데이터 편재 하에서 히스토그램 변환기법에 기초한 효율적인 병렬 해쉬 결합 알고리즘

박 웅 규[†] · 최 황 규[††] · 김 탁 곤[†††]

요 약

본 논문은 병렬 해쉬 결합 연산에서 데이터의 분산 시에 나타나는 부하의 불균형과 버켓 오버플로우를 해결하기 위한 새로운 데이터 분산 방법을 제안한다. 제안된 데이터 분산 방법은 편재된 분포를 갖는 데이터들을 히스토그램 변환기법에 의하여 각 노드의 성능에 따라 균일하게 분산시킨다. 또한 본 논문에서는 이 분산 방법을 병렬 해쉬 결합 연산에 적용하여 데이터 편재에 따른 성능 저하를 해결할 수 있는 알고리즘을 제안하고, 기존의 병렬 결합 알고리즘들과의 성능 비교를 위하여 모의 실험과 COREDB 병렬 데이터베이스 컴퓨터 상에서의 실험을 통하여 편재된 분포를 갖는 데이터에 대하여 성능 분석을 수행한다. 실험 결과에서 편재된 데이터에 대하여 기존의 다른 결합 연산 알고리즘보다 제안된 알고리즘이 우수한 성능을 나타냄을 보인다.

Effective Parallel Hash Join Algorithm Based on Histogram Equalization in the Presence of Data Skew

Ung Kyu Park[†] · Hwang Kyu Choi[††] · Tag Gon Kim[†††]

ABSTRACT

In this paper, we first propose a data distribution framework to resolve load imbalance and bucket overflow in parallel hash join. Using the histogram equalization technique, the framework transforms a histogram of skewed data to the desired uniform distribution that corresponds to the relative computing power of node processors in the system. Next we propose an efficient parallel hash join algorithm for handling skewed data based on the proposed data distribution methodology. For performance comparison of our algorithm with other hash join algorithms, we perform simulation experiments and actual execution on COREDB database computer with 8-node hypercube architecture. In these experiments, skewed data distribution of the join attribute is modeled using a Zipf-like distribution. The performance studies indicate that our algorithm outperforms other algorithms in the skewed cases.

1. Introduction

As performance becomes a critical issue in large database applications, parallel database systems are receiving increasing attention in both theory and practice. 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

※본 논문은 한국과학재단 1994년 상반기 Post Doc. 연구지원 결과의 일부임.
† 정 회 원:서원대학교 전자계산학과
†† 정 회 원:강원대학교 컴퓨터공학과
††† 정 회 원:한국과학기술원 전기 및 전자공학과
논문접수:1996년 8월 13일, 심사완료:1996년 11월 8일

database systems[6, 8, 10]. Among them, the parallel hash join algorithm(PHJA) has been found to be superior to other join algorithms for the uniform distribution of data[6, 10].

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[7]. With such data distribution, the PHJA shows two major problems in performance: *load imbalance and bucket overflow*[3]. 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 [6, 11], most algorithms proposed for the PHJA limit exploiting parallelism as the skewness of data distribution becomes large.

This paper proposes an efficient algorithm, called *skew 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. For performance comparison of our algorithm with other hash join algorithms, we perform simulation experiments and actual execution on the COREDB database computer with 8-node hypercube architecture. The performance studies indicate that our algorithm exhibits better performance compared with other algorithms in the skewed cases.

In this paper, the histogram transformation technique used for data partitioning of our proposed join algorithm is described in chapter 2. We discuss our data distribution framework and parallel join algorithm based on the histogram transformation in chapter 3. In chapter 4 and 5, the performance of our proposed join algorithm is evaluated and compared with other parallel hash join algorithms by using simulation and experimentation on COREDB database computer. Finally, conclusions are discussed in chapter 6.

2. 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 $x_1, x_2, \cdots, x_J (x_1 \le x_2 \le \cdots \le x_J)$, is mapped into an output random variable $\bar{y}$, ranged over $y_1, y_2, \cdots, y_K$ $(y_1 \le y_2 \le \cdots \le y_K)$, such that the output probability density follows a uniform density.

Since a histogram of discrete random variables can be approximated 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[9] follows that $f_{\bar{y}}(y)=\dfrac{f_{\bar{x}}(x)}{g'(x)}$, 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 $F_{\bar{x}}(x)=P(\bar{x}\le x)$ 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, $f_{\bar{y}}(y)=\dfrac{1}{y_K-y_1}$ for $y_1 \le y \le y_K$, the histogram equalization transfer function becomes

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

Let us now return to the discrete case. Suppose that

$H_{\bar{x}}(x)$ for $x=x_1, x_2,\cdots,x_J$ represents the fractional number of occurrence frequencies 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 equation (1) can be modified by

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

3. New Parallel Hash Join Algorithm

The following assumptions are made in the remainder of this paper. The parallel database system has P autonomous processors numbered by 1, 2,..., 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.

3.1 Data Distribution Framework for Load Balancing

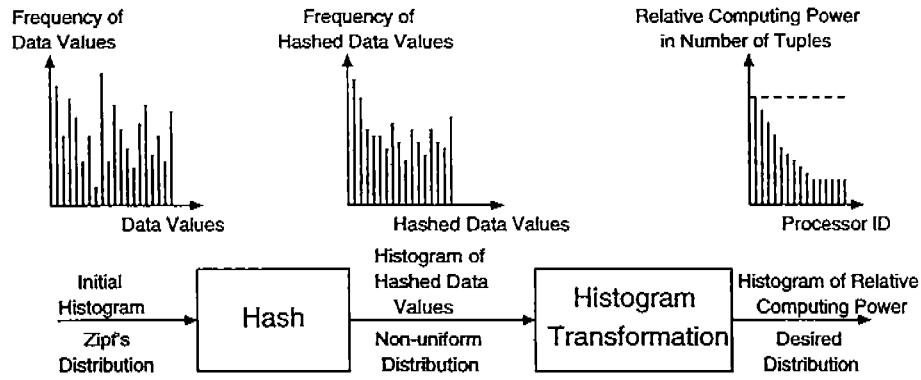
Our data distribution framework using histogram equalization methodology for parallel joins is shown in Fig. 1. Initially, we have a histogram of data values of the join attribute for a relation. Then, we hash the data values. Finally, the histogram of the hashed values is transformed into a uniformly distributed histogram using equation (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,..., P. Then the histogram equalization transfer function is obtained from equation (2) as

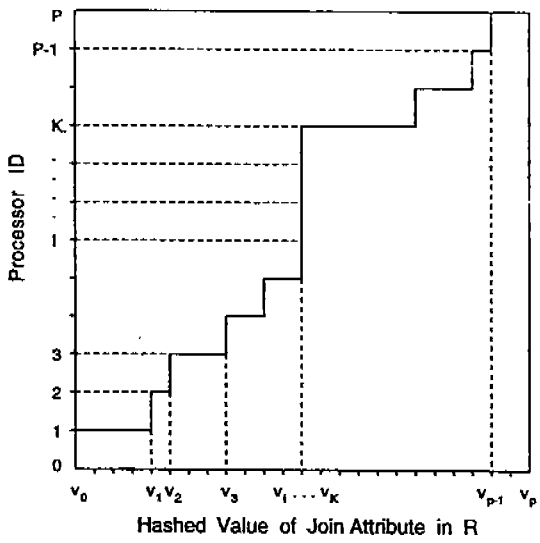
$$y=g(x)=\lceil P\sum_{m=0}^{x} H_{\bar{x}}(m)\rceil \quad (3)$$

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

Since the transfer function in equation (3) is monotonically an increasing function, an upper boundary value, x, for each partition is obtained by inverse transformation of equation (3). That is, the upper boundary value of the ith processor can be obtained by $g^{-1}(i)$. Let the P boundary values be $v_1, v_2,\cdots,v_P$ 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 $v_{i-1} < a \le v_i$ for $1 < i \le P$ and, in particular, into the range $v_0 \le a \le v_1$ for processor 1. Fig. 2 shows an example for determining the boundary



(Fig. 1) Data distribution framework for parallel joins.



(Fig. 2) Transfer function for histogram equalization.

values from a transfer function.

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 ,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. 2, 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.

3.2 Basic Procedure of Skew Resolution Join Algorithm

The basic procedure for our join algorithm, called skew resolution join algorithm(SRJA), 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 cumulative 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[10]. 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 diminishes in the presence of overflowed buckets because it requires an extra I/O to spool to disk and then re-read to perform the join[3, 10]. 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 following four join algorithms: the parallel hash join algorithm(PHJA), our skew resolution join algorithm(SRJA), the bucket-spreading hash join algorithm(BSJA) of [5], and the load balancing join algorithm(LBJA) of [3]. The BSJA[5] and the LBJA[3] are algorithms designed for handling data skew. The performance of the algorithms is evaluated in the presence of data skew.

4.1 Simulation Model

In our model, the four join algorithms 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 ith phase is sum of the three components: $T^i = T^i_{cpu} + T^i_{disk} + T^i_{comm}$. 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.

Since it is known that data distribution for many practical situations follows a variant of Zipf's Law[7], our experiments use synthetic sample databases following the Zipf's distribution. In the Zipf's distribution, the probability of a duplication for the ith join attribute value over N possible values in a relation is given by

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

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

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 the equation (4). 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 the join algorithms[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[11].

〈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	$1mS$
t^{rv}	Time for CPU to receive a page over network	$1mS$
t^{io}	Time to transfer a page between disk and memory	$20mS$
t^{tm}	Time to transfer a page in network	$3mS$
n	Number of tuples in a page	100
h	Number of elements in histogram table	1000
H	Size in pages of histogram table	$h/1000$

4.2 Simulation Method

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

Our performance evaluation requires many simulation runs with different data sets for the same parameter values. This is because data partition based on a hash function depends greatly on the data values of the join attribute in the data sets. For improving precision of experimental results, every simulation experiments are performed 10 times with the same parameter values. We take a mean value of the results of ten trials.

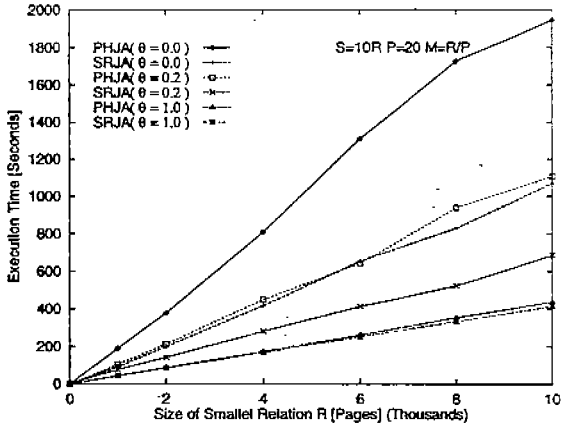
4.3 Simulation Results

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. Fig. 3 shows simulation results.

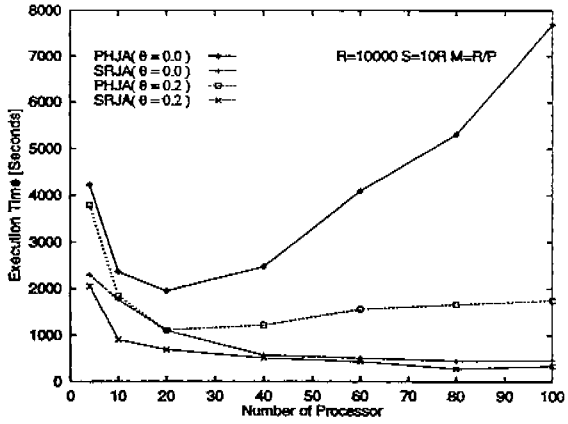
Fig. 3(a) shows the effects of the PHJA and the SRJA on relation size in the three sample databases: uniform distribution case($\theta = 1$), mild skew case ($\theta = 0.2$) and heavy skew case($\theta = 0$). The simulation results show that 1) the total time costs of the PHJA and SRJA are linearly incremented 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. 3(b), we show the effects of the PHJA and the SRJA 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 joining phase. In this case, we show that SRJA outperforms the PHJA.

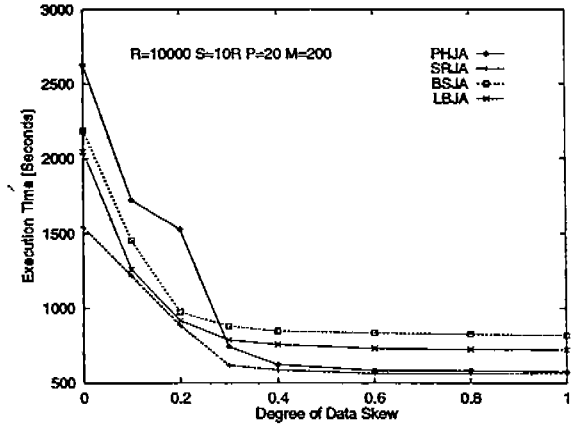
Fig. 3(c) shows the effects of the degree of data skew in the PHJA, the SRJA, the BSJA, and the LBJA. As the degree of data skew becomes larger, performances of the PHJA, the BSJA, and the LBJA are degraded rapidly. But performance of the SRJA is not degraded as much as them. In the figure, we observe that 1) our algorithm outperforms other skew handling algorithms(the BSJA and the LBJA)[3, 5] in



(a) Execution time versus relation size.



(b) Execution time versus number of processors.

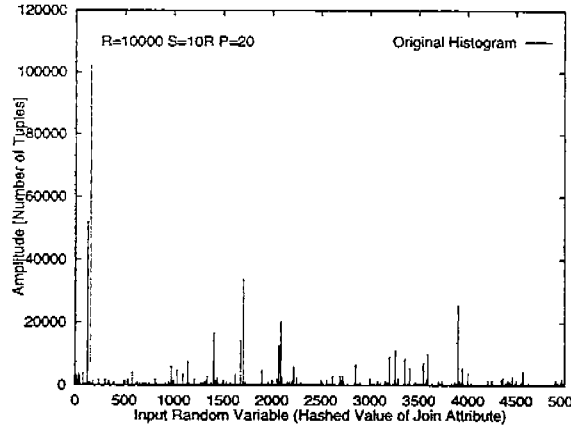


(c) Execution time versus degree of data skew.

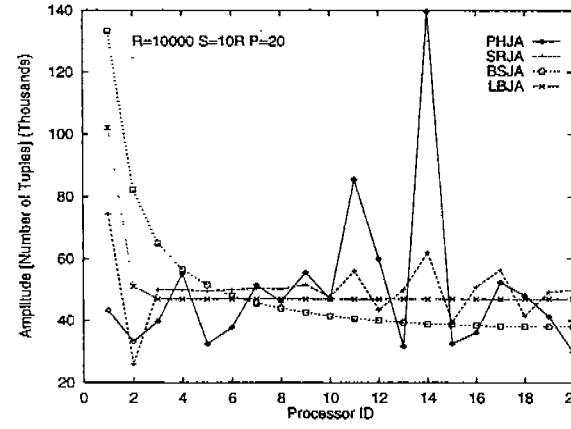
(Fig. 3) Simulation results.

all the cases, 2) in contrast to the PHJA, the BSJA and the LBJA performs well in the higher skewed cases, but the performance of the BSJA and the LBJA exhibits worse than that of the PHJA in the slightly skewed case and the uniform distribution case. This is because the BSJA and the LBJA require more processing time for CPU, disk I/O, and communication to manage data skew.

Fig. 4 shows histograms for hashed values before and after partitioning using the four join algorithms in the heavy skewed case. For the smaller relation R,



(a) Histogram before partitioning.



(b) Histogram after partitioning.

(Fig. 4) Histograms for smaller relation R in a heavy skewed case.

our algorithm gives relatively evenly allocated partitions among all the processors. However, other algorithms generate very large skewed partitions, leading to *load imbalance* and *bucket overflow* which cause performance degradation of parallel join algorithms. We also observe that Hua's algorithm(the LBJA) gives relatively smaller skewed partitions than Kitsurega's algorithm(the BSJA) and the PHJA.

5. Experimentation on A Hypercube Database Computer

Since the system parameters differ greatly from machine to macchine, the simulation model decribed previously allows us to perform sensitivity analysis with respect to various combinations of system and workload parameters. Therefore, the simulation results will be useful in projecting the performance of the proposed algorithm on various hardware platforms.

Nevertheless, to investigate the performance of the PHJA and the SRJA in real database system and provide more faith of simulation experiments, we implemented our algorithm in COREDB[4], which is a 3-dimensional hypercube machine developed by KAIST (Korea Advanced Institute of Science and Technology)'s Computer Research Engineering Laboratory. Currently, each node of the machine consists of a 68030 CPU, 8Mbytes DRAM and disk controllers. The operating system is UNIX System V release 3.

5.1 Experiments

For our experiments, we used benchmark relations with the join attribute generated from Zipf's distribution[2]. Table 2 shows the benchmark relation schema. As in simulation experiments, experiments are conducted on the three cases of sample data distributions: $\theta=1$(uniform distribution case), $\theta=0.2$ (mild skew case), and $\theta=0$(heavy skew case). Each synthetic sample database for the experiments is obtained by a random number generation by use of the equation (4). We also generate both relations with the same tuple distribution.

〈Table 2〉 Benchmark relation schema.

Attribute	Type	Range of Values	Order	Comment
Partition	Integer	0-9999	Sequential	Unique
Join	Integer	0-199	Random	Duplicate Zipf Distribution
String	Character Array		Fixed	100 Bytes in Size

The parameter values for the experiments are as follows. The size of the smaller relation R is 2000 tuples. The size of the larger relation S is five times the size of the smaller relation R. The size of each tuple in both relations is 108 bytes. Thus total size of the relation R is about 200 KBytes and that of the relation S is about 1 MBytes. The domain size for the join attribute is 200 and the number of distinct hashed values in the histogram table for the SRJA is 50.

5.2 Experimental Results

We performed the experiments for the three cases on the 4-node and 8-node configurations. Fig. 5 shows our experimental results. In the figure, partitioning time for the SRJA is obtained by adding costs for the histogram evaluation phase and the partitioning phase of the SRJA. From the results, we observe that 1) time cost for the local join operations is a dominating factor on performance of the parallel join algorithms in the skewed cases, 2) the SRJA outperforms the PHJA in the skewed cases.

To show the relative efficiencies of the two join algorithms, we measure the performance gain of the SRJA over the PHJA as follows:

$$[\text{Performance Gain} = \frac{T_{PHJA} - T_{SRJA}}{T_{PHJA}} \times 100\%]$$

In the equation, T_{PHJA} and T_{SRJA} are the total execution of the PHJA and SRJA, respectively. Table 3 shows the performance gain measured from the

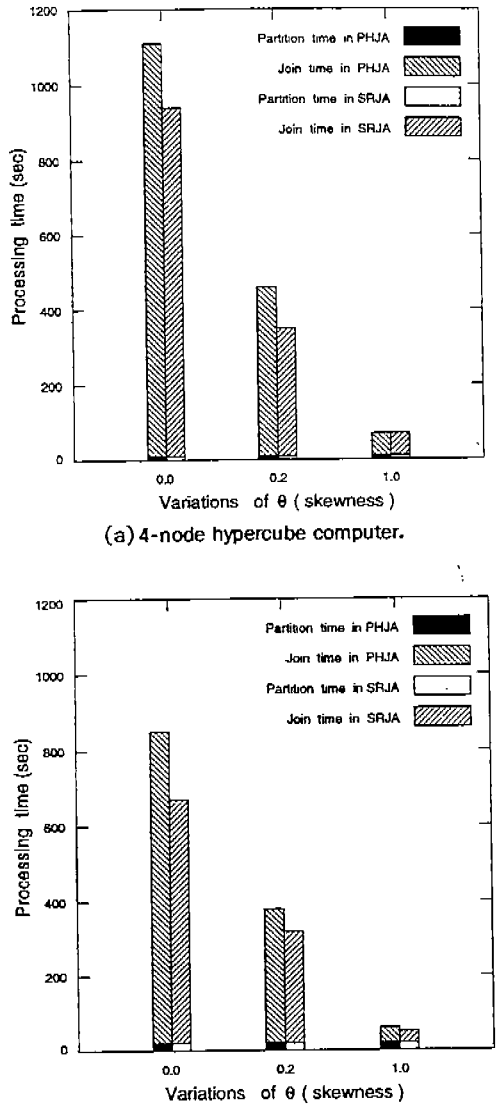
experimental results. In the table, we observe that in the heavy skew case($\theta=0.0$), as the dimension of hypercube increases, the performance gain is increased. This is because the PHJA gives a highly skewed partition in the heavy skew case and the size of such partition is not linearly decreased as the number of node processors in the system increases. However, the size of the maximum skewed partition in the SRJA is linearly decreased as the number of node processors increases. In the mild skew case($\theta=0.2$), as the dimension of hypercube increases, the performance gain is reduced. This is because the size of the maximum skewed partition in both the PHJA and the SRJA is linearly decreased as the number of node processors increases.

〈Table 3〉 Experimental results : performance gain.

	$\theta=0.0$	$\theta=0.2$	$\theta=1.0$
4-node	16.39	24.58	−0.02
8-node	23.58	13.81	0.05

5.3 Simulation Model Validation

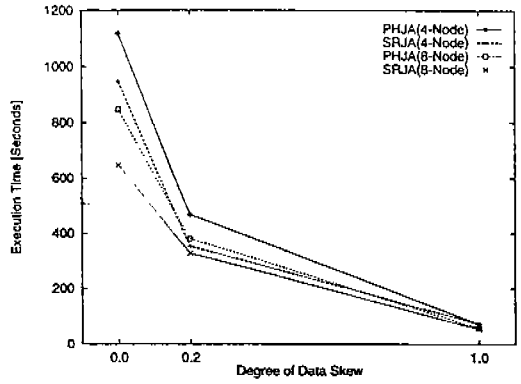
Experimental results obtained from implementation of our algorithm on the COREDB are useful to validate our simulation model. The results of the experiments are shown in Fig. 6(a). The same workload and system parameters are used to derive the corresponding simulation results which are shown in Fig. 6(b) for comparison. We observes that the experimental results and the simulation results behave very similarly. We also observe that the running times of the two algorithms are higher in the implementation experiments. This is due to the fact that overhead derived from virtual memory envioronment of UNIX is assumed to be ignored in our simulation model. The COREDB has relatively small size of the local memory capacity for supporting the join operations with large databases. Thus, the large join results generated from the skewed data lead to many operations for swap-in and swap-out in the virtual memory environments.



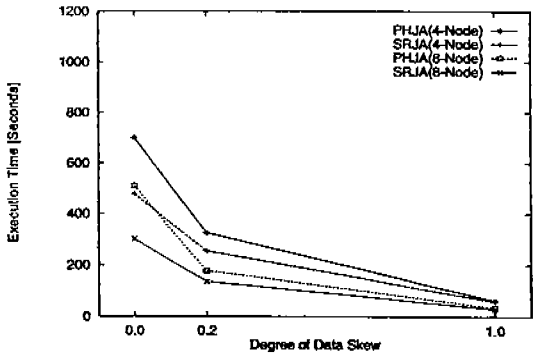
(Fig. 5) Experimental results : execution time versus system configuration.

6. Conclusions

In this paper, we first have proposed a data distribution framework for load balancing in parallel hash join. Our data distribution framework employs the



(a) Implementation Results.



(b) Simulation Results.

(Fig. 6) Validation of Simulation Model.

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 join algorithm has been evaluated by simulation experiments and actual execution in a hypercube database computer with several synthetic databases and compared with the other parallel hash join algorithms. Comparison results have shown that the proposed algorithm has better performance than the other algorithms in the skewed cases, 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] D. Bitton, D.J. DeWitt, and C. Turbyfill, Benchmarking database system:a systematic approach, In Proc. of Internat. Conf. on Very Large Database pp. 8-19, 1985.

[3] K.A. Hua and C. Lee, Handling data skew in multiprocessor database computers using partition tuning, In Proc. of Internat. Conf. on Very Large Database, pp. 525-535, 1991.

[4] Y.C. Kim, et al., A hypercube database computer-COREDB, In Proc. of 1993 UNIX Symposium, pp. 449-459, 1993.

[5] 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. of Internat. Conf. on Very Large Database, pp. 210-221, 1990.

[6] M.S. Lakshmi and P.S. Yu, Effectiveness of parallel joins, IEEE Trans. Knowledge and Data Engineering 2(4), pp. 410-424, 1990.

[7] C.A. Lynch, Selectivity estimation and query optimization in large databases with highly skewed distributions of column values, In Proc. of Internat. Conf. on Very Large Database pp. 240-251, 1988.

[8] P. Mishra and M.H. Eich, Join processing in relational databases, ACM Computing Surveys, 24(1), pp. 63-113, 1992.

[9] A. Papoulis, Probability, Random Variables, and Stochastic Processes, Mcgraw-Hill, New York, NY, 1991.

[10] D.A. Schneider and D.J. DeWitt, A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment, In Proc. of ACM-SIGMOD Internat. Conf. on

Management of Data, pp. 110-121, 1989.

[11] C.B. Walton, A.G. Dale, and R.M. Jenevein, A taxonomy and performance model of data skew effects in parallel joins, In Proc. of Internat. Conf. on Very Large Database, pp. 537-548, 1991.

박 웅 규

1984년 서강대학교 전자공학과(학사)
1986년 한국과학기술원 전기 및 전자공학과(석사)
1995년 한국과학기술원 전기 및 전자공학과(박사)
1986년~1991년 한국전자통신연구소 컴퓨터개발부 연구원
1991년~현재 서원대학교 전산학과 조교수
관심분야: 병렬 데이타베이스 시스템, 멀티미디어 서버 시스템, 음성합성

최 황 규

1984년 경북대학교 전자공학과(학사)
1986년 한국과학기술원 전기 및 전자공학과(석사)
1989년 한국과학기술원 전기 및 전자공학과(박사)
1990년 3월~1993년 2월 강원대학교 전자공학과 조교수
1993년 3월~현재 강원대학교 컴퓨터공학과 부교수
1994년 7월~1995년 7월 Univ. of Florida Database R&D Center 방문교수
관심분야: 병렬 데이터베이스 시스템, 멀티미디어 저장 시스템, 병렬 I/O 시스템, 실시간 데이터 베이스 시스템

김 탁 곤

1975년 부산대학교 전자공학과(학사)
1980년 경북대학교 전자공학과(석사)
1988년 Department of Computer Engineering, University of Arizona, Tucson(박사)
1980년~1983년 국립 수산대학교 통신공학과 교수
1987년~1989년 Department of Electric and Computer Engineering, University of Kansas, Lawrence 교수
1991년 9월~현재 한국과학기술원 전기 및 전자공학과 교수
관심분야: Discrere-event system modeling/simulation, Computer system architecture, Software engineering methodology