VICTORIA UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF COMPUTER AND MATHEMATICAL SCIENCES

Parallel Processing of Aggregate Functions
in the Presence of Data Skew

Kevin H. Liu, Yi Jiang and Clement H. C. Leung

(71 COMP 25)

April 1996

(AMS : 68P15)

TECHNICAL REPORT

VICTORIA UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF COMPUTER AND MATHEMATICAL SCIENCES
P O BOX 14428
MCMC
MELBOURNE, VICTORIA 8001
AUSTRALIA

TELEPHONE (03) 9688 4492
FACSIMILE (03) 9688 4050

Footscray Campus

PARALLEL PROCESSING OF AGGREGATE FUNCTIONS IN THE PRESENCE OF DATA SKEW

Kevin H. Liu, Yi Jiang and Clement H. C. Leung

Department of Computer and Mathematical Sciences

Victoria University of Technology, Ballarat Road, Footscray,

PO Box 14428 MMC, Melbourne, Victoria 3000, AUSTRALIA

Email: {kevin, jiang, clement}@matilda.vut.edu.au

Fax: +613-688 4050

ABSTRACT

Parallelising aggregate functions often involves queries of more than one relation, so that performance improvement can be achieved in both the intra-operation and inter-operation parallelism levels. Join before aggregation is the conventional way for processing aggregate functions in uni-processor systems and parallel processing of aggregate functions has received little attention. In this paper, the effects of the sequence of aggregation and join operations are identified in a parallel processing environment. Three parallel methods of processing aggregate functions for general queries are presented which mainly differ in their selection of partitioning attribute, i.e *JPM* partitions on join attribute, *APM* fragments on group-by attribute, and *HPM* adaptively partitions on both join and group-by attribute with a logical hybrid architecture. Their cost models are provided which incorporate the effect of data skew. The performance of these methods are investigated and the results are reported.

1. Introduction

With the increasing complexity of database applications and the advancing inexpensive multiprocessor technology, parallel query processing has emerged as an efficient way to improve database performance [DeWi92, Moha94]. The parallelisation of query processing can be conducted at intra-operation level, inter-operation level or combination of both. Intra-operation parallelism attempts to, one at a time, distribute the load of each single relational operation involved in the query to all processors and performs the operation in parallel; while the inter-operation parallelism aims at parallelising the operations of the query and allocating them to different processors for execution [DeWi92].

Parallel processing of major relational operations (e.g. selection, projection and join) has been studied extensively in recent years [Bell89, Ghan94, Seve90, Kell91, Wolf93a, Wolf93b]. By contrast, parallel aggregation receives much less attention although it is critical to the performance of database applications such as decision support, quality control, and statistical databases. Aggregation may be classified into *scalar aggregate* and *aggregate function* [Bitt83, Grae93]. The former refers to the simple aggregation that produces a single value from one relation such as counting the number of tuples or summing the quantities of a given attribute; while the latter refers to those that cluster the tuples of the relation(s) into groups and produce one value for each group. The queries with aggregate functions often involve more than one relation, and thus require join operations. The issues on parallel processing scalar aggregate has been studied in [Shat94] for locally distributed databases. A selectivity estimation is required for their adaptive methods, and the cost components of the analytical models are provided.

This paper concentrates on the issues of aggregate functions and investigates efficient parallel processing methods for queries involving aggregations and joins[1]. Three methods, namely, *join-partition method (JPM), aggregation-partition method (APM)* and *hybrid-partition method (HPM),* are presented. *JPM* and *APM* mainly differ in the selection of partitioning attribute for distributing workload over the processors and *HPM* is an adaptive method based on *APM* and *JPM* with a logical hybrid architecture. Furthermore, all methods take into account the problem of data skew since the skewed load distribution may affect the query execution time significantly. The performance of the parallel aggregation methods have been compared under various queries and different environments with a simulation study, and the results are also presented. In the next section, we discuss the critical issues on parallelising aggregate functions namely, the selection of partitioning attribute, the sequence of aggregation and join operation, and the skewness. The parallel processing methods and their cost models are introduced in Section 3 followed by a sensitivity analysis in Section 4.

2. Parallelising Aggregate Functions

For simplicity of description and without loss of generality, we consider queries that involve only one aggregation function and a single join. The example queries given below arise from a Suppliers-Parts-Projects database. The first query clusters the part shipment by their city locations and selects the cities with average quantity of shipment

[1] Hereafter, aggregation means simple aggregation operation (e.g. AVG and SUM on an attribute in one relation) while aggregate function (or aggregation query) consists of aggregation and join operation.

between 500 and 1000. The second query retrieves the project number, name and the total quantity of shipment for each project.

SUPPLIER (<u>s#</u>, **sname, status, city)**
PARTS (<u>p#</u>, **pname, colour, weight, price, city)**
PROJECT (<u>j#</u>, **jname, city, budget)**
SHIPMENT (<u>s#, p#, j#</u>, **qty)**

```
Query 1: SELECT  parts.city, AVG(qty)
           FROM  parts, shipment
           WHERE  parts.p#=shipment.p#
           GROUP BY  parts.city
           HAVING  AVG(qty)>500 AND AVG(qty)<1000;

Query 2: SELECT  project.j#, project.jname, SUM(qty)
           FROM  project, shipment
           WHERE  project.j#=shipment.j#
           GROUP BY  project.j#, project.name
           HAVING  SUM(qty)>1000;
```

For parallel query processing, we assume a parallel architecture that consists of a host and a set of working processors as shown in Figure 1. The host accepts queries from users and distributes each query with required base relations to the processors for execution. The processors perform the query in parallel with possibly intermediate data transmission between each other through the network, and finally send the result of the query to the host.

In the present architecture, an aggregation query is carried out in three phases:

- *Data partitioning*, the operand relations of the query are partitioned and the fragments are distributed to each processor;
- *Parallel processing*, the query is executed in parallel by all processors and the intermediate results are produced;

- *Data consolidation*, the final result of the query is obtained by consolidating the intermediate results from the processors.

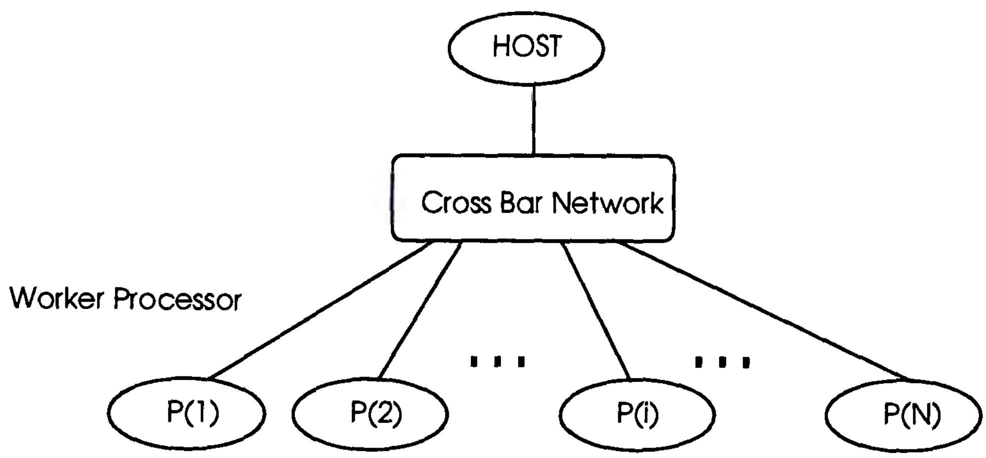


Figure 1

2.1 Selection of Partition Attribute

Choosing proper partition attribute is a key issue in the above procedure. Although in general any attributes of the operand relations may be chosen, two particular attributes, i.e. join attribute and group-by attribute, are usually considered (e.g. *p#* and *city* in the first example query). If the join attribute is chosen, both relations can be partitioned into N fragments using either range partitioning or hash partitioning strategy, where N is the number of processors. The cost for parallel join operation can therefore be reduced by a factor of N^2 as compared with a single processor system. However, after join and local aggregation at each processor, a global aggregation is required at the data consolidation phase since the local aggregation is performed on a subset of the group-by attribute. In contrast, if the group-by attribute is used for data partitioning, the relation with the group-by attribute can be partitioned into N fragments while the other relation needs to be broadcast to all processors in order to perform the join, leading to a reduction in join cost by a factor of merely N. Although, in the second method, the join cost is not reduced as much as in the first method, no global aggregation is required after local join and aggregation at each processor because the tuples with identical values of the group-by attribute have been allocated to the same processor. Assuming that there are indexes on the join attribute and 4 processors, and the execution time is given in terms of the number of tuples processed in the absence of data skew, Figures 2 and 3 illustrate the execution time of two types of partitioning strategies on the first example query.

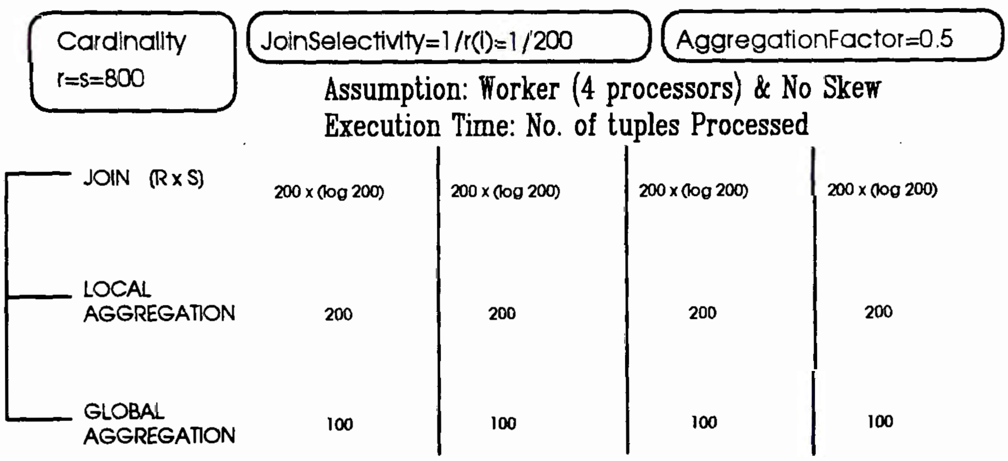


Figure 2. Join-Partition method

2.2 Sequence of Aggregation and Join Operation

When the join attribute and group-by attribute are the same as shown in the second example query (i.e. *j#*), the selection of partitioning attribute becomes obvious. Instead of performing join first, the aggregation would be carried out first followed by the join since the join is more expensive in cost and it would be beneficial to reduce the join relation sizes by applying aggregation first. Generally, aggregation should always precede join whenever it is possible with the exception that the size reduction gained from aggregation is marginal or the join selectivity factor is extremely small. Figure 4 shows that by applying aggregation first, the execution time of the second example query is much lower than that of join operation first as shown in Figures 2 and 3. However, aggregation before join may not always be possible, and the semantic issues on aggregation and join and the conditions under which the aggregation would be performed before join can be found in [Kim82, Daya87, Bult87, Yan94]. In the following sections, we assume the more general case where aggregation can not be executed before join, since earlier aggregation reduces execution time and thus we process aggregation before join if it is possible.

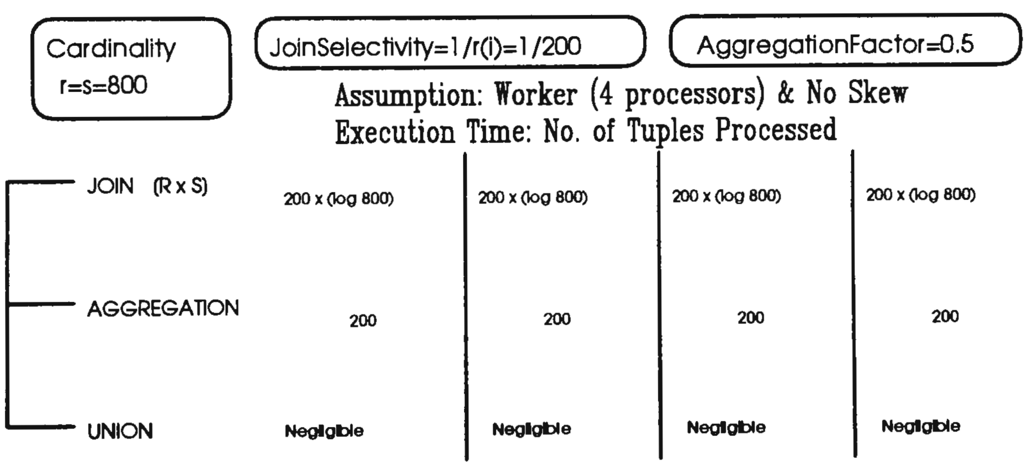


Figure 3. Aggregation-Partition method

2.3 Skewness

Another issue in parallel processing of aggregation queries is the problem of data skew, and in this context, data skew consists of *data partitioning skew* and *data processing skew*. The data partitioning skew refers to uneven distributing tuples of the operand relations over the processors and thus results in different processing loads. This is mainly caused by skewed value distribution in the partitioning attribute as well as improper use of the partitioning method. The data processing skew, on the other hand, refers to uneven sizes of the intermediate results generated by the processors. This is caused by the different selectivity factors of the join/aggregation of the local fragments. Since the intermediate results of the join operation are processed for aggregation, the data processing skew may lead to different loads of the processors even when there is no data partitioning skew. As skewness may reduce the improvement of query response time gained from parallel processing substantially, it must be considered in parallelising aggregation queries.

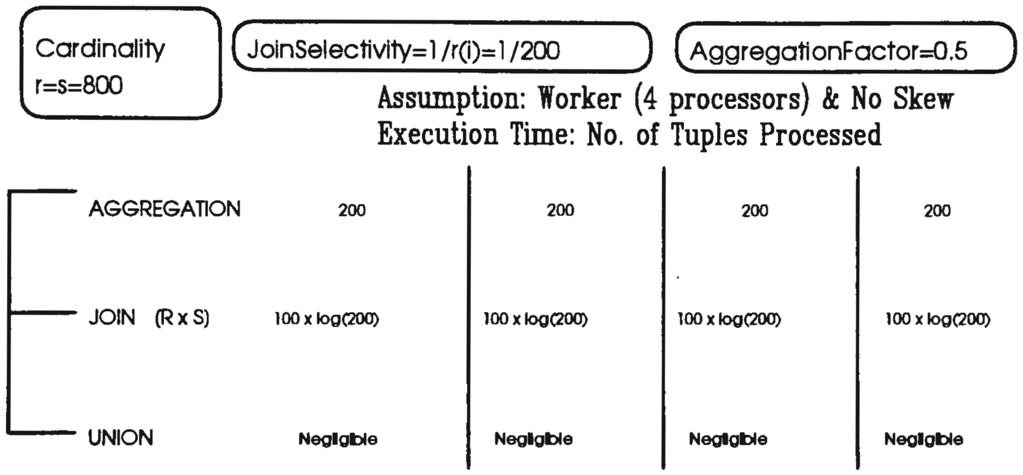


Figure 4. Aggregation Before Join

Parameters	Meaning
N	the total number of processors
m	the number of processor clusters
n	the number of processor in each cluster ($N = m \times n$)
r, s	the number of tuples in base relations R and S
r_i, s_i	the number of tuples of fragments of relations R and S at processor i
$Sel(i)$	join selectivity factor for fragment i
$Agg(i)$	aggregation factor for fragment i
θ	reduction factor after performing *Having* clause
α	data partitioning skew factor
β	data processing skew factor
T_{comm}	the average data transmission time for each message
T_{join}	the average join time for each tuple
T_{agg}	the average aggregation time for each tuple
z	message size in terms of the number of tuples

Table 1. Notations

3. Parallel Processing Methods

We present in this section three parallel processing methods for queries that involve joins and aggregation functions. The notations used in the description of the methods and in the subsequent performance evaluation are given in Table 1.

3.1 Join Partitioning Method (*JPM*)

The *JPM* can be stated as follows. First, the relations *R* and *S* are partitioned into *N* fragments in terms of join attribute, i.e. the tuples with the same join attribute values in the two relations fall into a pair of fragments. Each pair of the fragments will be sent to one processor for execution. Upon receipt of the fragments, in parallel, the processors perform the join operation and then the local aggregation operation on the fragments allocated. After that, a global aggregation operation is carried out by re-distributing the local aggregation results across the processors such that the result tuples with identical values of group-by attribute are allocated to the same processors. Then, each processor performs a *N*-way merging with the local aggregation results, followed by doing a restriction operation for the *Having* clause if exists at local processors. Finally, the host simply consolidates the partial results from the processors by a union operation, and produces the query result.

Given the notation early, the execution time for the *JPM* method can be expressed as follows

$$JPM = T_{comm} \times \left(\max(r_i + s_i)\right) + T_{join} \times \left(\max(r_i \log s_i)\right) + T_{agg} \times \left(\max(r_i \times s_i \times Sel(i))\right)$$

$$+T_{comm} \times \left(\max(r_i \times s_i \times Sel(i) \times Agg(i))\right)$$

$$+T_{agg} \times \left(\max\left(r_i \times s_i \times Sel(i) \times Agg(i)\right)\right) \times (1+1). \tag{1}$$

The maximum execution time for each of the components in the above equation varies with the degree of skewness, and could be far from average execution time. Therefore, we introduce two skew factors α and β to the above cost equation, and α describes the data partitioning skew while β represents the data processing skew. Assume that α follows the Zipf distribution where the ith common word in natural language text occurs with a frequency proportional to i [Knut73, Wolf93b], i.e.

$$p_i = \frac{1}{i \times \sum_{j=1}^{N} \frac{1}{j}} = \frac{1}{i \times H_N}$$

where H_N is the Harmonic number and could be approximated to $(\gamma + \ln N)$ [Leun94]. Notice that the first element p_1 always gives the highest probability and the last element p_N gives the lowest. Considering both operand relations R and S use the same number of processors and follow the Zipf distribution, the data partitioning skew factor α thus can be represented as

$$\alpha = \alpha_r = \alpha_s = \frac{1}{H_N} = \frac{1}{\gamma + \ln N},$$

where $\gamma = 0.57721$ known as the Euler Constant and N is the number of processors.

The other skew factor β for data processing skew is affected by the data partitioning skew factors in both operand relations since the join/aggregation results rely on the operand fragments. Therefore, the range of β falls in $[\alpha_r \times \alpha_s,\ 1]$. However, the actual value of β is difficult to estimate because the largest fragments from the two relations are usually not allocated to the same processor, resulting the β much less than the product of α_r and α_s. We assume in this paper $\beta = (\alpha_r \times \alpha_s + 1)/2 = (\alpha^2 + 1)/2$ and a detail discussion on skewness can be found in [Liu95, Leun94].

Applying the skew factors to the above cost equation, we also make the following assumptions and simplifications:

- $J_i = r_i \times s_i \times Sel(i) = J$ i.e. in the absence of the skewness,
- $Agg(i) = Agg$,
- $T_{join} = T_{agg} = T_{proc}$,
- data transmission is carried out by message passing with a size z.

The cost equation (1) can then be re-written below

$$JPM = T_{comm}\left\{\left[\alpha(r+s) + \frac{J \times Agg}{\beta}\right]/z\right\}$$

$$+T_{proc}\left[\alpha r \times \log(\alpha s) + \frac{J}{\beta} + \frac{(1+1) \times J \times Agg}{\beta}\right]$$

$$= T_{comm}\left[\left(\frac{r+s}{\gamma + \ln N} + \frac{2 \times (\gamma + \ln N)^2}{1 + (\gamma + \ln N)^2} \times J \times Agg\right)/z\right]$$

$$+T_{proc}\left(\frac{r}{\gamma + \ln N}\log\left(\frac{s}{\gamma + \ln N}\right) + \frac{2 \times (\gamma + \ln N)^2}{1 + (\gamma + \ln N)^2} \times J \times (1 + 2 \times Agg)\right).$$

3.2 Aggregation Partitioning Method (*APM*)

In the *APM* method, the relation with the group-by attribute, say *R*, is partitioned into *N* fragments in terms of the group-by attribute, i.e. the tuples with identical attribute values will be allocated to the same processor. The other relation S needs to be broadcast to all processors in order to perform the binary join. After data distribution, each processor first conducts the joining one fragment of R with the entire relation S, followed by the group-by operation and *having* restriction if exists on the join result. Since the relation R is partitioned on group-by attribute, the final aggregation result can be simply obtained by an union of the local aggregation results from the processors, i.e. the step of merging of the local results used in *JPM* method is not required. Consequently, the cost of the *APM* can be given by

$$APM = T_{comm} \times \left(\max\left(r_j + s\right)\right) + T_{join} \times \left(\max\left(r_j \log s\right)\right)$$
$$+T_{agg} \times \left(\max\left(r_j \times s \times Sel(j)\right) \times \left(1 + Agg(j)\right)\right)$$
$$+T_{comm} \times \left(\max\left(r_j \times s \times Sel(j) \times Agg(j) \times \theta\right)\right). \qquad (2)$$

The skew factors α and β can be added to the above equation in the same way for the *JPM* method. For the purpose of comparison of the two methods, we assume that $J_j = r_j \times s \times Sel(j) = J$ and $Agg(j) = \frac{1}{N} Agg$. The time of *APM* method can then be expressed as

$$APM = T_{comm}\left[\left(\alpha r + s + \frac{J \times Agg \times \theta}{\beta \times N}\right)/z\right] + T_{proc}\left[(\alpha r) \times \log s + \frac{J}{\beta} \times \left(1 + \frac{Agg}{N}\right)\right]$$
$$= T_{comm}\left[\left(\frac{r}{\gamma + \ln N} + s + \frac{2(\gamma + \ln N)^2}{1 + (\gamma + \ln N)^2} \times \frac{J \times Agg \times \theta}{N}\right)/z\right]$$
$$+T_{proc}\left(\frac{r}{\gamma + \ln N}\log s + \frac{2(\gamma + \ln N)^2}{1 + (\gamma + \ln N)^2} \times J \times \left(1 + \frac{Agg}{N}\right)\right).$$

3.3 Hybrid Partitioning Method (*HPM*)

The *HPM* method is a combination of the *JPM* and *APM* methods. In the *HPM*, the processors are divided into *m* clusters each of which has *N/m* processors as shown in Figure 5. Based on the proposed logical architecture, the data partitioning phase is carried out in two steps. First, the relation with group-by attributes is partitioned into processor clusters in the same way of the *APM*, i.e. partitioning on the group-by attribute and the other relation is broadcast to the cluster. Second, within each cluster, the fragments of the first relation and the entire broadcast relation is further partitioned by the join attributes as the *JPM* does. Depending on the parameters such as the cardinality of the relations and the skew factors, a proper value of *m* will be chosen such that the minimum query execution time is achieved.

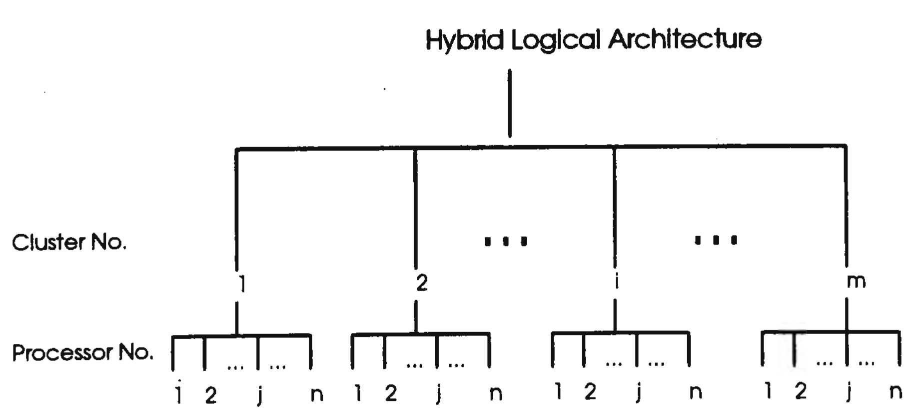


Figure 5. Logical Architecture for *HPM*

The detail of the *HPM* method is described below:

<u>Step 1</u> Partition the relation *R* on group-by attribute to *m* clusters, denoted by r_i.

Within each cluster, further partition the fragment r_i and the other relation *S* on join attribute to $n=N/m$ processors, denoted by r_{ij} and s_j. Therefore, the total data transmission time is given by

$$T_{comm} \times \left(\max\left(r_{ij} + s_j\right)\right)$$

where i is in the range of $[1,m]$ and j is in the range of $[1,n]$.

Step 2 Carry out join at each processor and the maximum processing time is expressed as

$$T_{join} \times \left(\max\left(r_{ij} \log s_j\right)\right)$$

Step 3 Perform local aggregation at each processor with the execution time

$$T_{agg} \times \left(\max\left(r_{ij} \times s_j \times Sel(j)\right)\right)$$

Step 4 Redistribute the local aggregation results to the processors within each cluster by partitioning the results on the group-by attribute. The transmission time is given as

$$T_{comm} \times \left(\max\left(r_{ij} \times s_j \times Sel(j) \times Agg(j)\right)\right)$$

Step 5 Merge the local aggregation results within each cluster and this requires the time

$$T_{agg} \times \left(\max\left(r_{ij} \times s_j \times Sel(j) \times Agg(j)\right)\right)$$

Step 6 Perform the *Having* predicate in each cluster with the processing time

$$T_{agg} \times \left(\max\left(r_{ij} \times s_j \times Sel(j) \times Agg(j)\right)\right)$$

Step 7 Transfer the results from the clusters to the host. The time for data consolidation in the host is small and thus only data transmission cost is counted, i.e.

$$T_{comm} \times \left(\max\left(r_{ij} \times s_j \times Sel(j) \times Agg(j) \times \theta\right)\right)$$

The total execution time of the *HPM* is the sum of the time of the above steps and can be given as

$$HPM = T_{comm} \times \left(\max\left(r_{ij} + s_j\right)\right) + T_{join} \times \left(\max\left(r_{ij} \log s_j\right)\right) + T_{agg} \times \left(\max\left(r_{ij} \times s_j \times Sel(j)\right)\right)$$

$$+T_{comm} \times \left(\max\left(r_{ij} \times s_j \times Sel(j) \times Agg(j)\right)\right)$$

$$+T_{agg} \times \left(\max\left(r_{ij} \times s_j \times Sel(j) \times Agg(j)\right)\right) \times 2$$

$$+T_{comm} \times \left(\max\left(r_{ij} \times s_j \times Sel(j) \times Agg(j) \times \theta\right)\right). \qquad (3)$$

By applying the same simplification assumptions in the previous methods and $Agg(j) = \frac{1}{m} Agg$, (3) is re-written as

$$HPM = T_{comm}\left[\alpha_n \times \alpha_m \times r + \alpha_n \times s + \frac{(1+\theta)}{\beta_n} \times J \times \frac{Agg}{m}\right] / z$$

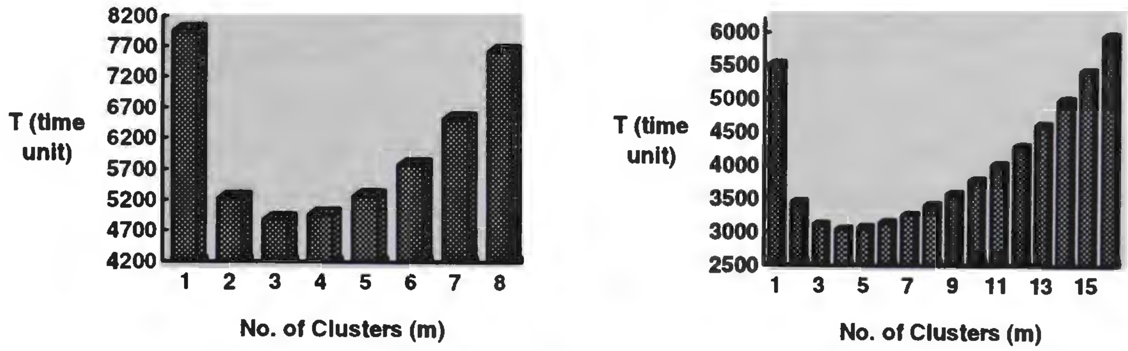
$$+T_{proc}\left[\alpha_n \times \alpha_m \times r \times \log(\alpha_n \times s) + \frac{J}{\beta_n} \times \left(1 + 2 \times \frac{Agg}{m}\right)\right],$$

where $J = r_{ij} \times s_j \times Sel(j)$, $\alpha_n = \frac{1}{\gamma + \ln n}$, $\alpha_m = \frac{1}{\gamma + \ln m}$, and $\beta_n = \frac{1 + (\gamma + \ln n)^2}{2(\gamma + \ln n)^2}$.

It can be seen from the above execution time equation that the number of clusters m has strong influence on the performance of *HPM*. Figures 6(a) and 6(b) show the changes of query execution time when increasing the number of clusters in the *HPM*. It appears that a value of $m \approx \left\lceil \sqrt{N} \right\rceil$ approximately gives the optimal cost of the query[2] although the

[2]The approximation can also show the robutness of the adaptive method (*HPM*).

precise value of m may be worked out by finding the minimum value of the differentiation of the equation (3).



(a). 8 processors (b). 16 processors

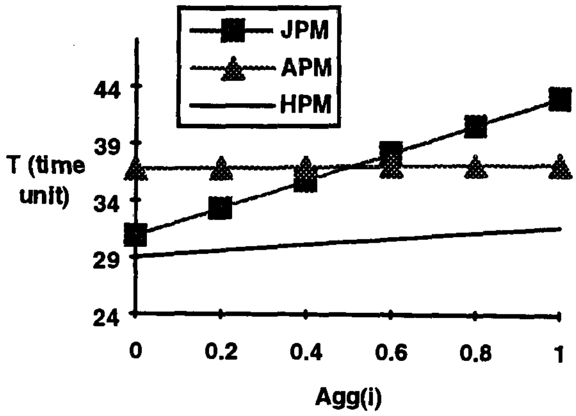
Figure 6. Cost vs No. of Clusters in *HPM*

Parameters	Values
N	16
m	$\sqrt{16}=4$
r	1000 tuples
s	1000 tuples
$Sel(i)$	$5(N/r) = 0.08$
$Agg(i)$	0.5
θ	0.5
α	0.2985
β	0.5446
α_n	0.5093
α_m	0.5093
β_n	0.6297
T_{comm}	0.1 standard time unit per message
T_{proc}	0.01 standard time unit per tuple
z	100 tuples per message

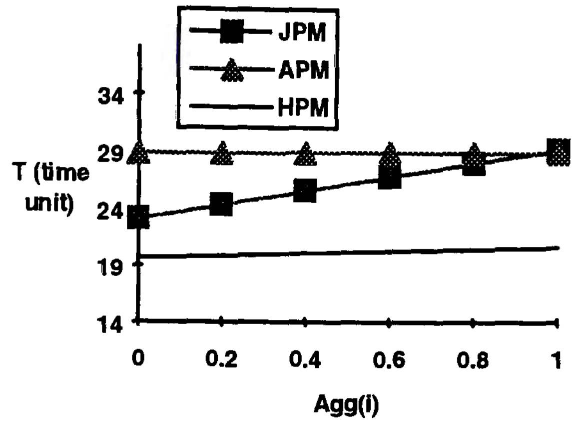
Table 2. Default Parameters Values Listing

4. Sensitivity Analysis

The performance of the three parallel processing methods presented may vary with a number of parameters listed in Table 1. In this section, we analysis among them the effect of the aggregation factor, the join selectivity factor, the degree of skewness (α and β), the relation cardinality, and the ratio of T_{comm} / T_{proc}. The default parameter values are given in Table 2.



(a). 16 processors

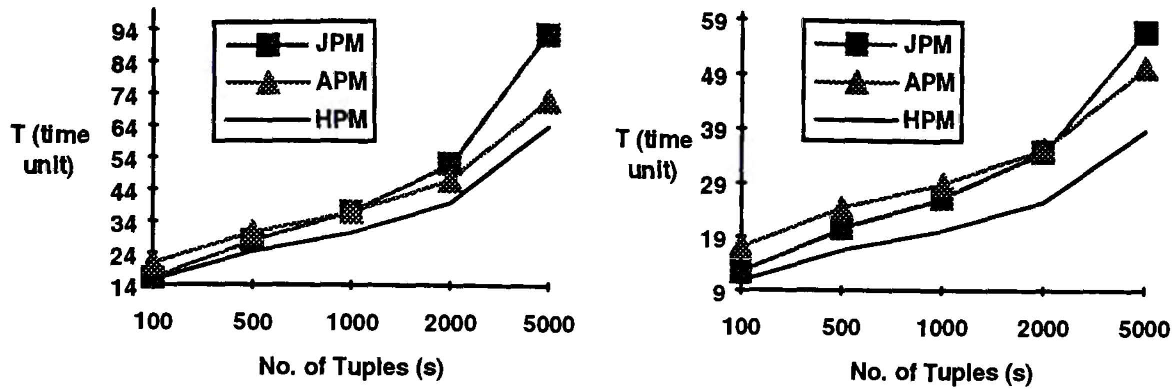


(b). 32 processors

Figure 7. Varying Aggregation Factor

4.1 Varying the Aggregation Factor

The aggregation factor β is defined by the ratio of result size after aggregation to the size of the base relation and its impact on three methods is shown in Figure 7. Not surprisingly, *APM* is insensitive to aggregation factor. The reasons include little data to transmit after joining and processing the *Having* predicate comparing with data partitioning at beginning and broadcasting relation S, and little data to select with the group-by condition (*Having*) as the aggregation factor is reduced by a factor of the number of processors. Generally, the larger the aggregation factor, the more the running time is needed as shown in Figure 7(a) with 16 processors. Moreover, increasing the number of processors will reduce the running time despite data partitioning and processing skew, and the performance of *JPM* is better than that of *APM* except the aggregation factor is very large as plotted in Figure 7(b) with 32 processors. In both Figure 7(a) and 7(b), *HPM* offers the best performance.



(a). 16 processors (b). 32 processors

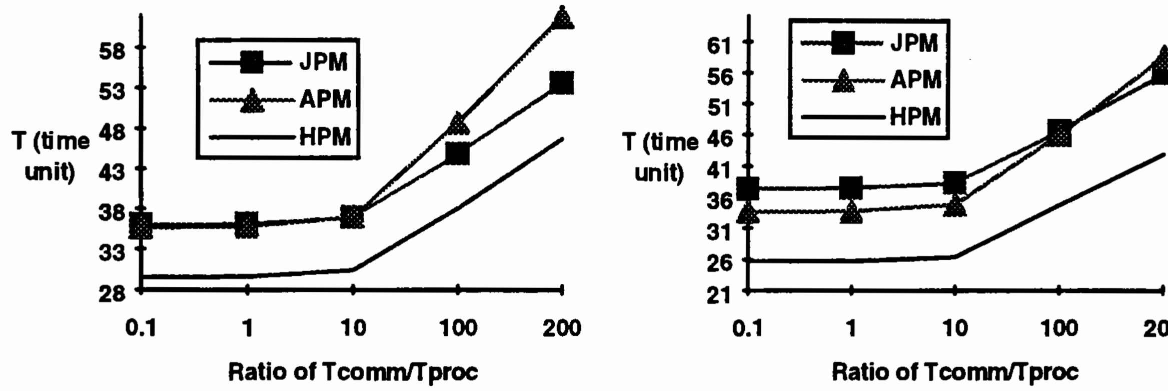
Figure 8. Varying the cardinality of relation S

4.2 Varying the Relation Cardinality

The cardinality of the operand relations are assumed the same elsewhere in the sensitivity analysis and their influences on performance are investigated in this subsection. Figure 8 shows the query execution time when we fix the cardinality of one relation and increase the cardinality of another relation. *JPM* appears to be better than *APM* only when the varied relation size is small while the *HPM* again outperforms *APM* and *JPM* in all situations. Comparing 8(a) to 8(b), increasing processors will raise the cross-over point of *JPM* and *APM*.

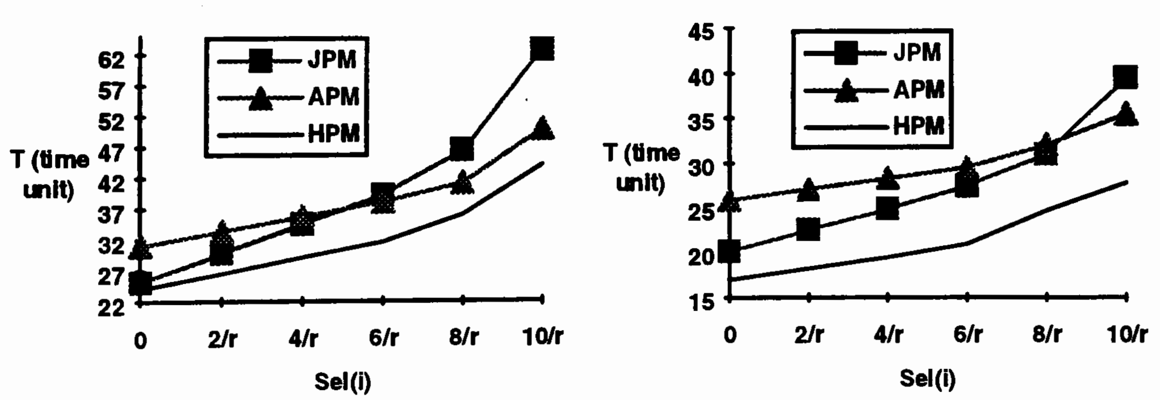
4.3 Varying the Ratio of T_{comm} / T_{proc}

The ratio of T_{comm} / T_{proc} reflects the characteristic of network used in the parallel architecture. Primarily, data communication is not a critical issue any more in parallel database systems comparing with distributed database systems [Alma94]. As we increase the ratio shown in Figure 9, system performance decreases since we treat T_{proc} as a standard time unit and magnify the communication cost, i.e. higher ratio means more expensive communication. Being a parallel database system, the ratio tends to stay small and *APM* is the most sensitive to the communication cost. Nevertheless, *HPM* will always perform better than either *JPM* or *APM* with improvement on fragmenting relations within each cluster.



(a). 16 processors (b). 32 processors

Figure 9. Varying the ratio of T_{comm} / T_{proc}



(a). 16 processors (b). 32 processors

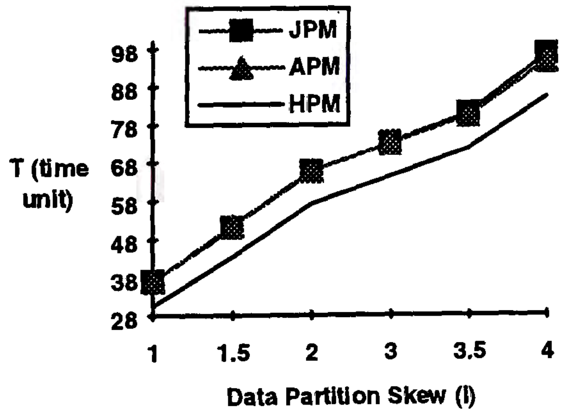
Figure 10. Varying the selectivity factor

4.4 Varying the Join Selectivity Factor

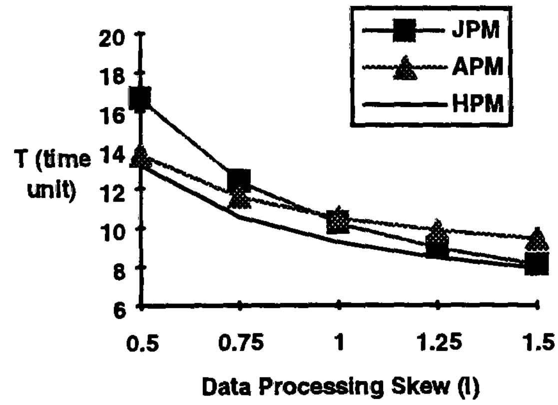
The join selectivity factor has significant influence on parallel aggregation processing as it determines the number of tuples resulting from join intermediately. After that, those tuples are processed for aggregation and evaluated with the predicates. Eventually, the qualified tuples are unioned to form the query result. Lower selectivity factor involves less aggregation processing time and transferring time, and thus favours *JPM* as displayed in Figure 10. Less processors will reduce the impact of the entire second relation (both communication and processing) on running time so it favours *APM*.

4.5 Varying the Degree of Skewness

Figure 11(a) indicates the tendency of the performance when the data processing skew changes accordingly with the data partitioning skew whereas Figure 11(b) provides the comparison when we ignore the data partitioning skew, i.e. $\alpha = 1/N$ and alter the data processing skew. The values on the horizontal axis of both figures represent the expanding skewness factor which then is multiplied by the basic unit given by Zipf distribution. Unlike the α, the β is inversely proportional to data processing skew and the larger the factor β, the less the data processing skew is. We conclude from Figure 11 that either of partitioning skew or processing skew degrades the performance of parallel processing, *HPM* outperforms *APM* and *JPM* even in the presence of skewness, and *APM* is less affected by the skewness compared with *JPM*.

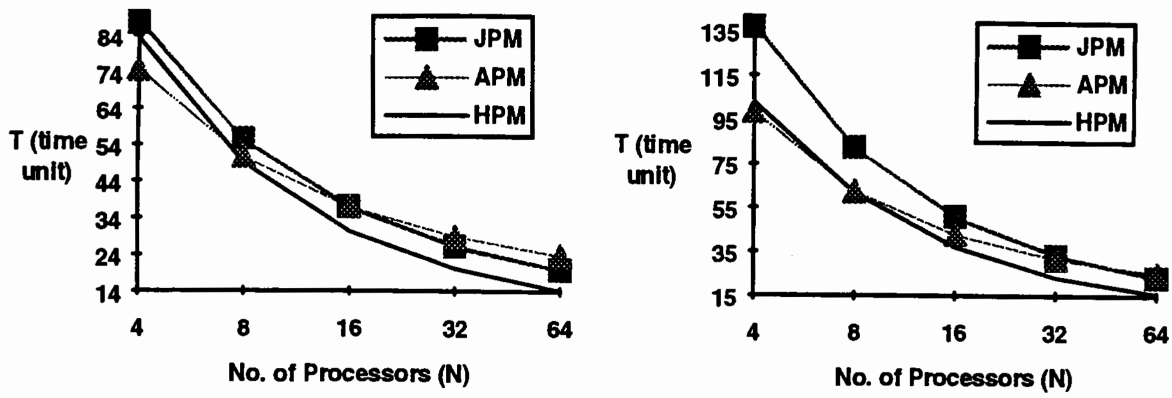


(a). Data partition skew



(b). Data processing skew

Figure 11. Varying the skewness with 16 processors



(a). *Agg(i)*=0.5, *Sel(i)*=5(*N*/*r*) (b). *Agg(i)*=0.6, *Sel(i)*=10(*N*/*r*)

Figure 12. Varying the number of processors

4.6 Varying the Number of Processors

One of the desired goals of parallel processing is to have linear scale up which can be achieved when twice as much processors perform twice as large a task in the same time. When the number of processors is increased, as we expected, the performance is upgraded in spite of the skewness shown in Figure 12. *APM* performs extremely well when the number of processors is small, and it is even better than *HPM* because the number of clusters in *HPM* may not be optimised and less processors make the communication cost insignificant which favours *APM*. However, when the database system scales up, both *HPM* and *JPM* performs better than *APM*.

5. Conclusion

Traditionally, join operation is processed before aggregation operation and relations are partitioned on join attribute. In this paper, we demonstrate that group-by attribute may also be chosen as the partition attribute and present three parallel methods for aggregation queries, *JPM*, *APM*, and *HPM*. These methods differ in the way of distributing query relations, i.e. partitioning on the join attribute, on the group-by attribute, or on a combination of both; consequently, they give rise to different query execution costs. In addition, the problem of data skew has been taken into account in the proposed methods as it may adversely affect the performance advantage of the parallel processing. A performance comparison of these methods has been provided under various circumstances of queries and processors. The results show that when the join selectivity factor is small and the degree of skewness is low, *JPM* leads to less cost; otherwise *APM* is desirable. Nevertheless, the hybrid method (*HPM*) is always superior to the other two methods since the data partitioning is adaptively made on both join attribute and group-by attribute. In addition, it is found that the partitioning on group-by attribute method is insensitive to the aggregation factor and thus the method will simplify algorithm design and implementation.

Reference

[Alma94] Almasi G. S. and A. Gottlieb, "Highly parallel computing", *The Benjamin/Cummings Publishing Company, Inc.*, 1994

[Bell89] Bell D. A., D. H. O. Ling, and S. McClean S., "Pragmatic estimation of join sizes and attribute correlations", *Proceedings of 5th IEEE Data Engineering Conference*, Los Angeles, 1989

[Bitt83] Bitton D., H. Boral, D. J. DeWitt, and W. K. Wilkinson, "Parallel algorithms for the execution of relational database operations", *ACM Transactions on Database Systems*, Vol 8, No.3, September 1993

[Bult87] Bultzingsloewen G., "Translating and optimizing SQL queries having aggregate", *Proceedings of the 13th International Conference on Very Large Data Bases*, Brighton, U.K., 1987

[Daya87] Dayal U., "Of nests and trees: a unified approach to processing queries that contain nested subqueries, aggregates, and quantitiers", *Proceedings of the 13th International Conference on Very Large Data Bases*, Brighton, U.K., 1987

[DeWi92] DeWitt D.J., J. F. Naughton, D. A. Schneider, and S. Seshadri, "Practical skew handling in parallel joins", *Proceedings of the 8th International Conference on Very Large Data Bases*, Vancouver, British Columbia, Canada 1992

[Ghan94] Ghandeharizadeh S. and D. J. DeWitt, "MAGIC: a multiattribute declustering mechanism for multiprocessor database machines", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 5, 1994

[Grae93] Graefe G., "Query evaluation techniques for large databases", *ACM Computing Surveys*, Vol.25, No.2, June 1993

[Kell91] Keller A. M. and S. Roy, "Adaptive parallel hash join in main-memory databases", *Proceedings of the 1st International Conference on Parallel and Distributed Information Systems*, Dec 1991

[Kim82] Kim W., "On optimizing an SQL-like nested query", *ACM Transactions on Database Systems*, Vol.7, No.3, September 1982

[Knut73] Knuth D. E., "The art of computer programming", Volume 3, *Addison-Wesley Publishing Company, INC.*, 1973

[Leun94] Leung C. H. C. and K. H. Liu, "Skewness analysis of parallel join execution", Technical Report, Department of Computer and Mathematical Sciences, Victoria University of Technology, Melbourne, Victoria, Australia, 1994

[Liu95] Liu K. H., Leung C. H. C., and Y. Jiang, "Analysis and taxonomy of skew in parallel database systems", *High Performance Computing Symposium'95*, Montreal, Canada, July 1995, pp 304-315

[Liu96] Liu K. H., Y. Jiang, and C.H.C. Leung, "Query execution in the presence of data skew in parallel databases", *Australian Computer Science Communications*, vol 18, no 2, 1996, pp. 157-166

[Moha94] Mohan C., Pirahesh H., Tang W. G., and Y. Wang, "Parallelism in relational database management systems" *IBM Systems Journal*, Vol.33, No.2, 1994

[Schn89] Schneider D. A. and D. J. DeWitt, "A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment", *Proceedings of the International Conference on ACM SIGMOD*, New York, U.S.A. 1989

[Seve90] Severance C., Pramanik S., and P. Wolberg, "Distributed linear hashing and parallel projection in main memory databases", *Proceedings of the 16th International Conference on Very Large Data Bases*, Brisbane, Australia 1990

[Shat94] Shatdal A. and J. F. Naughton, "Processing aggregates in parallel database systems", *Computer Sciences Technical Report # 1233*, Computer Sciences Department, University of Wisconsin-Madison, June 1994

[Wolf93a] Wolf J. L., Dias D. M., and P. S. Yu, "A parallel sort-merge join algorithm for managing data skew", *IEEE Transactions On Parallel And Distributed Systems*, Vol. 4, No.1, January 1993

[Wolf93b] Wolf J. L., Yu P. S., Turek J. and D. M. Dias, "A parallel hash join algorithm for managing data skew", *IEEE Transactions On Parallel and Distributed Systems*, Vol.4, No. 12, December 1993

[Yan94] Yan W. P. and P. Larson, "Performing group-by before join", *Proceedings of the International Conference on Data Engineering*, 1994