local histograms for  $R$  in parallel.

- 1.3. Each processor determines the boundary values in parallel.

**Phase 2. Partitioning phase.** Each processor partitions and distributes its portion of both relations using the histogram equalization transfer function and the boundary values. As a result of this phase, the corresponding partitions of the two relations that have the same ranged boundary values reside on the same processor.

**Phase 3. Joining phase.** Each processor finally performs the joining step by using the conventional hash-based join algorithm on the partition pairs in parallel.

The I/O accesses from secondary storage and communication cost through an interconnection network become major limiting factors on the performance of parallel join operations. Therefore, the join algorithm should be carefully designed in order to minimize the I/O and communication costs. A useful general observation is that an imbalance in the number of tuples of the smaller relation  $R$  per processor is much worse than an imbalance in the number of tuples of the larger relation  $S$  per processor. This is because an imbalance in the number of building tuples per processor requires extra buckets in the local joins, thus driving up the number of I/Os significantly [8]. The partitioning scheme in our algorithm only attempts to balance tuples of the smaller relation  $R$  per processor to minimize the I/O cost. Thus, in contrast to the PHJA, our partitioning scheme has only one extra scan of each processor's portion of the smaller relation  $R$ . Moreover, our scheme requires additional communication cost for broadcasting the local histograms to each processor.

Note that during the local joining, the size of each bucket should be smaller than the memory capacity. However, nonuniform distribution of the join attribute values may generate *bucket overflow*. The performance in the presence of overflowed buckets diminishes because it requires extra I/O to spool to disk and then re-read to

perform the join [2,8]. We can also use the histogram equalization technique to resolve the bucket overflow. That is, we can adopt the histogram equalization technique for locally partitioning the portion of the relations distributed at each processor into several buckets suitable in memory.

#### 4. Simulation experimentation

Simulation experiments are conducted for performance comparisons of the PHJA and our SRJA. The performance of both algorithms is evaluated in the presence of data skew. It is known that data distribution for many practical situations follows a variant of Zipf's Law [5]. In the Zipf's distribution, the probability of a duplication for the  $i$ th join attribute value over  $N$  possible values in a relation is given by

$$p_i = \frac{c}{i^{(1-\theta)}}, \quad 1 \leq i \leq N,$$

$$\text{where } c = \frac{1}{H_N^{(1-\theta)}}, \quad H_N^{(1-\theta)} = \sum_{i=1}^N \frac{1}{i^{(1-\theta)}}. \quad (4)$$

In the distribution,  $\theta = 1$  corresponds to the uniform distribution, while  $\theta = 0$  corresponds to a highly skewed case.

##### 4.1. Simulation model

In our model, the PHJA and SRJA are simulated to obtain their total execution time. The total execution time comprises the CPU time, the I/O time, and the communication time. The execution time during the  $i$ th phase is sum of the three components:  $T^i = T_{cpu}^i + T_{disk}^i + T_{comm}^i$ . Thus the total execution time for the join operations can be expressed as  $T = \sum_i T^i$ .

The following assumptions are made in simulation experiments: the cost for writing the joined results into disks is not taken into account because this cost has the same effect on each join algorithm; the network has an ability of broadcasting and point-to-point communication with the same transmission cost.

The simulation experiments are performed on

the three cases of samples data distributions:  $\theta = 1$  (uniform distribution case),  $\theta = 0.2$  (mild skew case) and  $\theta = 0$  (heavy skew case). Each data in synthetic sample database for simulation experiments is obtained by random number generation by use of Eq. (4). To evaluate communication cost for the partitioning phase,  $T_{comm}$ , for the three cases, let  $R_i$  and  $T_i$  be the number of pages received from, and transmitted to the processor  $i$ , respectively. Then, communication cost for PHJA and SRJA is given by  $T_{comm} = t_{tm} * \max\{(R_1 + T_1), (R_2 + T_2), \dots, (R_P + T_P)\}$ , where  $t_{tm}$  is time to transfer a page. For the purpose of a variance reduction on mean difference between the simulated costs of two algorithms, the same synthetic database is used to simulate both the PHJA and our SRJA [1].

The parameter values of simulation experiments are as follows. The size of the relation  $S$  is ten times the size of the smaller relation  $R$ . The memory size on each processor is  $M = R/P$ . The domain size for the join attribute of each relation is 10000. The rest of parameters are set to the values shown in Table 1 [9].

Simulation experiments are execution driven. That is, performance is measured while actually executing the join algorithms of PHJA and SRJA. The PHJA is experimented by the following steps. The execution time for the partitioning phase is

Table 1  
Parameter values for simulation experiments

$t_{cm}$	Time to compare with two attributes	3 $\mu$ s
$t_{hs}$	Time to compute a hash function of a key	9 $\mu$ s
$t_{mv}$	Time to move a tuple in memory	20 $\mu$ s
$t_{bj}$	Time to build a join result tuple	10 $\mu$ s
$t_{ad}$	Time to update a variable in memory	4 $\mu$ s
$t_{sd}$	Time for CPU to send a page over network	1 ms
$t_{rv}$	Time for CPU to receive a page over network	1 ms
$t_{io}$	Time to transfer a page between disk and memory	20 ms
$t_{tm}$	Time to transfer a page in network	3 ms
$n$	Number of tuples in a page	100
$h$	Number of elements in histogram table	5000
$H$	Size in pages of histogram table	$h/1000$

calculated in terms of the number of the basic operations by performing actual hash-based partitioning of the synthetic sample database. The execution time for the joining phase is evaluated by calculating the CPU and I/O time costs based on the number of I/O operations on the skewed partition. The total execution time for the PHJA is evaluated by adding the costs of the two phases.

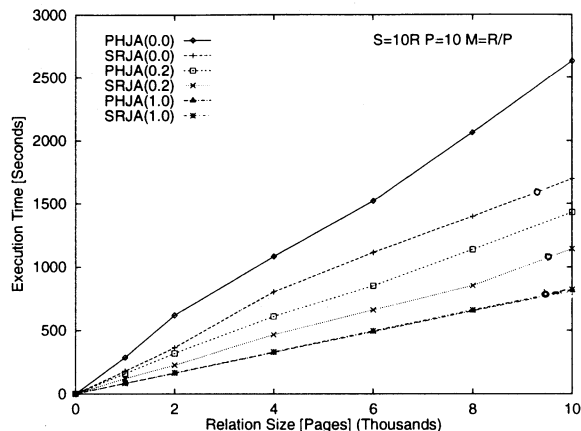
The SRJA is experimented similarly. The execution time for phase 1 and phase 2 of the SRJA is calculated in terms of the number of the basic operations by performing actual partitioning based on the histogram equalization methodology of the synthetic sample database. The execution time for the joining phase is obtained by the same method as used in the PHJA. The total execution time for the SRJA is evaluated by adding the costs of the three phases for each.

#### 4.2. Simulation results

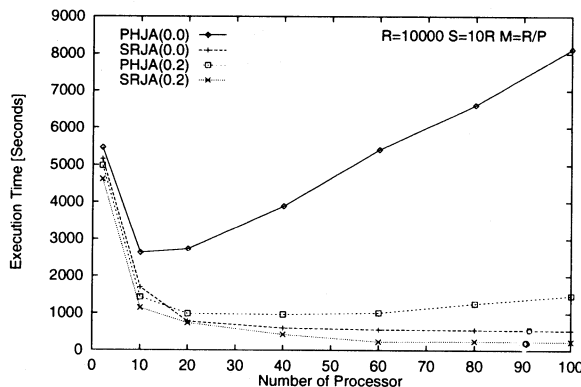
Several simulation experiments are performed by considering the three points of view: the effects of relation sizes, the effects of the system configurations, and the effects of the degree of data skew. Every simulation experiment are performed 10 times with the same parameter values. We take a mean value of the results of ten trials. Figure 4 shows simulation results.

Fig. 4(a) shows the effects of the join algorithms on relation size in the three sample databases: uniform distribution case ( $\theta = 1$ ), mild skew case ( $\theta = 0.2$ ) and heavy skew case ( $\theta = 0$ ). The simulation results show that (1) the total time costs of the PHJA and SRJA are linearly incremented by relation size, (2) the difference in the total time costs between the PHJA and SRJA in the heavy skew case is higher than that of the mild skew case and (3) the difference of the total time costs between the two algorithms in the uniform distribution case is little.

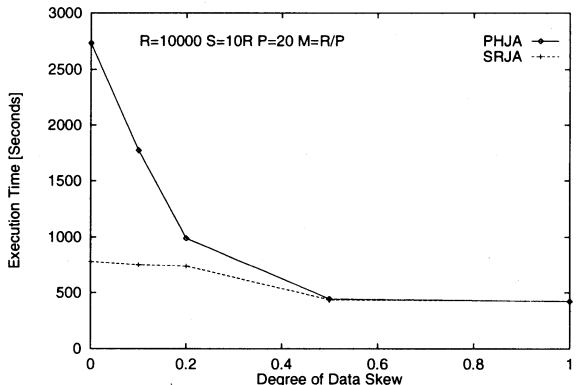
In Fig. 4(b), we show the effects of the join algorithms on the number of processors in two skewed sample databases. As the number of processors becomes large, the performance of the PHJA is rapidly degraded in the heavy skew case because of *bucket overflow* during the local join-



(a) Execution time versus relation size.



(b) Execution time versus number of processors.



(c) Execution time versus degree of data skew.

Fig. 4. Simulation results.

ing phase. In this case, we show that SRJA outperforms the PHJA.

Finally, we show the effects of the degree of data skew in Fig. 4(c). As the degree of data skew becomes larger, performance of the PHJA is de-

graded rapidly. In the worst case, the PHJA may generate one large bucket leading to *load imbalance* and *bucket overflow* which cause the performance degradation of parallel join operations.

## 5. Conclusions

In this paper, we first have proposed a data distribution framework for parallel join. Our data distribution framework employs the histogram equalization technique, which evenly distributes data across processors. We then have proposed an efficient parallel join algorithm based on the data distribution framework which takes data skew into account. Our proposed join algorithm is carefully considered to minimize the I/O and communication costs and is designed to reduce bucket overflow and load imbalance for real world situations.

The performance of the proposed algorithm has been evaluated by simulation experimentation with several synthetic databases and compared with the conventional parallel hash-based join algorithm. Comparison results have shown that the proposed algorithms have better performance than the parallel hash-based join algorithm in the presence of data skew, with negligible overhead in the absence of data skew.

## References

- [1] J. Banks and J.S. Carson, *Discrete-event System Simulation* (Prentice-Hall, Englewood Cliffs, NJ, 1984).
- [2] K.A. Hua and C. Lee, Handling data skew in multiprocessor database computers using partition tuning, in: *Proc. Internat. Conf. on Very Large Database* (1991) 525–535.
- [3] M. Kitsuregawa and Y. Ogawa, Bucket spreading parallel hash: A new robust, parallel hash join method for data skew in the super database computer (SDC), in: *Proc. Internat. Conf. on Very Large Database* (1990) 210–221.
- [4] M.S. Lakshmi and P.S. Yu, Effectiveness of parallel joins, *IEEE Trans. Knowledge and Data Engineering* **2** (1990) 410–424.
- [5] C.A. Lynch, Selectivity estimation and query optimization in large databases with highly skewed distributions of column values, in: *Proc. Internat. Conf. on Very Large Database* (1988) 240–251.
- [6] P. Mishra and M.H. Eich, Join processing in relational databases, *ACM Comput. Surveys* **24** (1992) 63–113.
- [7] A. Papoulis, *Probability, Random Variables, and Stochastic Processes* (Mcgraw-Hill, New York, NY, 1991).
- [8] D.A. Schneider and D.J. DeWitt, A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment, in: *Proc. ACM-SIGMOD Internat. Conf. on Management of Data* (1989) 110–121.
- [9] C.B. Walton, A.G. Dale and R.M. Jenevein, A taxonomy and performance model of data skew effects in parallel joins, in: *Proc. Internat. Conf. on Very Large Database* (1991) 537–548.
- [10] J.L. Wolf, D.M. Dias, P.S. Yu and J. Turek, An effective algorithm for parallelizing hash joins in the presence of data skew, in: *Proc. Internat. Conf. on Data Engineering* (1991) 200–209.