PARALLEL PROCESSING OF ”GROUP-BY JOIN” QUERIES ON SHARED NOTHING MACHINES

M. Al Hajj Hassan and M. Bamha
LIFO, Université d'Orléans
B.P. 6759, 45067 Orléans Cedex 2, France.
{*mohamad.alhajjhassan,mostafa.bamha*}*@univ-orleans.fr*

Keywords: PDBMS[a], Parallel joins, Data skew, Join product skew, GroupBy-Join queries, BSP cost model.

[a]PDBMS : Parallel DataBase Management Systems.

Abstract: SQL queries involving join and group-by operations are frequently used in many decision support applications. In these applications, the size of the input relations is usually very large, so the parallelization of these queries is highly recommended in order to obtain a desirable response time. The main drawbacks of the presented parallel algorithms that treat this kind of queries are that they are very sensitive to data skew and involve expensive communication and Input/Output costs in the evaluation of the join operation. In this paper, we present an algorithm that minimizes the communication cost by performing the group-by operation before redistribution where only tuples that will be present in the join result are redistributed. In addition, it evaluates the query without the need of materializing the result of the join operation and thus reducing the Input/Output cost of join intermediate results. The performance of this algorithm is analyzed using the scalable and portable BSP (Bulk Synchronous Parallel) cost model which predicts a near-linear speed-up even for highly skewed data.

1 INTRODUCTION

Data warehousing, On-Line Analytical Processing (OLAP) and other multidimensional analysis technologies have been employed by data analysts to extract interesting information from large database systems in order to improve the business performance and help the organisations in decision making. In these applications, aggregate queries are widely used to summarize large volume of data which may be the result of the join of several tables containing billions of records (Datta et al., 1998; Chaudhuri and Shim, 1994). The main difficulty in such applications is that the result of these analytical queries must be obtained interactively (Datta et al., 1998; Tsois and Sellis, 2003) despite the huge volume of data in warehouses and their rapid growth especially in OLAP systems (Datta et al., 1998). For this reason, parallel processing of these queries is highly recommended in order to obtain acceptable response time (Bamha, 2005). Research has shown that join, which is one of the most expensive operations in DBMS, is parallelizable with near-linear speed-up only in ideal cases (Bamha and Hains, 2000). However, data skew degrades the performance of parallel systems (Bamha and Hains, 1999; Bamha and Hains, 2000; Seetha and Yu, 1990; Hua and Lee, 1991; Wolf et al., 1994; DeWitt et al., 1992). Thus, effective parallel algorithms that evenly distribute the load among processors and minimizes the inter-site communication must be employed in parallel and distributed systems in order to obtain acceptable performance.

In traditional algorithms that treat ”GroupBy-Join” queries[1], join operations are performed in the first step and then the group-by operation (Chaudhuri and Shim, 1994; Yan and Larson, 1994). But the response time of these queries is significantly reduced if the group-by operation is performed before the join (Chaudhuri and Shim, 1994), because group-by reduces the size of the relations thus minimizing the join and data redistribution costs. Several algorithms that perform the group-by operation before the join operation were presented in the literature (Shatdal

[1]GroupBy-Join queries are queries involving group-by and join operations.

and Naughton, 1995; Taniar et al., 2000; Taniar and Rahayu, 2001; Yan and Larson, 1994).
In the ”Early Distribution Schema” algorithm presented in (Taniar and Rahayu, 2001), all the tuples of the tables are redistributed before applying the join or the group-by operations, thus the communication cost in this algorithm is very high. However, the cost of its join operation is reduced because the group-by is performed before the expensive join operation. In the second algorithm, ”Early GroupBy Scheme” (Taniar and Rahayu, 2001), the group-by operation is performed before the distribution and the join operations thus reducing the volume of data. But in this algorithm, all the tuples of the group-by results are redistributed even if they do not contribute in the join result. This is a drawback, because in some cases only few tuples of relations formed of million of tuples contribute in the join operation, thus the distribution of all these tuples is useless.

These algorithms fully materialize the intermediate results of the join operations. This is a significant drawback because the size of the result of this operation is generally large with respect to the size of the input relations. In addition, the Input/Output cost in these algorithms is very high where it is reasonable to assume that the output relation cannot fit in the main memory of each processor, so it must be reread in order to evaluate the aggregate function.

In this paper, we present a new parallel algorithm used to evaluate ”GroupBy-Join” queries on Shared Nothing machines (a multiprocessors machine where each processor has its own memory and disks (DeWitt and Gray, 1992)). In this algorithm, we do not materialize the join operation as in the traditional algorithms where the join operation is evaluated first and then the group-by and aggregate functions (Yan and Larson, 1994). So the Input/Output cost is minimal because we do not need to save the huge volume of data that results from the join operation.
We also use the histograms of both relations in order to find the tuples which will be present in the join result. After finding these tuples, we apply on them the grouping and aggregate function, in each processor, before performing the join. Using our approach, we reduce the size of data and communication costs to minimum. It is proved in (Bamha and Hains, 2000; Bamha and Hains, 1999), using the BSP model, that histogram management has a negligible cost when compared to the gain it provides in reducing the communication cost. In addition, Our algorithm avoids the problem of data skew because the hashing functions are only applied on histograms and not on input relations.
The performance of this algorithm is analyzed using the scalable and portable BSP cost model (Skillicorn et al., 1997) which predicts for our algorithm a near-linear speed-up even for highly skewed data.

The rest of the paper is organized as follows. In section 2, we present the BSP cost model used to evaluate the processing time of the different phases of the algorithm. In section 3, we give an overview of different computation methods of ”GroupBy-Join” queries. In section 4, we describe our algorithm. We then conclude in section 5.

2 THE BSP COST MODEL

Bulk-Synchronous Parallel (BSP) cost model is a programming model introduced by L. Valiant (Valiant, 1990) to offer a high degree of abstraction like PRAM models and yet allow portable and predictable performance on a wide variety of multi-processor architectures (Bisseling, 2004; Skillicorn et al., 1997). A BSP computer contains a set of processor-memory pairs, a communication network allowing inter-processor delivery of messages and a global synchronization unit which executes collective requests for a synchronization barrier. Its performance is characterized by 3 parameters expressed as multiples of the local processing speed:

- the number of processor-memory pairs p,
- the time l required for a global synchronization,
- the time g for collectively delivering a 1-relation (communication phase where each processor receives/sends at most one word). The network is assumed to deliver an h-relation in time $g * h$ for any arity h.

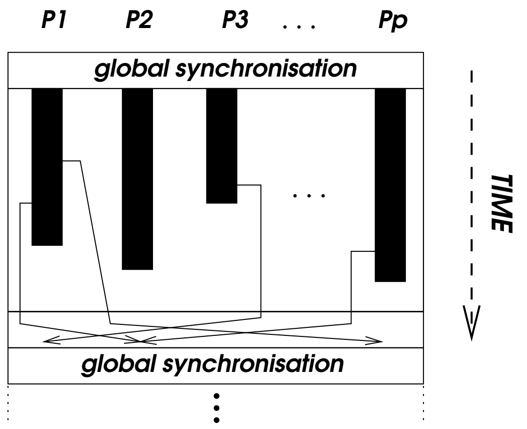

Figure 1: A BSP superstep.

A BSP program is executed as a sequence of *supersteps*, each one divided into (at most) three successive and logically disjoint phases. In the first phase each processor uses only its local data to perform sequential computations and to request data transfers to/from other nodes. In the second phase the network delivers the requested data transfers and in the third phase a global synchronization barrier occurs, making the transferred data available for the next superstep. The execution time of a superstep s is thus the sum of the maximal local processing time, of the data delivery time and of the global synchronization time:

$$\text{Time}(s) = \max_{i:processor} w_i^{(s)} + \max_{i:processor} h_i^{(s)} * g + l$$

where $w_i^{(s)}$ is the local processing time on processor i during superstep s and $h_i^{(s)} = \max\{h_{i+}^{(s)}, h_{i-}^{(s)}\}$ where $h_{i+}^{(s)}$ (resp. $h_{i-}^{(s)}$) is the number of words transmitted (resp. received) by processor i during superstep s. The execution time, $\sum_s \text{Time}(s)$, of a BSP program composed of S supersteps is therefore a sum of 3 terms: $W+H*g+S*l$ where $W = \sum_s \max_i w_i^{(s)}$ and $H = \sum_s \max_i h_i^{(s)}$. In general W, H and S are functions of p and of the size of data n, or (as in the present application) of more complex parameters like data skew and histogram sizes. To minimize execution time of a BSP algorithm, design must jointly minimize the number S of supersteps and the total volume h (resp. W) and imbalance $h^{(s)}$ (resp. $W^{(s)}$) of communication (resp. local computation).

3 COMPUTATION OF ”GROUP-BY JOIN” QUERIES

In DBMS, the aggregate functions can be applied on the tuples of a single table, but in most SQL queries, they are applied on the output of the join of multiple relations. In the later case, we can distinguish two types of ”GroupBy-Join” queries. We will illustrate these two types using the following example.
In this example, we have three relations that represent respectively Suppliers, Products and the quantity of a product shipped by a supplier in a specific date.

```
SUPPLIER (Sid, Sname, City)
PRODUCT (Pid, Pname, Category)
SHIPMENT (Sid, Pid, Date, Quantity)
```

Query 1

```
Select p.Pid, SUM (Quantity)
From PRODUCT as p, SHIPMENT as s
Where p.Pid = s.Pid
Group By p.Pid
```

Query 2

```
Select p.Category, SUM (Quantity)
From PRODUCT as p, SHIPMENT as s
Where p.Pid = s.Pid
Group By p.Category
```

The purpose of $Query1$ is to find the total quantity of every product shipped by all the suppliers, while that of $Query2$ is to find the total amount of every category of product shipped by all the suppliers.
The difference between $Query1$ and $Query2$ lies in the group-by and join attributes. In $Query1$, the join attribute (Pid) and the group-by attribute are the same. In this case, it is preferable to carry out the group-by operation first and then the join operation (Taniar et al., 2000; Taniar and Rahayu, 2001), because the group-by operation reduces the size of the relations to be joined. As a consequence, applying the group-by operation before the join operation in PDBMS results in a huge gain in the communication cost and the execution time of the ”GroupBy-Join” queries.
In the contrary, this can not be applied on Query 2, because the join attribute (Pid) is different from the group-by attribute ($category$).

In this paper, we focus on ”GroupBy-Join” queries when the join attributes are part of the group-by attributes. In our algorithm, we succeeded to redistribute only tuples that will be present in the join result after applying the aggregate function. Therefore, the communication cost is reduced to minimum.

4 PRESENTED ALGORITHM

In this section, we present a detailed description of our parallel algorithm used to evaluate ”GroupBy-Join” queries when the join attributes are part of the group-by attributes. We assume that the relation R (resp. S) is partitioned among processors by horizontal fragmentation and the fragments R_i for $i = 1, ..., p$ are almost of the same size on each processor, i.e. $|R_i| \simeq \frac{|R|}{p}$ where p is the number of processors.
For simplicity of description and without loss of generality, we consider that the query has only one join attribute x and that the group-by attribute set consists of x, an attribute y of R and another attribute z of S . We also assume that the aggregate function f is applied on the values of the attribute u of S. So the treated query is the following:

```
Select $R.x, R.y, S.z, f(S.u)$
From $R, S$
Where $R.x = S.x$
Group By $R.x, R.y, S.z$
```

In the rest of this paper, we use the following notation for each relation $T \in \{R, S\}$:

- T_i denotes the fragment of relation T placed on processor i,

- $Hist^w(T)$ denotes the histogram[2] of relation T with respect to the attribute w, i.e. a list of pairs (v, n_v) where $n_v \neq 0$ is the number of tuples of relation T having the value v for the attribute w. The histogram is often much smaller and never larger than the relation it describes,
- $Hist^w(T_i)$ denotes the histogram of fragment T_i,
- $Hist_i^w(T)$ is processor i's fragment of the histogram of T,
- $Hist^w(T)(v)$ is the frequency (n_v) of value v in relation T,
- $Hist^w(T_i)(v)$ is the frequency of value v in sub-relation T_i,
- $AGGR^w_{f,u}(T)$ [3] is the result of applying the aggregate function f on the values of the aggregate attribute u of every group of tuples of T having identical values of the group-by attribute w. $AGGR^w_{f,u}(T)$ is formed of a list of tuples having the form (v, f_v) where f_v is the result of applying the aggregate function on the group of tuples having value v for the attribute w (w may be formed of more than one attribute),
- $AGGR^w_{f,u}(T_i)$ denotes the result of applying the aggregate function on the attribute u of the fragment T_i,
- $AGGR^w_{f,u,i}(T)$ is processor i's fragment of the result of applying the aggregate function on T,
- $AGGR^w_{f,u}(T)(v)$ is the result f_v of the aggregate function of the group of tuples having value v for the group-by attribute w in relation T,
- $AGGR^w_{f,u}(T_i)(v)$ is the result f_v of the aggregate function of the group of tuples having value v for the group-by attribute w in sub-relation T_i,
- $\|T\|$ denotes the number of tuples of relation T, and
- $|T|$ denotes the size (expressed in bytes or number of pages) of relation T.

The algorithm proceeds in four phases. We will give an upper bound of the execution time of each superstep using BSP cost model. The notation $O(...)$ hides only small constant factors: *they depend only on the program implementation but neither on data nor on the BSP machine parameters.*

Phase 1: Creating local histograms

In this phase, the local histograms $Hist^x(R_i)_{i=1,\dots,p}$ (resp. $Hist^x(S_i)_{i=1,\dots,p}$) of blocks R_i (resp. S_i) are created in parallel by a scan of the fragment R_i (resp. S_i), on processor i, in time $c_{i/o} * \max_{i=1,\dots,p} |R_i|$ (resp. $c_{i/o} * \max_{i=1,\dots,p} |S_i|$) where $c_{i/o}$ is the cost of writing/reading a page of data from disk.

In addition, the local fragments $AGGR^{x,z}_{f,u}(S_i)_{i=1,\dots,p}$ of blocks S_i are created on the fly while scanning relation S_i in parallel, on each processor i, by applying the aggregate function f on every group of tuples having identical values of the couple of attributes (x, z). At the same time, the local histograms $Hist^{x,y}(R_i)_{i=1,\dots,p}$ are also created.
(In this algorithm the aggregate function may be MAX, MIN, SUM or $COUNT$. For the aggregate AVG a similar algorithm that merges the $COUNT$ and the SUM algorithms is applied).

In principle, this phase costs:

$$Time_{phase1} = O\big(c_{i/o} * \max_{i=1,\dots,p}(|R_i| + |S_i|)\big).$$

Phase 2: Creating the histogram of $R \bowtie S$

The first step in this phase is to create the histograms $Hist_i^x(R)$ and $Hist_i^x(S)$ by a parallel hashing of the histograms $Hist^x(R_i)$ and $Hist^x(S_i)$. After hashing, each processor i merges the messages it received to constitute $Hist_i^x(R)$ and $Hist_i^x(S)$. While merging, processor i also retains a trace of the network layout of the values d of the attribute x in its $Hist_i^x(R)$ (resp. $Hist_i^x(S)$): this is nothing but the collection of messages it has just received. This information will help in forming the communication templates in phase 3.

The cost of redistribution and merging step is (cf. to proposition 1 in (Bamha and Hains, 2005)):

$$Time_{phase2.a} = O\Big(min\big(g * |Hist^x(R)| + ||Hist^x(R)||, g * \frac{|R|}{p} + \frac{||R||}{p}\big) + min\big(g * |Hist^x(S)| + ||Hist^x(S)||, g * \frac{|S|}{p} + \frac{||S||}{p}\big) + l\Big),$$

where g is the BSP communication parameter and l the cost of a barrier of synchronisation.

We recall that, in the above equation, for a relation $T \in \{R, S\}$, the term $min(g * |Hist^x(T)| + ||Hist^x(T)||, g * \frac{|T|}{p} + \frac{||T||}{p})$ is the necessary time to compute $Hist^x_{i=1,\dots,p}(T)$ starting from the local histograms $Hist^x(T_i)_{i=1,\dots,p}$.

The histogram[4] $Hist_i^x(R \bowtie S)$ is then computed on each processor i by intersecting $Hist_i^x(R)$ and

[2] Histograms are implemented as a balanced tree (B-tree): a data structure that maintains an ordered set of data to allow efficient search and insert operations.

[3] $AGGR^w_{f,u}(T)$ is implemented as a balanced tree (B-tree).

[4] The size of $Hist(R \bowtie S) \equiv Hist(R) \cap Hist(S)$ is generally very small compared to $|Hist(R)|$ and $|Hist(S)|$ because $Hist(R \bowtie S)$ contains only values that appears in both relations R and S.

$Hist_i^x(S)$ in time:

$$Time_{phase2.b} = O\Big(\max_{i=1,\dots,p} \big(min(||Hist_i^x(R)||, ||Hist_i^x(S)||)\big)\Big).$$

The total cost of this phase is:

$$Time_{phase2} = Time_{phase2.a} + Time_{phase2.b} = O\Big(min\big(g * |Hist^x(R)| + ||Hist^x(R)||, g * \frac{|R|}{p} + \frac{||R||}{p}\big) + min\big(g * |Hist^x(S)| + ||Hist^x(S)||, g * \frac{|S|}{p} + \frac{||S||}{p}\big) + \max_{i=1,\dots,p} \big(min(||Hist_i^x(R)||, ||Hist_i^x(S)||)\big) + l\Big).$$

Phase 3: Data redistribution

In order to reduce the communication cost, only tuples of $Hist^{x,y}(R)$ and $AGGR_{f,u}^{x,z}(S)$ that will be present in the join result will be redistributed.
To this end, we first compute on each processor j the intersections $\overline{Hist}^{(j)x}(R_i) = Hist^{(j)x}(R_i) \cap Hist_j(R \bowtie S)$ and $\overline{Hist}^{(j)x}(S_i) = Hist^{(j)x}(S_i) \cap Hist_j(R \bowtie S)$ for $i = 1, ..., p$ where $Hist^{(j)x}(R_i)(resp. Hist^{(j)x}(S_i))$ is the fragment of $Hist^x(R_i)$ (resp. $Hist^x(S_i)$) which was sent by processor i to processor j in the second phase.
The cost of this step is:

$$O(\sum_i ||Hist^{(j)x}(R_i)|| + \sum_i ||Hist^{(j)x}(S_i)||).$$

We recall that,
$\sum_i ||Hist^{(j)x}(R_i)|| = || \cup_i Hist^{(j)x}(R_i)|| \leq min(||Hist^x(R)||, \frac{||R||}{p})$
and
$\sum_i ||Hist^{(j)x}(S_i)|| = || \cup_i Hist^{(j)x}(S_i)|| \leq min(||Hist^x(S)||, \frac{||S||}{p})$,
thus the total cost of this step is:

$$Time_{phase3.a} = O\Big(min\big(||Hist^x(R)||, \frac{||R||}{p}\big) + min\big(||Hist^x(S)||, \frac{||S||}{p}\big)\Big).$$

Now each processor j sends each fragment $\overline{Hist}^{(j)x}(R_i)$ (resp. $\overline{Hist}^{(j)x}(S_i)$) to processor i. Thus, each processor i receives $\sum_j |\overline{Hist}^{(j)x}(R_i)| + \sum_j |\overline{Hist}^{(j)x}(S_i)|$ pages of data from the other processors.
In fact, $Hist^x(R_i) = \cup_j Hist^{(j)x}(R_i)$ and $|Hist^x(R_i)| = \sum_j |Hist^{(j)x}(R_i)| \geq \sum_j |Hist^{(j)x}(R_i) \cap Hist^x(R \bowtie S)|$, thus $|Hist^x(R_i)| \geq \sum_j |\overline{Hist}^{(j)x}(R_i)|$ (this also applies to $Hist^x(S_i)$).
Therefore, the total cost of this stage of communication is at most:

$$Time_{phase3.b} = O\Big(g * \big(|\overline{Hist}^x(R_i)| + |\overline{Hist}^x(S_i)|\big) + l\Big).$$

Remark 1 *$\cup_j \overline{Hist}^{(j)x}(R_i)$ is simply the intersection of $Hist^x(R_i)$ and the histogram $Hist^x(R \bowtie S)$ which will be noted:*

$$\overline{Hist}^x(R_i) = \cup_j \overline{Hist}^{(j)x}(R_i) = Hist^x(R_i) \cap Hist^x(R \bowtie S).$$

Hence $\overline{Hist}^x(R_i)$ is only the restriction of the fragment of $Hist^x(R_i)$ to values which will be present in the join of the relations R and S. (this also applies to $\overline{Hist}^x(S_i)$).

Now, each processor obeys all the distributing orders it has received, so only tuples of $\overline{Hist}^{x,y}(R_i) = Hist^{x,y}(R_i) \cap \overline{Hist}^x(R_i)$ and $\overline{AGGR}_{f,u}^{x,z}(S_i) = AGGR_{f,u}^{x,z}(S_i) \cap \overline{Hist}^x(S_i)$ are redistributed.

To this end, we first evaluate $\overline{Hist}^{x,y}(R_i)$ and $\overline{AGGR}_{f,u}^{x,z}(S_i)$. The cost of this step is of order:

$$Time_{phase3.c} = O\Big(\max_{i=1,\dots,p} \big(||Hist^{x,y}(R_i)|| + ||AGGR_{f,u}^{x,z}(S_i)||\big)\Big),$$

which is the necessary time to traverse all the tuples of $Hist^{x,y}(R_i)$ and $AGGR_{f,u}^{x,z}(S_i)$ and access $\overline{Hist}^x(R_i)$ and $\overline{Hist}^x(S_i)$ respectively on each processor i.

Now, each processor i distributes the tuples of $\overline{Hist}^{x,y}(R_i)$ and $\overline{AGGR}_{f,u}^{x,z}(S_i)$. After distribution, all the tuples of $\overline{Hist}^{x,y}(R_i)$ and $\overline{AGGR}_{f,u}^{x,z}(S_i)$ having the same values of the join attribute x are stored on the same processor. So, each processor i merges the blocks of data received from all the other processors in order to create $\overline{Hist}_i^{x,y}(R)$ and $\overline{AGGR}_{f,u,i}^{x,z}(S)$.
The cost of distributing and merging the tuples is of order (cf. to proposition 1 in (Bamha and Hains, 2005)):

$$Time_{phase3.d} = O\Big(min\big(g * |\overline{Hist}^{x,y}(R)| + ||\overline{Hist}^{x,y}(R)||, g * \frac{|R|}{p} + \frac{||R||}{p}\big) + min\big(g * |\overline{AGGR}_{f,u}^{x,z}(S)| + ||\overline{AGGR}_{f,u}^{x,z}(S)||, g * \frac{|S|}{p} + \frac{||S||}{p}\big) + l\Big),$$

where the terms:

$$min\big(g * |\overline{Hist}^{x,y}(R)| + ||\overline{Hist}^{x,y}(R)||, g * \frac{|R|}{p} + \frac{||R||}{p}\big)$$

and

$$min\big(g * |\overline{AGGR}_{f,u}^{x,z}(S)| + ||\overline{AGGR}_{f,u}^{x,z}(S)||, g * \frac{|S|}{p} + \frac{||S||}{p}\big)$$

represent the necessary time to compute $\overline{Hist}_i^{x,y}(R)$ and $\overline{AGGR}_{f,u,i}^{x,z}(S)$ starting from $\overline{Hist}^{x,y}(R_i)$ and $\overline{AGGR}_{f,u}^{x,z}(S_i)$ respectively.

The total time of the redistribution phase is:

$$Time_{phase3} =$$
$$O\Big(min\big(g * |\overline{Hist}^{x,y}(R)| + ||\overline{Hist}^{x,y}(R)||,$$
$$g * \frac{|R|}{p} + \frac{||R||}{p}\big) + min\big(||Hist^x(R)||, \frac{||R||}{p}\big)$$
$$+ min\big(g * |\overline{AGGR}_{f,u}^{x,z}(S)| + ||\overline{AGGR}_{f,u}^{x,z}(S)||,$$
$$g * \frac{|S|}{p} + \frac{||S||}{p}\big) + min\big(||Hist^x(S)||, \frac{||S||}{p}\big)$$
$$+ \max_{i=1,\dots,p} \big(||Hist^{x,y}(R_i)|| + ||AGGR_{f,u}^{x,z}(S_i)||\big) + l\Big).$$

We mention that we only redistribute $\overline{Hist}^{x,y}(R_i)$ and $\overline{AGGR}_{f,u}^{x,z}(S_i)$ and their sizes are generally very small compared to $|R_i|$ and $|S_i|$ respectively. In addition, the size of $|Hist^x(R \bowtie S)|$ is generally very small compared to $|Hist^x(R)|$ and $|Hist^x(S)|$. Thus, we reduce the communication cost to minimum.

Phase 4: Global computation of the aggregate function

In this phase, we compute the global aggregate function on each processor. We use the following algorithm where $AGGR_{f,u,i}^{x,y,z}(R \bowtie S)$ holds the final result on each processor i. The tuples of $AGGR_{f,u,i}^{x,y,z}(R \bowtie S)$ have the form (x, y, z, v) where v is the result of the aggregate function.

```
Par (on each node in parallel) $i = 1, ..., p$
  $AGGR_{f,u,i}^{x,y,z}(R \bowtie S)$ = NULL [5]
  For every tuple $t$ of relation $\overline{Hist}_i^{x,y}(R)$ do
    $freq = \overline{Hist}_i^{x,y}(R)(t.x, t.y)$
    For every entry $v_1$ = $\overline{AGGR}_{f,u,i}^{x,z}(S)(t.x, z)$ do
      Insert a new tuple $(t.x, t.y, z, f(v_1, freq))$
      into $AGGR_{f,u,i}^{x,y,z}(R \bowtie S)$;
    EndFor
  EndFor
EndPar
```

The time of this phase is: $O\big(\max_{i=1,\dots,p} ||AGGR_{f,u,i}^{x,y,z}(R \bowtie S)||\big)$, because the combination of the tuples of $\overline{Hist}_i^{x,y}(R)$ and $\overline{AGGR}_{f,u,i}^{x,z}(S)$ is performed to generate all the tuples of $AGGR_{f,u,i}^{x,y,z}(R \bowtie S)$.

Remark 2 *In practice, the imbalance of the data related to the use of the hash functions can be due to:*

- *a bad choice of the hash function used. This imbalance can be avoided by using the hashing techniques presented in the literature making it possible to distribute evenly the values of the join attribute with a very high probability (Carter and Wegman, 1979),*
- *an intrinsic data imbalance which appears when some values of the join attribute appear more frequently than others. By definition a hash function maps tuples having the same join attribute values to the same processor. These is no way for a clever hash function to avoid load imbalance that result from these repeated values (DeWitt et al., 1992). But this case cannot arise here owing to the fact that histograms contains only distinct values of the join attribute and the hashing functions we use are always applied to histograms.*

[5]This instruction creates a B-tree to store histogram's entries.

The global cost of evaluating the "GroupBy-Join" queries is of order:

$$Time_{total} = O\Big(c_{i/o} * \max_{i=1,\dots,p} (|R_i| + |S_i|)$$
$$+ \max_{i=1,\dots,p} ||AGGR_{f,u,i}^{x,y,z}(R \bowtie S)||$$
$$+ min(g * |Hist^x(R)| + ||Hist^x(R)||, g * \frac{|R|}{p} + \frac{||R||}{p})$$
$$+ min(g * |Hist^x(S)| + ||Hist^x(S)||, g * \frac{|S|}{p} + \frac{||S||}{p})$$
$$+ min\big(g * |\overline{Hist}^{x,y}(R)| + ||\overline{Hist}^{x,y}(R)||,$$
$$g * \frac{|R|}{p} + \frac{||R||}{p}\big)$$
$$+ min\big(g * |\overline{AGGR}_{f,u}^{x,z}(S)| + ||\overline{AGGR}_{f,u}^{x,z}(S)||,$$
$$g * \frac{|S|}{p} + \frac{||S||}{p}\big)$$
$$+ \max_{i=1,\dots,p} \big(||Hist^{x,y}(R_i)|| + ||AGGR_{f,u}^{x,z}(S_i)||\big) + l\Big).$$

Remark 3 *In the traditional algorithms, the aggregate function is applied on the output of the join operation. The sequential evaluation of the "groupBy-Join" queries requires at least the following lower bound:* $bound_{inf_1} = \Omega\big(c_{i/o} * (|R| + |S| + |R \bowtie S|)\big)$. *Parallel processing with p processors requires therefore:* $bound_{inf_p} = \frac{1}{p} * bound_{inf_1}$.

Using our approach in the evaluation of the "GroupBy-Join" queries, we only redistribute tuples that will be effectively present in the "groupBy-Join" result, which reduces the communication cost to minimum. This algorithm has an asymptotic optimal complexity because all the terms in $Time_{total}$ are bounded by those of $bound_{inf_p}$.

5 CONCLUSION

The algorithm presented in this paper is used to evaluate the "GroupBy-Join" queries on Shared Nothing machines when the join attributes are part of the group-by attributes. Our main contribution in this algorithm is that we do not need to materialize the costly join operation which is necessary in all the other algorithms presented in the literature, thus we reduce its Input/Output cost. It also helps us to avoid the effect of data skew which may result from computing the intermediate join results and from redistributing all the tuples if AVS (Attribute Value Skew) exists in the relation. In addition, we partially evaluate the aggregate function before redistributing the data between processors or evaluating the join operation, because group-by and aggregate functions reduce the volume of data. To reduce the communication cost to minimum, we use the histograms to distribute only the tuples of the grouping result that will effectively be present in the output of the join operation. This algorithm is proved to have a near-linear speed-up, using the BSP cost model, even for highly skew data. Our experience with the join operation (Bamha and Hains, 2000; Bamha and Hains, 1999; Bamha and Hains, 2005) is evidence that the above theoretical analysis is accurate in practice.

REFERENCES

Bamha, M. (2005). An optimal and skew-insensitive join and multi-join algorithm for ditributed architectures. In *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA'2005). 22-26 August, Copenhagen, Danemark*, volume 3588 of *Lecture Notes in Computer Science*, pages 616–625. Springer-Verlag.

Bamha, M. and Hains, G. (2000). A skew insensitive algorithm for join and multi-join operation on Shared Nothing machines. In *the 11th International Conference on Database and Expert Systems Applications DEXA'2000*, volume 1873 of *Lecture Notes in Computer Science*, London, United Kingdom. Springer-Verlag.

Bamha, M. and Hains, G. (2005). An efficient equi-semi-join algorithm for distributed architectures. In *Proceedings of the 5th International Conference on Computational Science (ICCS'2005). 22-25 May, Atlanta, USA*, volume 3515 of *Lecture Notes in Computer Science*, pages 755–763. Springer-Verlag.

Bamha, M. and Hains, G. (September 1999). A frequency adaptive join algorithm for Shared Nothing machines. *Journal of Parallel and Distributed Computing Practices (PDCP), Volume 3, Number 3, pages 333-345*. Appears also in Progress in Computer Research, F. Columbus Ed. Vol. II, Nova Science Publishers, 2001.

Bisseling, R. H. (2004). *Parallel Scientific Computation : A Structured Approach using BSP and MPI*. Oxford University Press, USA.

Carter, J. L. and Wegman, M. N. (April 1979). Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154.

Chaudhuri, S. and Shim, K. (1994). Including Group-By in Query Optimization. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 354–366, Santiago, Chile.

Datta, A., Moon, B., and Thomas, H. (1998). A case for parallelism in datawarehousing and OLAP. In *Ninth International Workshop on Database and Expert Systems Applications, DEXA 98*, IEEE Computer Society, pages 226–231, Vienna.

DeWitt, D. J. and Gray, J. (1992). Parallel database systems : The future of high performance database systems. *Communications of the ACM*, 35(6):85–98.

DeWitt, D. J., Naughton, J. F., Schneider, D. A., and Seshadri, S. (1992). Practical Skew Handling in Parallel Joins. In *Proceedings of the 18th VLDB Conference*, pages 27–40, Vancouver, British Columbia, Canada.

Hua, K. A. and Lee, C. (1991). Handling data skew in multiprocessor database computers using partition tuning. In Lohman, G. M., Sernadas, A., and Camps, R., editors, *Proc. of the 17th International Conference on Very Large Data Bases*, pages 525–535, Barcelona, Catalonia, Spain. Morgan Kaufmann.

Seetha, M. and Yu, P. S. (December 1990). Effectiveness of parallel joins. *IEEE, Transactions on Knowledge and Data Enginneerings*, 2(4):410–424.

Shatdal, A. and Naughton, J. F. (1995). Adaptive parallel aggregation algorithms. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 24(2):104–114.

Skillicorn, D. B., Hill, J. M. D., and McColl, W. F. (1997). Questions and Answers about BSP. *Scientific Programming*, 6(3):249–274.

Taniar, D., Jiang, Y., Liu, K., and Leung, C. (2000). Aggregate-join query processing in parallel database systems,. In *Proceedings of The Fourth International Conference/Exhibition on High Performance Computing in Asia-Pacific Region HPC-Asia2000*, volume 2, pages 824–829. IEEE Computer Society Press.

Taniar, D. and Rahayu, J. W. (2001). Parallel processing of 'groupby-before-join' queries in cluster architecture. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid, Brisbane, Qld, Australia*, pages 178–185. IEEE Computer Society.

Tsois, A. and Sellis, T. K. (2003). The generalized pre-grouping transformation: Aggregate-query optimization in the presence of dependencies. In *VLDB*, pages 644–655.

Valiant, L. G. (August 1990). A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111.

Wolf, J. L., Dias, D. M., Yu, P. S., and Turek, J. (1994). New algorithms for parallelizing relational database joins in the presence of data skew. *IEEE Transactions on Knowledge and Data Engineering*, 6(6):990–997.

Yan, W. P. and Larson, P.-. (1994). Performing group-by before join. In *Proceedings of the 10th IEEE International Conference on Data Engineering*, pages 89–100. IEEE Computer Society Press.