Proceedings of the 1994 Summer Computer Simulation Conference
ISBN: 1-56555-029-3

PERFORMANCE EVALUATION OF QUANTILE-SELECTION BASED PARTITION FOR PARALLEL JOINS IN HYPERCUBE COMPUTER SYSTEM

Heung-Shik Kim† and Tag Gon Kim‡

† ‡ Department of Electrical Engineering
Korea Advanced Institute of Science and Technology
373-1 Kusung-Dong, Yusung-Ku, Taejon, 305-701, Korea

† Department of Computer Science
Inje University
Kimhae, Kyungnam, 621-749, Korea

ABSTRACT

The effectiveness of parallel join operations largely depends on data distributions among processors in a parallel database machine. Several parallel join algorithms are proposed for parallel relational database systems, and the parallel hash-based join algorithm(PHJA) is the most commonly used one. However, all algorithms including PHJA may limit the degree of parallelism in execution as the skewness of data distributions becomes large. This paper proposes a new partitioning algorithm for parallel join operations in a hypercube database system. Based on quantile selection, the algorithm makes data distribution to be approximately even among node processors in the hypercube. Performance of the parallel join algorithms based on the proposed partitioning algorithm is evaluated and compared with the hash-based parallel join algorithm.

Keywords : partition, selection, quantile elements, parallel join, hypercube

1 Introduction

Database management systems are covered with the efficient management and processing of very large number of data elements. As performance becomes a critical issue in large database applications, parallel database machines are receiving increasing attention in both theory and practice. Relational join operations are the most important part of the query processing in the relational database systems. Efficient algorithms are necessary for the good performance. Parallel processing of the relational join operations improves response time.

When a parallel database system distributes data unevenly across processors, load imbalance results in performance degradation. However, when the tuples to be joined are evenly distributed across all the processors, the join operations have the best performance. But many partitioning algorithms do not guarantee even partition.

Recently many parallel join algorithms have been studied on hypercube architectures(Fried 1990; Omiecinski and Lin 1989; Omiecinski and Lin 1992). (Omiecinski and Lin 1989, Omiecinski and Lin 1992) presented the parallel hash-partitioned join algorithm. It partitions the relations **R** and **S** into disjoint subsets called buckets and assigns the buckets to each processor, then join in parallel. But it does not guarantee the even partitioning. Fried 1990) presented the nearest neighbor pairing algorithm. During n pairing step, each node partitions its tuple into two. The first set consists of all tuples whose attribute values are assigned to hypercube nodes whose j-th bit address is 0. All remaining tuples are placed in other set. Then each node send the second set to the neighbor node. After n pairing step, each node proceeds joining in parallel. (Chang et al. 1991) proposed the select-partitioned join in uni-processor environments. For selecting bound values, it uses the cumulative distribution function.

This paper presents a new partitioning method which guarantee about even partitioning in a hypercube computer system. It also proposes a parallel join algorithm based on this partitioning method. To do that, we first show that the quantile median is approximately equal to the median of total elements, therefore the quantile median splits the total set into two subsets of almost same size. Error rate of the quantile median does not exceed 0.2 percent. Next we apply this quantile median into multi-partitioning elements in a hypercube computer system to partition the two relations **R** and **S** evenly across each node.

2 Partitioning using Quantile Elements

Consider that m is the capacity of a buffer for which reads and writes blocks from a disk into internal storage. Then, extracting median element whenever we read m elements have some improvements in performance(Schönage et al. 1976). In this paper we extract q quantile elements in m elements of one reading block. Therefore we can use the multi-selection(Dobkin and Munro 1981) and well-known several selection algorithms(Blum et al. 1973;Floyd and Rivest 1975;Motoki 1982;Park and Kim 1989;Yu et al. 1978).

The following parameters and operators(Blum et al. 1973) will be used henceforth.

X	a set of n distinct elements, $\|\|X\|\| = n$.
$\|\|X\|\|$	a cardinality of the set X.
m	the number of elements suitable to the capacity of the internal storage.
S_i	the sorted segment which has m elements in maximum.
p	the number of sorted segments, that is $p = \lceil n/m \rceil$.
q	the number of quantile elements in a sorted segment which has c elements, and it is assumed to be a divisor of c.
Y	a set of the quantile elements.
y_{ij}	the j-th quantile element in the i-th sorted segment (S_i) for $1 \le i \le r,\ 1 \le j \le q$.
$i\theta X$	the i-th smallest element of set X for $1 \le i \le \|\|X\|\|$.
$x\rho X$	the rank of x in set X for $x \in X$, $(x\rho X)\theta X = x$.

Definition 2.1 A column is a set of m sorted elements.

Definition 2.2 Quantile elements are the boundary elements which divide one column into several subsegments of same size(Figure 2.1).

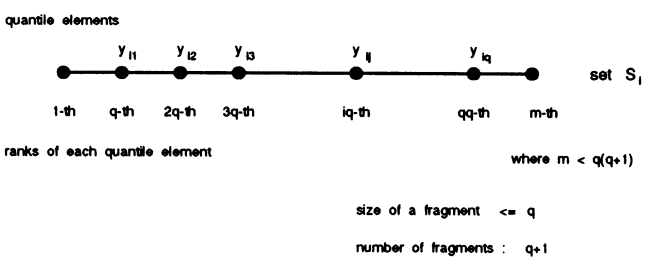

Figure 2.1. Quantile elements in c sorted segment(X_c).

When m elements are read from a disk the internal storage, one column consists of m elements. Then this column is divided into $(q+1)$ subsegments of same size by means of q quantile elements. A column consists of $(\lfloor (m+1)/(q+1) \rfloor (q+1) - 1)$ elements. Let c be the number of elements which is processed at one time in internal storage for finding the quantile elements.Then $c \leq m$. Symbol s is declared on each subsegment of a column, and the size of s is $(c+1)/(q+1)$. Each i-th quantile element is the is-th smallest element in c elements($1 \leq i \leq q$). Therefore the i-th quantile element is greater than $((i-1)s + (s-1))$ elements and less than $((q-i+1)s - 1)$ elements in c elements(Figure 2.1). Especially it is less than $(c - is)$ elements in case of the last column because the size c of this column is not always equal to $((q+1)s - 1)$. Therefore if we define y_i as the i-th smallest quantile element and X_c as the set of c elements in this column, then we generally obtain the following by considering the last column.

$$y_i = is\ \theta\ X_c \quad \text{and} \quad is \leq (y_i\ \rho\ X_c) < (c - (q-i)s)$$

where $y_1 < y_2 < \cdots < y_q$.

The number of columns in n elements is $p = \lfloor n/c \rfloor$. If we extract q elements from one column, the total number of the extracted quantile elements in n elements are pq. Let us define these pq elements be the set Y. Then, the rank of the j-th smallest element of the set Y in the set X is given as the following lemma.

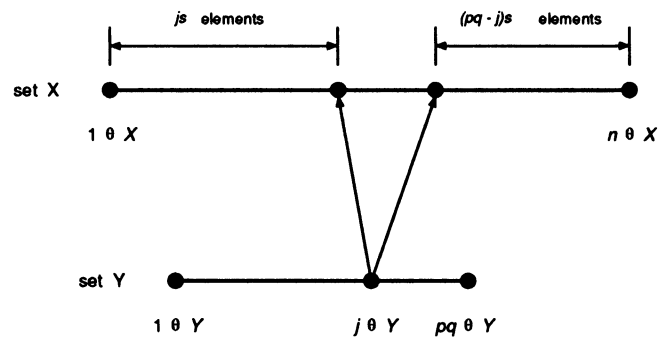

Figure 2.2. Rank of $(j\ \theta\ Y)$ in set X.

Lemma 2.1 The rank of the $j\theta Y$ in the set X is as follows.

$$js \leq (j\ \theta\ Y)\ \rho\ X < n - (pq - j)s.$$

Proof. From the property of the quantile elements, the value of each element of the set Y is at least greater than or equal to each value of the s elements in the set X. If we define the y_j as the j-th smallest element in the set Y, i.e. $y_j = j\ \theta\ Y$, the element y_j is greater than $(j-1)s$ elements and greater than or equal to js elements(Figure 2.2). Therefore

$$js \leq y_j\ \rho\ X.$$

Also y_j is less than $(pq - j)$ elements in the set Y. Consequently it is less than $(pq - j)s$ elements in the set X. Therefore the rank of y_j in the set X is obtained by substracting elements, that is greater than the y_j, from all n elements(Figure 2.2). As a result,

$$y_j\ \rho\ X < n - (pq - j)s.$$

◇

2.1 Median of Quantile Elements

If we are to divide a large number elements into two subsets of same size and maintaining the range, we firstly have to find its median element. If all data are processed by any one processor in a parallel database system, performance is degraded by load imbalance.

There are two partitioning methods : the random method and the hash method. Random partition is very simple, but join operations, which need data elements of same range, must process the local data elements as well as the other processors' data elements. It needs additional I/O and communication overhead between processors for data redistribution. It is too disagreeable to our application for join operation. The hash-based partitioning method uses a hashing function. In this case there is no necessity for redistributing the data elements, but it may not guarantee even partitioning. If data are skewed to some processors in the worst case rehashing is required for even partitioning.

In this section we propose a partitioning method which guarantee approximately even partitioning, and range preserving. In this method, we firstly obtain the median element using a quantile selection. Next we divide total elements into two sorted sets on the basis of this quantile median element. Elements in one subset are less than or equal to this quantile median element and others are greater than this element. This partitioning method preserves the order of total elements and also partitions evenly.

We assume a shared-nothing architecture with two processors, p_1 and p_2, each having their local memory and a hard disk. And the size of data elements in processor p_1 is n_1, the size of data elements in processor p_2 in n_2, and n_1 and n_2 are much larger than their internal memory. At this time the median element of total data elements, n_1 and n_2, is the m-th elements of $(n_1 + n_2)$ elements.

$$\begin{aligned} m &= \lfloor \frac{n_1 + n_2}{2} \rfloor \\ E_m &= m\text{-th element that is a median element.} \end{aligned}$$

Therefore when data are partitioned in the basis of this median element, the processor p_1 has the elements which are smaller than or equal to this element and the processor p_2 has the elements which are larger than this median element.

Because n_1 and n_2 are much larger than internal storage, finding the median element by sorting $(n_1 + n_2)$ elements needs many disk I/Os thus being inefficient. Therefore in any one processor, we read the data elements suitable to process in internal storage at a time into a segment, and then select quantile elements from a segment using multiselection. For that we first obtain the quantile elements until all number of data elements in secondary storage are read. Concurrently we obtain the quantile elements in other processor by the same method. Next, we collect the all quantile elements into any one processors. Next we find a median element from these total quantile elements. Finally we read all n_1 data elements in p_1 and transfer all data elements that are larger than the quantile median element, into the p_2. We read all n_2 data elements in p_2 and transfer all data elements that are smaller than or equal to the quantile median element into the p_1. In this case we know that the data elements of two processors are divided into two groups of approximately same size.

Algorithm *QUANTILE_SELECT(X, n, Y)*

Input : A set of X of n distinct elements in secondary storage.

Output : median element of the set of quantile elements.

Step 1 : Initialize the size of a subsegment in a segment and the number of segments.

$$\begin{aligned} s &= \lceil \sqrt{m} - 1 \rceil\ ; \\ p &= \lceil n/m \rceil\ ; \end{aligned}$$

Step 2 : Repeat following substeps r times:

Step 2.1 : Read c elements into a segment : if the number of remained elements is greater than or equal to m, then $c = m$. Otherwise $c < m$.

Step 2.2 : Calculate the number of quantile elements in a segment :

$$\begin{array}{lll} \text{until} & (p-1)\text{step}, & j = s; \\ \text{last} & p - \text{th step}, & j = \lfloor c/s \rfloor; \end{array}$$

Step 2.3 : Call *MULTI_SELECT*$(c, k_1, k_2, \ldots, k_j)$: multi-select from a segment.

k_1 is s-th element,
k_2 is $2s$-th element,
and k_j is js-th element.

Step 2.4 : Call *ATTACH*$(Y, k_1, k_2, \ldots, k_j)$: add the quantile elements of a segment to the set of quantile elements.

Step 3 : Find median elements from Y, and return the median :

$$\text{median} = \lfloor \|Y\|/2 \rfloor - \text{th element}.$$

2.2 Partitioning Elements by Quantile Multi-select

We examine that the quantile median element divides total elements into two groups with about same size. Therefore in order to examine the propriety, we describe the computational and experimental error limit by comparing a median element by quantile selection with a median element of total elements. If the number of quantile elements is the same as that of total elements, then the error limit becomes zero. But time to obtain the median element with minimum error limit is too long. If the number of quantile elements is very small, then error limit is very large. In this case partitioning elements is of meaningless. Therefore the optimum number of quantile elements are $(q-1)s$ where $q = \sqrt{m}$. We know that the median selection time is minimum when $q = \sqrt{m}$(Kim et al. 1992).

Experimental exceeding error rate of the median of quantile elements compared with the median of the elements is

$$exceeding\_error = \frac{m_t \rho X - m_q \rho X}{\|X\|/2} \times 100\%$$

where m_t and m_q are median elements of total and quantile elements, respectively. The exceeding error varies by the amount of total elements and the size of one segment. It is approximately in range of $\pm 0.2\%$. Experimental method is as follows. We firstly read elements into segments at a time, obtain the quantile elements of each segments, and gather these elements to the set of quantile elements Y. For the next time we find the median from this set of quantile elements. Then we examine the rank of quantile median with respect to total elements. For any randomly generated data elements, we experiment several times with same method and different sets of data, and obtain the maximum and average error rates of several cases as shown in Table 1.

Number of Elements	S = 1023		S = 2499		S = 4095	
	E_{max}	E_{ave}	E_{max}	E_{ave}	E_{max}	E_{ave}
10000	0.575	0.186	0.385	0.134	0.570	0.267
50000	0.222	0.056	0.189	0.061	0.253	0.098
100000	0.131	0.044	0.164	0.037	0.148	0.041
200000	0.099	0.025	0.103	0.031	0.108	0.031

Table 1. Several case of experiments.

For example, assume the case where the number of total data elements is 50000, and the size of a segment is 1023 $(= 32^2 - 1)$. The size of quantile elements is 1516. Therefore the 758-th smallest element is the quantile median element. Now we can find the rank of quantile median element in total elements for several data as follows: 25001-th, 25112-th, 25126-th, 25126-th, 25014-th, 25000-th, 25119-th, 24933-th, $\cdots$, 25053-th and 25136-th smallest elements.

In this sample we know that the quantile median element is in the range of $\pm 0.056\%$ of total median element, and that elements approximately evenly partitioned into two parts by this median element. We now describe analytically.

Theorem 3.1 The rank of quantile median element is within one fragmentation, $\pm f$, of the rank of total median element, and it is in $\pm \frac{f}{n/2} \times 100\%$ error rate.

Proof We prove the theorem by using the notations in section 2. Assume that a quantile median element of the set Y is y_{ij}. Then, let us examine the range of y_{ij} in the set X.
In the sorted Y, there exists the order of y_{ab}, y_{ij}, and y_{uv}, such as y_{ab} is the largest element of

$$y_{ij-1} < y_{ab} < y_{ij} \quad \text{and} \quad y_{ab+1} > y_{ij}.$$

Therefore the maximum number of elements among y_{ab} and y_{ij} is one fragmentation. And y_{uv} is the smallest element of

$$\begin{array}{c} y_{ij} < y_{uv} < y_{ij+1} \quad \text{and} \\ y_{uv-1} < y_{ij} \quad \text{and} \quad y_{uv-1} < y_{ab}, \quad \text{and} \\ y_{uv} < y_{ab+1}. \end{array}$$

Also the maximum number of elements among y_{ij} and y_{uv} is one fragmentation.
Therefore there exists y_{ab} such that

$$y_{ij-1} < y_{ab}, \quad y_{uv-1} < y_{ab} \quad \text{and} \quad y_{ab} < y_{ij},$$

and exists y_{uv} such that

$$y_{ij+1} > y_{uv}, \quad y_{ab+1} > y_{uv} \quad \text{and} \quad y_{uv} > y_{ij}.$$

Then

$$\begin{array}{c} y_{uv-1} < y_{ab} < y_{ij} < y_{uv} < y_{ab+1}, \\ y_{uv-1} < y_{ij} < y_{uv} \quad \text{and} \quad y_{ab} < y_{ij} < y_{ab+1}. \end{array}$$

Collectively we know that the median of total elements exists within the limits of one fragment which contains the quantile median element, y_{ij}.
Therefore exceeding error rate is less than $\pm \frac{2f}{n} \times 100\%$. ◇

Experimental and computational analysis show that the rank of quantile median is a very close to the rank of total median. Also we have shown an almost even partitioning method using this quantile median element and quantile selection method.

3 Parallel Join on Hyper-cube Structure

3.1 Hyper-cube Structure

An n-dimensional hypercube, Q_n or n-cube, is a multiprocessor system whose network topology is based on an n-cube. It has 2^n nodes and each node is connected with n adjacent nodes and maximum distance among nodes is n. In our model, each node consists of a processor, main memory and secondary storage. And each node has n channels to send message, n channels to receive.

Definition 3.1 The k-cube, Q_k, is contained two $(k-1)$-cubes, $Q_{\{0\}k-1}$ and $Q_{\{1\}k-1}$.

Subcube $Q_{\{0\}k-1}$ and $Q_{\{1\}k-1}$ are same architecture with excepting MSB, and $(k-1)$-cube, $Q_{\{0\}k-1}$, are divided into two $(k-2)$-cubes, $Q_{\{00\}k-2}$ and $Q_{\{01\}k-2}$, and also $(k-1)$-cube, $Q_{\{1\}k-1}$, are divided into two $(k-2)$-cubes, $Q_{\{10\}k-2}$ and $Q_{\{11\}k-2}$ as shown by Figure 3.1.

By above subcubes, data communication paths are determined statically. For example, in 4-cube, if we transfer data from node

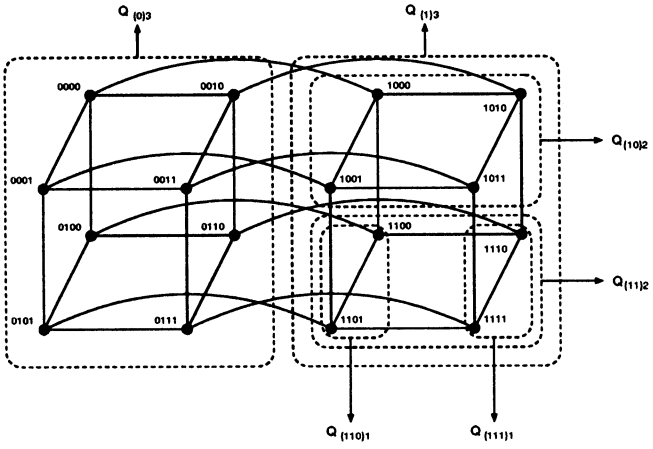

Figure 3.1. 4-cube architecture and relative cubes and nodes.

0010 to 1111, communication path is determined by MSB-free *relative node* order. We know that $Q_{\{0\}3}$ and $Q_{\{1\}3}$ are *relative subcube*. Therefore *relative node* of node 0010 is node 1010. Secondly we consider that the node 1010 is the part of $Q_{\{1\}3}$ and contained two subcubes, $Q_{\{10\}2}$ and $Q_{\{11\}2}$. Therefore *relative node* of node 1010 is node 1110. Finally, we know that the node 1110 and 1111 are adjacent. Communication path of the node 0010 to the node 1111 is 0010 → 1010 → 1110 → 1111. Conversely communication path of the node 1111 to the node 0010 are determined by the same method, 1111 → 0111 → 0011 → 0010.

3.2 Quantile Select Based Partition and Parallel Join

We know that the quantile median element partitions the large data elements into two parts almost evenly. We now apply this partitioning method to multiprocessor architecture in order to partition data to each node evenly. In k-cube of 2^k nodes, we will have $(2^k - 1)$ partitioning elements.

We extract quantile elements from each nodes in parallel. In each node we sort a segment which is as small as possible to fit in main memory. We then extract quantile elements as an ordered set of approximately $\sqrt{s}$ elements from each segments. This process continues until total elements are read. The number of quantile elements in each node is

$$\lfloor N_i/s \rfloor \times \sqrt{s} + (N_i - \lfloor Ni/s \rfloor)/\sqrt{s}.$$

Total number of quantile elements in k-cube are

$$\sum_{i=0}^{2^k-1} \{\lfloor N_i/s \rfloor \times \sqrt{s} + (N_i - \lfloor N_i/s \rfloor)/\sqrt{s}\}.$$

In this set Y of total quantile elements, we extract the number of $(2^k - 1)$ partitioning elements to divide Q_k (k-dimension cube) using the multi-selection algorithm.

Now we examine how the two subcubes are relatively partitioned by any partitioning elements. We know that all node addresses of $Q_{\{x_1x_2\cdots x_i0\}k-i-1}$ are less than all node addresses of $Q_{\{x_1x_2\cdots x_i1\}k-i-1}$. Therefore all elements of subcube Q_{k-i} are partitioned into the two subcube Q_{k-i-1} using median elements. Each node is divided by exchanging the data elements to relative partitioning node by the median elements. To evenly partition 2^k nodes in k-cube Q_k, the number of partitioning elements is $2^k - 1$. For example, 3-cube contains eight nodes and is partitioned by seven partitioning elements.

Lemma 2.2 *Parallel partitioning element* of *relative nodes* between subcube $Q_{\{x_1x_2\cdots x_i0\}k-i-1}$ and $Q_{\{x_1x_2\cdots x_i1\}k-i-1}$, where x_i is 0 or 1 for all $1 \le i < k$ and x_i is null for $i = 0$, is $(\sum_{p=1}^{i+1} 2^{k-p})$-th partitioning element for all 1 p-th bit in $\{x_1x_2\cdots x_i1\}$ of $(i+1)$ bits.

Algorithm *QUANT_SELECT_PART_JOIN(k, R_k, S_k)*

Input : k-cube, Q_k, and relation R_k and S_k.

Output : the result of $R \bowtie S$.

Step 1 : Extract quantile elements in parallel: read the smaller relation in each node, and select quantile elements using a well-known selection algorithm, and transmit these obtained quantile elements to NODE_0.

Step 2 : Select partitioning elements: in NODE_0, select $2^k - 1$ partitioning elements using all the received quantile elements, and broadcast these partitioning elements to all other nodes.

Step 3 : Partition the relations R and S in parallel: read **R** and **S** and partition these relations using partitioning elements, and broadcast the ranged data elements to each nodes.

Step 4 : Join the relation R and S in parallel.

---

4 Performance Measurement

In this section, we analyze our method to evaluate the processing time. We assume that the relations **R** and **S** are sufficiently larger than total memory in the system, initially their tuples are evenly distributed among the nodes, and **R** is the smaller relation.

4.1 Notations

We use the following notations for analyzing this algorithm.

$\|\|R\|\|$	size of the relation **R** (tuples)
P	size of a page (bytes)
n	dimension of cube
N	total number of nodes in the system, 2^n
PR	size of a node address (bytes)
TS_R	size of a tuple in **R** (bytes)
TP_R	number of tuples in a page for **R** (tuples)
PS	maximum packet size (bytes)
$comm$	communication rate (bits/sec)
$comp$	comparing time (sec)
$move$	time for moving one tuple in main memory (sec)
IO	time for reading/writing a page from/to the disk (sec)
$M+1$	size of memory in each processor (pages)

4.2 Modeling and Simulation

For modeling and simulation, we employ shared-nothing hypercube architecture. We will compare the performance of our method with PHJA in several cases. Parameters used are as follow: P is 4000 byte, $comm$ = 4 Mb/s, $comp$ = 3 μs, $move$ = 20 μs, and IO = 20ms, and size of R and S are 32000 words and 64000 words.

In the quantile elements selection phase, we read the smaller relation **R** and select the quantile elements, then transfer these elements to NODE_0.

$$T_{SEL} = T_{Rread} + T_{Qsel} + T_{Qcomm}$$

$$T_{Rread} = \lceil \frac{\|R\|}{N \times TP_R} \rceil \times IO$$

$$T_{Qsel} = \frac{\|R\|}{N} \times (comp + move) \times 2$$

$$T_{Qcomm} = \lceil \frac{\|R\|}{N \times TP_R} \rceil \times \lfloor \sqrt{P-1} \rfloor \times \frac{8}{comm}$$

In the partitioning elements selection phase, we obtain $N-1$ partitioning elements from transferred total quantile elements in $NODE_0$, and broadcast these elements to all other nodes. In each node, all data elements must be sorted by joining attribute.

$$T_{PESEL} = \lceil \frac{\|R\|}{TP_R} \rceil \times \lfloor \sqrt{P-1} \rfloor \times (comp + move) \times 2 + (N-1) \times TS_R \times \frac{8}{comm}$$

In the relation partitioning phase, we distribute the partitioned data using partitioning elements into all nodes. Therefore we read **R** and **S**, partition and distribute, and write the transferred data.

$$T_{PART} = \lceil (\frac{\|R\|}{N \times TP_R} + \frac{\|S\|}{N \times TP_S}) \rceil \times (2 \times IO + comp + \frac{8}{comm} \times \frac{N-1}{N})$$

In the joining phase, we join the relations **R** and **S** using the sort-merge join method because all data elements were sorted in partitioning phase.

$$T_{JOIN} = \lceil (\frac{\|R\|}{N \times TP_R} + \frac{\|S\|}{N \times TP_S}) \rceil \times (IO + comp + move)$$

Therefore total cost is

$$T_{TOTAL} = T_{SEL} + T_{PESEL} + T_{PART} + T_{JOIN}.$$

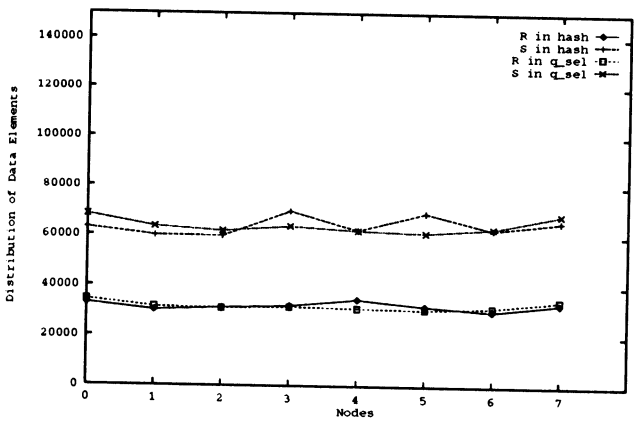

Figure 4.1.a. Data distribution when $\theta = 1$ in Zipf distribution (Even distribution)

In Figure 4.1.a and 4.1.b, we show data distribution and processing time for PHJA and QSJA when $\theta = 1$ in Zipf distribution(Choi et al. 1993). Note that PHJA has better performance because of approximately even partition. In this case, joining time and partitioning time may be equal, but QSJA has more processing time for quantile selecting of partitioning elements. Therefore QSJA is slower than PHJA.

In Figure 4.2.a and 4.2.b, other data sets are used. Data distribution of PHJA becomes small skewed when $\theta = 0.2$ in Zipf distribution. But QSJA has approximately even distribution of data elements. In this case, QSJA has better performance than PHJA. Figure 4.2.c is flow of processing. In this Figure, we know that joining time is much larger than partitioning time in case of unskewness.

In last case, we use a data set of duplicated tuples in deep skew such as $\theta = 0$ in Zipf distribution. In Figure 4.3.a and 4.3.b, data distribution of PHJA is skewed to some nodes. But QSJA is approximately even because it distributes to each node preserving range by partitioning elements. Therefore we know that QSJA has better performance in case of skewness by duplicated tuples.

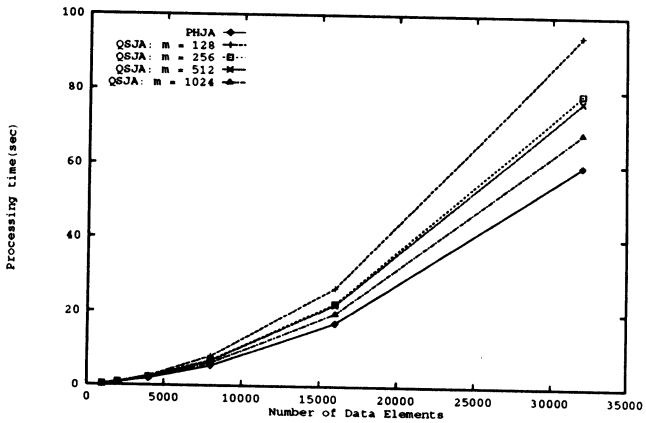

Figure 4.1.b. Processing time for even distribution

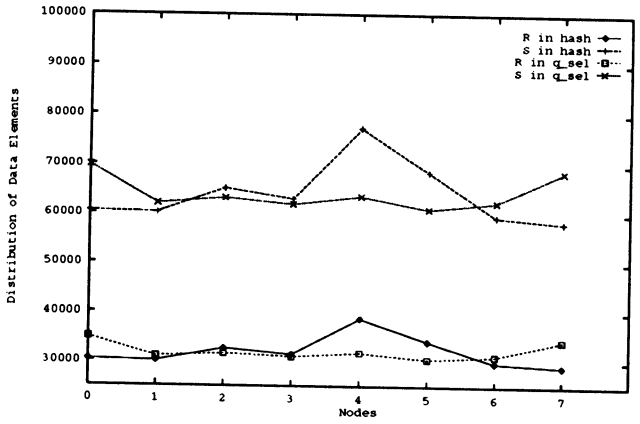

Figure 4.2.a. Data distribution when $\theta = 0.2$ in Zipf Distribution (mild skewed distribution)

5 Conclusions and Future Research

We proposed a new partitioning method which guarantees the approximately even partitioning in a hypercube computer system. Based on this partitioning method we proposed a parallel join algorithm. We showed that the quantile median was approximately equal to the median of total elements, therefore the quantile median divided the total set into two subsets of almost same size. Error rate of the quantile median does not exceed 0.2 percent of median of total elements. The quantile median was applied to multi-partitioning elements in a hypercube computer system to partition the two relations **R** and **S** evenly across each node.

Performance of parallel join based our partitioning method was measured and compared with the parallel hash partitioned join. Ours showed better performance than PHJA.

References

Blum, M.;R.W. Floyd; V. Pratt; R.L. Rivest; and R.E. Tarjan, 1973, "Time bounds for selection," *J. Comput. System Sci.*, Vol. 7, pp. 448–461.

Chang, H.; J.S. Park; and M. Kim, 1991, "Select-partitioned join: an improved partition-based join algorithm," *Inform. Syst.*, Vol. 16, no. 2, pp. 199–209.

Choi, H.K.; U.K. Park; and T.G. Kim, 1993, "Simulation based evaluation of parallel join algorithms," In *Proc. Summer Simula-*

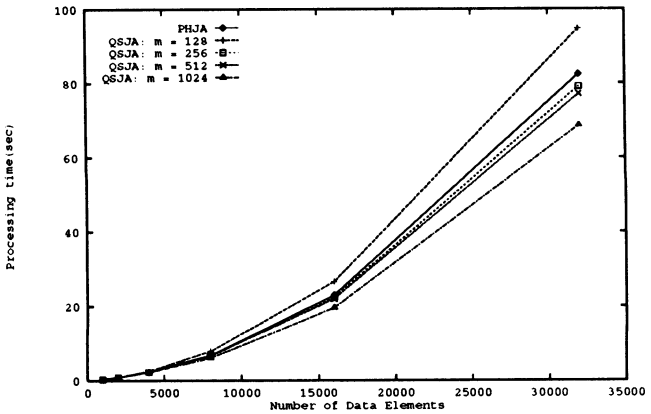

Figure 4.2.b. Processing time for mild skewed distribution

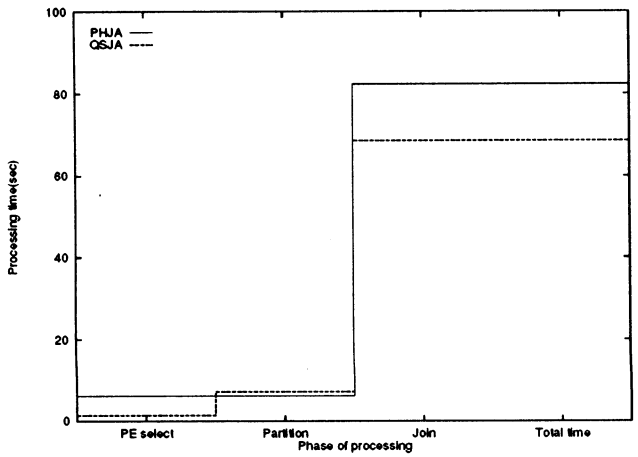

Figure 4.2.c. Phase of processing for mild skewed distribution

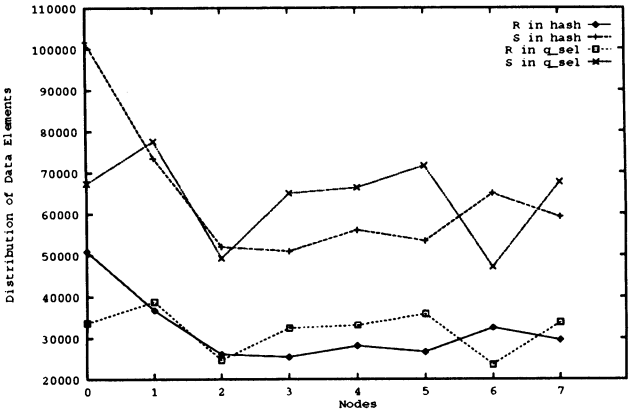

Figure 4.3.a. Data distribution when $\theta = 0$ in Zipf distribution (deep skewed distribution)

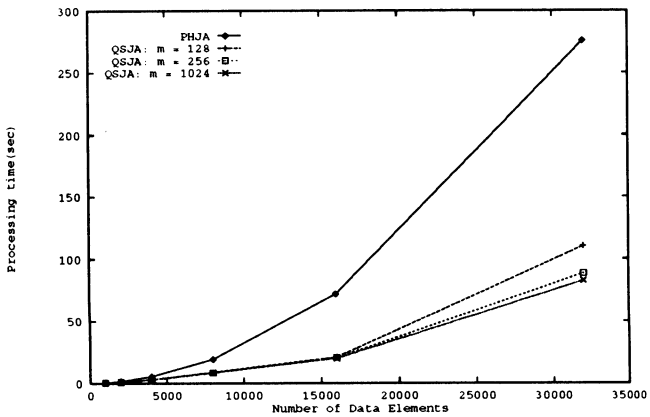

Figure 4.3.b. Processing time by deep skewed distribution

tion Conf..

Dobkin, D. and J.I. Munro, 1981, "Optimal time minimal space selection algorithm," *J. ACM*, Vol. 28, no. 3, pp. 454–461.

Floyd, R.W. and R.L. Rivest, 1975, "Expected time bounds for selection," *Comm. ACM*, Vol. 18, no. 3, pp.165–172.

Frieder, O., 1990, "Multiprocessor algorithms for relational-database operators on hypercube systems," *IEEE Computer*, Vol. 23, No. 11, pp. 13–28.

Kim, H.S.; J.S. Park; and M. Kim, 1992, "An algorithm for the K-selection problem using special-purpose sorters," *IEICE Trans. Inf. & Syst.*, Vol. E75-D, no. 5, pp. 704–708.

Motoki, T., 1982, "A note on upper bounds for the selection problem," *Inform. Process. Lett.*, Vol. 15, no. 5, pp. 214–219.

Munro, J.I. and M.S. Paterson, 1980, "Selection and sorting with limited storage," *Theo. Comput. Sci.*, Vol 12, pp. 315–323.

Omiecinski, E.R. and E.T. Lin, 1989, "Hash-based and index-based join algorithms for cube and ring connected multicomputers," *IEEE tran. on Knowledge and Data Engineering*, Vol. 1, No. 3, pp. 329–343.

Omiecinski, E.R. and E.T. Lin, 1992, "The adaptive-hash join algorithm for a hypercube multicomputer," *IEEE tran. on Parallel and Distributed Systems*, Vol. 3, No. 3, pp. 334–349.

Park, J.S. and M. Kim, 1989, "A selection algorithm with a practical upper bound on expected number of comparisons," *Software - Practice and Experience*, Vol. 19, no. 11, pp. 1105–1110.

Schönage, A.; M. Paterson; and N. Pippenger, 1976, "Finding the median," *J. Comput. System Sci.*, Vol. 13, pp. 184–199.

Yu, C.T.; M.K. Siu; and K. Lam, 1978, "On a partitioning problems," *ACM TODS*, Vol. 3, no. 3, pp. 299–309.