AN OPTIMAL EVALUATION OF GROUPBY-JOIN QUERIES IN DISTRIBUTED ARCHITECTURES

M. Al Hajj Hassan and M. Bamha
LIFO, Université d'Orléans, B.P. 6759, 45067 Orléans Cedex 2, France

Keywords: Parallel DataBase Management Systems (PDBMS), Parallel joins, Data skew, Join product skew, GroupBy-Join queries, BSP cost model.

Abstract: SQL queries involving join and group-by operations are fairly common in many decision support applications where 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 most significant drawbacks of the algorithms presented in the literature for treating such queries are that they are very sensitive to data skew and involve expansive communication and Input/Output costs in the evaluation of the join operation. In this paper, we present an algorithm that overcomes these drawbacks because it evaluates the "GroupBy-Join" query without the need of the direct evaluation of the costly join operation, thus reducing its Input/Output and communication costs. Furthermore, the performance of this algorithm is analyzed using the scalable and portable BSP (Bulk Synchronous Parallel) cost model which predicts a linear speedup even for highly skewed data.

1 INTRODUCTION

Aggregate functions used to summarize large volume of data based on a designated grouping are widely employed in applications such as: the decision support application, OnLine Analytical Processing (OLAP) and Data Warehouse (Taniar et al., 2000), because in such applications, aggregated and summarized data are more important than detailed records (Datta et al., 1998). Aggregate operations may be applied on the output of the join of multiple tables having potentially billions of records. These tables may rapidly grow every day especially in OLAP systems. Moreover, the output of these queries must be obtained in a reasonable processing time. For these reasons, parallel processing of such queries results in huge performance gain especially in PDBMS. However, the use of efficient parallel algorithm in PDBMS is fundamental in order to obtain an acceptable performance (Bamha and Hains, 2000; Seetha and Yu, 1990).
Several parallel algorithms for evaluating "GroupBy-Join" queries were presented in the literature (Shatdal and Naughton, 1995; Taniar et al., 2000), but these algorithms are inefficient due to their high communication cost because all the tuples of the relations are redistributed between processors. Some of these tuples may not even contribute in the result of the join operation.
In addition, these algorithms fully materialize the intermediate results of the join operations and the Input/Output cost is very high where it is reasonable to assume that the output relation cannot fit in the main memory of every processor, so it must be reread from disk in order to evaluate the aggregate function. Finally, these algorithms cannot solve the problem of data skew because data redistribution is generally based on hashing data into buckets and hashing is known to be inefficient in the presence of high frequencies (Bamha, 2005; Seetha and Yu, 1990).

In this paper, we present a new parallel algorithm used to evaluate the "GroupBy-Join" queries on Shared Nothing machines (a distributed architecture where each processor has its own memory and own disks), when the join attributes are different from the group-by attributes. Our main contribution is that, in this algorithm, we do not need to materialize the join operation as in the traditional algorithms where the join operation is evaluated first and then the group-by

Al Hajj Hassan M. and Bamha M. (2007).
AN OPTIMAL EVALUATION OF GROUPBY-JOIN QUERIES IN DISTRIBUTED ARCHITECTURES.
In *Proceedings of the Third International Conference on Web Information Systems and Technologies - Internet Technology*, pages 246-252
DOI: 10.5220/0001281302460252

and aggregate functions (Yan and Larson, 1994). It is also insensitive to data skew and its communication and Input/Output costs are reduced to a minimum.

In this algorithm, we partially evaluate the aggregate function before redistributing the tuples. This helps in reducing the cost of data redistribution. We use the histograms of both relations in order to find the tuples that participate in the result of the join operation. It is proved in (Bamha and Hains, 2005; Bamha and Hains, 2000), using the BSP model, that histogram management has a negligible cost when compared to the gain it provides in reducing the communication cost.
In traditional algorithms, all the tuples of the output of the join are redistributed using a hash function. In the contrary, in our algorithm we only redistribute the result of the semi-join of the histograms which are very small compared to the size of input relations. This helps in reducing the amount of data transferred over the network and therefore the communication cost. The performance of this algorithm is analyzed using the BSP cost model which predicts for our algorithm a linear speedup even for highly skewed data.

2 COMPUTATION OF "GROUPBY-JOIN" QUERIES

In DBMS, we can distinguish two types of "GroupBy-Join" queries. In the first type the join attributes and the group-by attributes are the same. In this case, it is preferable to carry out the group-by and aggregate functions first and then the join operation (Taniar et al., 2000), because this helps in reducing the size of the relations to be joined and consequently decreasing the communication cost and the query execution time. In the contrary, this can not be applied on the second type of queries, because the join attributes are different from the group-by attributes (see (Al Hajj Hassan and Bamha, 2007) for a long and detailed version of this paper). In this paper, we will focus on this type of "GroupBy-Join" queries where we present an algorithm which partially evaluates the aggregate functions before redistributing the tuples using histograms, thus reducing the communication cost as much as possible.

3 GROUPBY-JOIN QUERIES: A NEW APPROACH

In this section, we present a detailed description of a new parallel algorithm used to evaluate the "GroupBy-Join" queries when the group-by attributes are different from the join attributes. We assume that the relation R (resp. S) is evenly partitioned among processors by horizontal fragmentation such that $|R_i| \simeq \frac{|R|}{p}$ $(i = 1, ..., p)$ 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 one attribute y of R and another attribute z of S. We also assume that the aggregate function is applied on the values of the attribute u of S.

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, a sub-relation of T,
- $Hist^w(T)$ denotes the histogram[1] 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 while $Hist_i^w(T)$ 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 while $Hist^w(T_i)(v)$ is its frequency in sub-relation T_i,
- $AGGR_{f,u}^w(T)$ [2] is the result of applying the aggregate function f on the values of the attribute u of every group of tuples of T having identical values of the group-by attributes w. $AGGR_{f,u}^w(T)$ is formed of a list of tuples (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),
- $AGGR_{f,u}^w(T_i)$ denotes the result of applying the aggregate function on the attribute u of relation T_i while $AGGR_{f,u,i}^w(T)$ is processor i's fragment of the result of applying the aggregate function on T,
- $AGGR_{f,u}^w(T)(v)$ (resp.$AGGR_{f,u}^w(T_i)(v)$) is the result f_v of the aggregate function of the group of tuples having value v for the group-by attribute w in relation T (resp. 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.

[1] 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.

[2] $AGGR_{f,u}^w(T)$ is implemented as a B-tree.

The algorithm proceeds in six phases. To study the cost of each phase we use the scalable and portable *Bulk-Synchronous Parallel* (BSP) cost model which is a programming model introduced by L. Valiant (Valiant, 1990). In this algorithm, 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 (Al Hajj Hassan and Bamha, 2007).

Phase 1: Creating local histograms

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

In addition, the local fragments $AGGR^{x,z}_{f,u}(S_i)(i = 1,...,p)$ of blocks S_i are also created 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) in time $c_{i/o} * \max_{i=1,...,p}|S_i|$.

In this phase we also compute the frequency of each value of the attribute x in $Hist^{x,y}(R_i)$ and $AGGR^{x,z}_{f,u}(S_i)$ needed in phase 2. So while creating $Hist^{x,y}(R_i)$ (resp. $AGGR^{x,z}_{f,u}(S_i)$), we also create on the fly their local histograms $Hist^{'x}(R_i)$ (resp. $Hist^{'x}(S_i)$) with respect to x, i.e. $Hist^{'x}(R_i)$ (resp. $Hist^{'x}(S_i)$) holds the frequency of each value of the attribute x in $Hist^{x,y}(R_i)$ (resp. $AGGR^{x,z}_{f,u}(S_i)$) where we count tuples having the same values of the attributes (x,y) only once.

We use the following algorithm to create $Hist^{'x}(R_i)$ and a similar one is used to create $Hist^{'x}(S_i)$.

Par (on each node in parallel) $i = 1,...,p$
  $Hist^{'x}(R_i)$ = NULL [3]
  For every tuple t **that will be inserted or used to modify** $Hist^{x,y}(R_i)$ **do**
   If $Hist^{x,y}(R_i)(t.x,t.y)$ = NULL **Then** [4]
    $freq_1 = Hist^{'x}(R_i)(t.x)$
    If $freq_1 \neq$ NULL **Then**
     Increment the frequency of $t.x$ in $Hist^{'x}(R_i)$
    Else
     Insert a new tuple $(t.x, 1)$ into $Hist^{'x}(R_i)$
    EndIf
   EndIf
  EndFor
EndPar

In principle, this phase costs:
$Time_{phase1} = O\big(c_{i/o} * \max_{i=1,...,p}(|R_i| + |S_i|)\big)$.

Phase 2: Local semi-joins computation

In order to minimize the communication cost, only tuples of $Hist^{x,y}(R)$ and $AGGR^{x,z}_{f,u}(S)$ that will be present in the join result are redistributed. To this end, we compute $\overline{Hist}^{x,y}(R_i) = Hist^{x,y}(R_i) \ltimes AGGR^{x,z}_{f,u}(S)$ and $\overline{AGGR}^{x,z}_{f,u}(S_i) = AGGR^{x,z}_{f,u}(S_i) \ltimes Hist^{x,y}(R)$ using proposition 2 presented in (Bamha and Hains, 2005), where we apply the hash function on the tuples of $Hist^{'x}(R_i)$ and $Hist^{'x}(S_i)$. In fact the number of tuples of $Hist^{'x}(R_i)$ and that of $Hist^{x}(R_i)$ are equal, what differs is only the value of their frequency attribute, so $|Hist^{'x}(R_i)| = |Hist^{x}(R_i)|$ (this also applies to $Hist^{'x}(S_i)$ and $Hist^{x}(S_i)$). Hence the cost of this phase is (Bamha and Hains, 2005):
$Time_{phase2} =$
$O\Big(\max_{i=1,...,p}||Hist^{x,y}(R_i)|| + \max_{i=1,...,p}||AGGR^{x,z}_{f,u}(S_i)||$
$+min\big(g*|Hist^{x}(R)|+||Hist^{x}(R)||, g*\frac{|R|}{p}+\frac{||R||}{p}\big)$
$+min\big(g*|Hist^{x}(S)|+||Hist^{x}(S)||, g*\frac{|S|}{p}+\frac{||S||}{p}\big)+l\Big)$.
where g is the BSP communication parameter and l is the cost of a barrier of synchronization.

During semi-join computation, we store for each value $d \in Hist^{'x}(R) \cap Hist^{'x}(S)$ an extra information called $index(d) \in \{1,2,3\}$ which will allow us to decide if, for a given value d, the frequencies of tuples of $Hist^{x,y}(R)$ and $AGGR^{x,z}_{f,u}(S)$ having the value d are greater (resp. lesser) than a threshold frequency f_0. It also permits us to choose dynamically the probe and the build relation for each value d of the join attribute. This choice reduces the global redistribution cost to a minimum.

In the rest of this paper, we use $f_0 = p * log(p)$ as the threshold frequency (Bamha and Hains, 2000).

For a given value $d \in Hist^{'x}(R) \cap Hist^{'x}(S)$ [5], the value $index(d) = 3$, means that $Hist^{'x}(R)(d) < f_0$ and $Hist^{'x}(S)(d) < f_0$, while $index(d) = 2$, means that $Hist^{'x}(S)(d) \geq f_0$ and $Hist^{'}x(S)(d) > Hist^{'x}(R)(d)$ and $index(d) = 1$, means that $Hist^{'x}(R)(d) \geq f_0$ and $Hist^{'x}(R)(d) \geq Hist^{'x}(S)(d)$. Note that unlike the algorithms presented in (Shatdal and Naughton, 1995; Taniar et al., 2000) where both relations R and S are redistributed, we will only redistribute $\overline{Hist}^{x,y}(R_i)$ and $\overline{AGGR}^{x,z}_{f,u}$ to find the final result. This reduces the communication costs to a minimum.

At the end of this phase, we will divide $\overline{Hist}^{x,y}(R_i)$ and $\overline{AGGR}^{x,z}_{f,u}(S_i)$ on each processor i into three sub-histograms such that: $\overline{Hist}^{x,y}(R_i) = \bigcup_{j=1}^{3} \overline{Hist}^{(j)x,y}(R_i)$ and

[5] The intersection of $Hist^{'x}(R)$ and $Hist^{'x}(S)$ is found while computing the semi-joins (c.f proposition 2 presented in (Bamha and Hains, 2005))

$\overline{AGGR}^{x,z}_{f,u}(S_i) = \bigcup_{j=1}^{3} \overline{AGGR}^{(j)x,z}_{f,u}(S_i)$ where all the tuples of $\overline{Hist}^{(1)x,y}(R_i)$ (resp. $\overline{AGGR}^{(1)x,z}_{f,u}(S_i)$) are associated to values d such that $index(d) = 1$ (resp. $index(d) = 2$), while that of $\overline{Hist}^{(2)x,y}(R_i)$ (resp. $\overline{AGGR}^{(2)x,z}_{f,u}(S_i)$) are associated to values d such that $index(d) = 2$ (resp. $index(d) = 1$), and all the tuples of $\overline{Hist}^{(3)x,y}(R_i)$ and $\overline{AGGR}^{(3)x,z}_{f,u}(S_i)$ are associated to values d such that $index(d) = 3$.

Phase 3: Creating the communication templates

The tuples of relations $\overline{Hist}^{(3)x,y}(R_i)$ and $\overline{AGGR}^{(3)x,z}_{f,u}(S_i)$ (have very low frequencies for the join attribute) have no effect neither on Attribute Value Skew (AVS) nor on Join Product Skew (JPS) so they are redistributed using a hash function. However the tuples of $\overline{Hist}^{(1)x,y}(R_i)$ and $\overline{AGGR}^{(1)x,z}_{f,u}(S_i)$ are associated to high frequencies for the join attribute so they have an important effect on AVS and JPS. So we will use an appropriate redistribution algorithm in order to efficiently avoid both AVS and JPS (Bamha and Hains, 2000).

3.a To this end, we partition the histogram $Hist'^{x}(R \bowtie S)$ (which is simply the intersection of $Hist'^{x}(R)$ and $Hist'^{x}(S)$) into two sub-histograms: $Hist^{(1,2)'x}(R \bowtie S)$ and $Hist^{(3)'x}(R \bowtie S)$ where the values $d \in Hist^{(1,2)'x}(R \bowtie S)$ are associated to high frequencies of the join attribute (i.e. $index(d) = 1$ or $index(d) = 2$) while the values $d \in Hist^{(3)'x}(R \bowtie S)$ are associated to low frequencies (i.e. $index(d) = 3$). This partition step is performed in parallel, on each processor i, by a local traversal of the histogram $Hist'^{x}_i(R \bowtie S)$ in time: $Time_{3.a} = O\big(\max_{i=1,...,p} ||Hist'^{x}_i(R \bowtie S)||\big)$.

3.b Communication templates for high frequencies:

We first create a communication template: the list of messages which constitutes the relations' redistribution. This step is performed jointly by all processors, each one not necessarily computing the list of its own messages, so as to balance the overall process.

So each processor i computes a set of necessary messages relating to the values d it owns in $Hist^{(1,2)'x}_i(R \bowtie S)$. The communication template of $\overline{Hist}^{(1)x,y}(R)$ is derived by applying the following algorithm. We also apply the same algorithm to compute the communication template of $\overline{AGGR}^{(1)x,z}_{f,u}(S)$, but we replace $Hist'^{x}(R)$ by $Hist'^{x}(S)$.

---

if $\big(Hist'^{x}(R)(d) mod(p) = 0\big)$ **then**
each processor j will hold a block of size
$block_j(d) = \dfrac{Hist'^{x}(R)(d)}{p}$ of tuples of value d.
else
begin
Pick a random value j_0 between 0 and $(p-1)$
if (processor's index j is between j_0 and $j_0 + \big(Hist'^{x}(R)(d) \bmod p\big)$) **then**
processor of index j will hold a block
of size: $block_j(d) = \lfloor \dfrac{Hist'^{x}(R)(d)}{p} \rfloor + 1$
else
processor of index j will hold a block
of size: $block_j(d) = \lfloor \dfrac{Hist'^{x}(R)(d)}{p} \rfloor$
end.

In the above algorithm, $\lfloor x \rfloor$ is the largest integral value not greater than x and $block_j(d)$ is the number of tuples of value d that processor j should own after redistribution of the fragments T_i of relation T.

The absolute value of $Rest_j(d) = Hist_j(T)(d) - block_j(d)$ determines the number of tuples of value d that processor j must send (if $Rest_j(d) > 0$) or receive (if $Rest_j(d) < 0$).

For $d \in Hist^{(1,2)'x}_i(R \bowtie S)$, processor i owns a description of the layout of tuples of value d over the network. It may therefore determine the number of tuples of value d which every processor must send/receive. This information constitutes the communication template. Only those j for which $Rest_j(d) > 0$ (resp. $Rest_j(d) < 0$) send (resp. receive) tuples of value of d. This step is thus completed in time: $Time_{3.b} = O\big(||Hist^{(1,2)'x}(R \bowtie S)||\big)$.

The tuples associated to low frequencies (i.e. tuples having $d \in Hist^{(3)'x}_i(R \bowtie S)$) have no effect neither on the AVS nor the JPS. These tuples are simply mapped to processors using a hash function.

Thus the cost of creating the communication templates is: $Time_{phase3} = O\big(\max_{i=1,...,p} ||Hist'^{x}_i(R \bowtie S)|| + ||Hist^{(1,2)'x}(R \bowtie S)||\big)$.

Phase 4: Data redistribution

4.a *Redistribution of tuples having* $d \in Hist^{(1,2)'x}_i(R \bowtie S)$: Every processor i holds, for every one of its local $d \in Hist^{(1,2)'x}_i(R \bowtie S)$, the non-zero communication volumes it prescribes as a part of communication template: $Rest_j(d) \neq 0$ for $j = 1, ..., p$. This information will take the form of *sending orders* sent to their target processor in a first superstep, followed then by the actual redistribution superstep where processors obey all orders they have received.

Each processor i first splits the processors indices j into two groups: those for which $Rest_j(d) > 0$ and those for which $Rest_j(d) < 0$. This is done by a sequential traversal of the $Rest_{..}(d)$ array.

Let α (resp. β) be the number of j's where $Rest_j(d)$ is positive (resp. negative) and $Proc(k)_{k=1,...,\alpha+\beta}$ the array of processor indices for which $Rest_j(d) \neq 0$ in the manner that: $Rest_{proc(j)}(d) > 0$ *for* $j = 1, ..., \alpha$ and $Rest_{proc(j)}(d) < 0$ *for* $j = 1+\alpha, ..., \beta$.
A sequential traversal of $Proc(k)_{k=1,...,\alpha+\beta}$ determines the number of tuples that each processor j will send. The sending orders concerning attribute value d are computed using the following procedure whose maximal complexity is $O\big(||Hist^{(1,2)'x}(R \bowtie S)||\big)$ because for a given d, no more than $(p-1)$ processors can send data and each processor i is in charge of redistribution of tuples having $d \in Hist_i^{(1,2)'x}(R \bowtie S)$.

```
i := 1;      j := α + 1;
while (i ≤ α) do
  begin
   n_tuples = min(Rest_proc(i)(d), −Rest_proc(j)(d));
   order_to_send(Proc(i),Proc(j),d,n-tuples);
   Rest_proc(i)(d) := Rest_proc(i)(d) - n_tuples;
   Rest_proc(j)(d) := Rest_proc(j)(d) + n_tuples;
   if Rest_proc(i)(d) = 0 then i := i + 1; endif
   if Rest_proc(j)(d) = 0 then j := j + 1; endif
  end.
```

For each processor i and $d \in Hist_i^{(1,2)'x}(R \bowtie S)$, all the order_to_send$(j, i, ...)$ are sent to processor j when $j \neq i$ in time $O\big(g * |Hist^{(1,2)'x}(R \bowtie S)| + l\big)$. Thus, this step costs: $Time_{4.a} = O\big(g*|Hist^{(1,2)'x}(R \bowtie S)|+||Hist^{(1,2)'x}(R \bowtie S)||+l\big)$.

4.b Tuples of $\overline{Hist}^{(3)x,y}(R_i)$ and $\overline{AGGR}_{f,u}^{(3)x,z}(S_i)$ (i.e. tuples having $d \in Hist_i^{(3)'x}(R \bowtie S)$) are associated to low frequencies, they have no effect neither on the AVS nor the JPS. These relations are redistributed using a hash function.
At the end of steps $4.a$ and $4.b$, each processor i, has local knowledge of how the tuples of semi-joins $\overline{Hist}^{x,y}(R_i)$ and $\overline{AGGR}_{f,u}^{x,z}(S_i)$ will be redistributed. Redistribution is then performed in time:

$$Time_{4.b} = O\Big(g * \big(|\overline{Hist}^{x,y}(R_i)| + |\overline{AGGR}_{f,u}^{x,z}(S_i)|\big) + l\Big).$$

Thus the total cost of the redistribution phase is:

$$Time_{phase4} = O\Big(g * \max_{i=1,...,p}(|\overline{Hist}^{x,y}(R_i)| + |\overline{AGGR}_{f,u}^{x,z}(S_i)| + |Hist^{(1,2)'x}(R \bowtie S)|) + ||Hist^{(1,2)'x}(R \bowtie S)|| + l\Big)$$

We mention that we only redistribute the tuples of the semi-joins $\overline{Hist}^{x,y}(R_i)$ and $\overline{AGGR}_{f,u}^{x,z}(S_i)$ where $|\overline{Hist}^{x,y}(R_i)|$ and $|\overline{AGGR}_{f,u}^{x,z}(S_i)|$ are generally very small compared to $|R_i|$ and $|S_i|$. In addition $|Hist^{'x}(R \bowtie S)|$ is generally very small compared to $|Hist^{x,y}(R)|$ and $|AGGR_{f,u}^{x,z}(S)|$. Thus we reduce the communication cost to a minimum.

Phase 5: local computation of the aggregate function

At this step, every processor has partitions of $\overline{Hist}^{x,y}(R)$ and $\overline{AGGR}_{f,u}^{x,z}(S)$. Using equation 2 in (Bamha, 2005), we can deduce that the tuples of $\overline{Hist}^{(1)x,y}(R_i)$, $\overline{Hist}^{(2)x,y}(R_i)$, $\overline{Hist}^{(3)x,y}(R_i)$ can be joined with the tuples of $\overline{AGGR}_{f,u}^{(2)x,z}(S_i)$, $\overline{AGGR}_{f,u}^{(1)x,z}(S_i)$, $\overline{AGGR}_{f,u}^{(3)x,z}(S_i)$ respectively. But the frequencies of tuples of $\overline{Hist}^{(1)x,y}(R_i)$ and $\overline{AGGR}_{f,u}^{(1)x,z}(S_i)$ are by definition greater than the corresponding (matching) tuples in $\overline{Hist}^{(2)x,y}(R_i)$ and $\overline{AGGR}_{f,u}^{(2)x,z}(S_i)$ respectively. So we will choose $\overline{Hist}^{(1)x,y}(R_i)$ and $\overline{AGGR}_{f,u}^{(1)x,z}(S_i)$ as the *build* relations and $\overline{Hist}^{(2)x,y}(R_i)$ and $\overline{AGGR}_{f,u}^{(2)x,z}(S_i)$ as *probe* relations. Hence, we need to duplicate the probe relations to all processors in time:

$$Time_{phase5.a} = O\Big(g * \big(|\overline{Hist}^{(2)x,y}(R)| + |\overline{AGGR}_{f,u}^{(2)x,z}(S)|\big) + l\Big).$$

Now, using the following algorithm, we are able to compute the local aggregate function on every processor without the necessity to fully materialize the intermediate results of the join operation.

In this algorithm, we create on each processor i, the relation $AGGR_{f,u}^{y,z}((R \bowtie S)_i)$ that holds the local results of applying the aggregate function on every group of tuples having the same value of the couple of attributes (y, z). $AGGR_{f,u}^{y,z}((R \bowtie S)_i)$ has the form (y, z, v) where y and z are the group-by attributes and v is the result of the aggregate function.

```
(1)Par (on each node in parallel) i = 1, ..., p
(2)  AGGR^{y,z}_{f,u}((R ⋈ S)_i) = NULL; [6]
(3)   For every tuple t of relation Hist^{(1)x,y}(R_i) do
(4)    For every entry v1 = AGGR^{(2)x,z}_{f,u}(S_i)(t.x, z) do
(5)     v2 = AGGR^{y,z}_{f,u}((R ⋈ S)_i)(t.y, z);
(6)     If v2 ≠ NULL Then
(7)      Update AGGR^{y,z}_{f,u}((R ⋈ S)_i)(t.y, z)=F(v1, v2)
         where F() is the aggregate function;
(8)     Else
(9)      Insert a new tuple (t.y, z, v1) into the
         histogram AGGR^{y,z}_{f,u}((R ⋈ S)_i);
(10)    EndIf
(11)   EndFor
(12)  EndFor
(13)  Repeat steps (3)...(12) but replace
      Hist^{(1)x,y}(R_i) in (3) by AGGR^{(1)x,z}_{f,u}(S_i)
      and AGGR^{(2)x,z}_{f,u}(S_i)(t.x, z) in (4) by
      Hist^{(2)x,y}(R_i)(t.x, y);
(14)  Repeat steps (3)...(12) but replace
      Hist^{(1)x,y}(R_i) in (3) by Hist^{(3)x,y}(R_i)
      and AGGR^{(2)x,z}_{f,u}(S_i)(t.x, z) in (4) by
      AGGR^{(3)x,z}_{f,u}(S_i)(t.x, z);
(15)EndPar
```

The cost of applying this algorithm is: $Time_{phase5.b} =$

$c_{i/o} * O\Big(\max_{i=1,\ldots,p}\big(|\overline{Hist}^{(1)x,y}(R_i) \bowtie \overline{AGGR}^{(2)x,z}_{f,u}(S)| + |\overline{Hist}^{(2)x,y}(R) \bowtie \overline{AGGR}^{(1)x,z}_{f,u}(S_i)| + |\overline{Hist}^{(3)x,y}(R_i) \bowtie \overline{AGGR}^{(3)x,z}_{f,u}(S_i)|\big)\Big)$

So the total cost of this phase is simply the sum of $Time_{phase5.a}$ and $Time_{phase5.b}$.

Phase 6: global computation of the aggregate function

In this phase, a global application of the aggregate function is carried out. For this purpose, every processor redistributes the local aggregation results, $AGGR^{y,z}_{f,u}((R \bowtie S)_i)$, using a common hash function whose input attributes are y and z. After hashing, every processor applies the aggregate function on the received messages in order to compute the global result $AGGR^{y,z}_{f,u}(R \bowtie S)$. The time of this step is:

$Time_{phase6} = O\Big(min\big(g * |AGGR^{y,z}_{f,u}(R \bowtie S)| + ||AGGR^{y,z}_{f,u}(R \bowtie S)||, g * \frac{|R \bowtie S|}{p} + \frac{||R \bowtie S||}{p}\big) + l\Big)$

where we apply proposition 1 in (Bamha and Hains, 2005)) to redistribute $AGGR^{y,z}_{f,u}((R \bowtie S)_i)$.

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

$Time_{total} = O\Big(c_{i/o} * \max_{i=1,\ldots,p}(|R_i| + |S_i|)$
$+min\big(g * |Hist^x(R)| + ||Hist^x(R)||, g * \frac{|R|}{p} + \frac{||R||}{p}\big)$
$+min\big(g * |Hist^x(S)| + ||Hist^x(S)||, g * \frac{|S|}{p} + \frac{||S||}{p}\big)$
$+g * \max_{i=1,\ldots,p}\big(|\overline{Hist}^{x,y}(R_i)| + |\overline{AGGR}^{x,z}_{f,u}(S_i)|$
$+|Hist^{(1,2)'x}(R \bowtie S)|\big) + ||Hist^{(1,2)'x}(R \bowtie S)||$
$+g * \big(|\overline{Hist}^{(2)x,y}(R)| + |\overline{AGGR}^{(2)x,z}_{f,u}(S)|\big)$
$+c_{i/o} * \max_{i=1,\ldots,p}\big(|\overline{Hist}^{(1)x,y}(R_i) \bowtie \overline{AGGR}^{(2)x,z}_{f,u}(S)|$
$+|\overline{Hist}^{(2)x,y}(R) \bowtie \overline{AGGR}^{(1)x,z}_{f,u}(S_i)|$
$+|\overline{Hist}^{(3)x,y}(R_i) \bowtie \overline{AGGR}^{(3)x,z}_{f,u}(S_i)|\big)$
$+min\big(g * |AGGR^{y,z}_{f,u}(R \bowtie S)| + ||AGGR^{y,z}_{f,u}(R \bowtie S)||,$
$g * \frac{|R \bowtie S|}{p} + \frac{||R \bowtie S||}{p}\big) + \max_{i=1,\ldots,p}||Hist^{x,y}(R_i)||$
$+ \max_{i=1,\ldots,p}||AGGR^{x,z}_{f,u}(S_i)|| + l\Big).$

Remark 1

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, the evaluation of the "GroupBy-Join" queries when the join attributes are different from the group-by attributes has an optimal asymptotic complexity when:

$max\ |\overline{Hist}^{(2)x,y}(R)|, |\overline{AGGR}^{(2)x,z}_{f,u}(S)|, |Hist^{(1,2)'x}(R \bowtie S)|$
$\leq c_{i/o} * max(\frac{|R|}{p}, \frac{|S|}{p}, \frac{|R \bowtie S|}{p})$,

this is due to the fact that the local join results have almost the same size and all the terms in $Time_{total}$ are bounded by those of $bound_{inf_p}$. This inequality holds if we choose a threshold frequency f_0 greater than p (which is the case for our threshold frequency $f_0 = p * log(p)$).

4 CONCLUSION

In this paper, we presented a parallel algorithm used to compute "GroupBy-Join" queries in a distributed architecture when the group-by attributes and the join attributes are not the same. This algorithm can be used efficiently to reduce the execution time of the query, because we do not materialize the costly join operation which is a necessary step in all the other algorithms presented in the literature that treat this type of queries, thus reducing the Input/Output cost. It also helps us to balance the load of all the processors even in the presence of AVS and to avoid the JPS which may result from computing the intermediate join results.

In addition, the communication cost is reduced to the minimum owing to the fact that only histograms and the results of semi-joins are redistributed across the network where their size is very small compared to the size of input relations.

The performance of this algorithm was analyzed using the BSP cost model which predicts an asymptotic optimal complexity for our algorithm even for highly skewed data.

In our future work, we will implement this algorithm and extend it to a GRID environment.

REFERENCES

Al Hajj Hassan, M. and Bamha, M. (2007). An optimal evaluation of groupby-join queries in distributed architectures. Research Report RR-2007-01, LIFO, Université d'Orléans, France.

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, LNCS 3588, pages 616–625.

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*, LNCS 1873, London, United Kingdom.

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*, LNCS 3515, pages 755–763.

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.

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. *ACM SIGMOD Record*, 24(2):104–114.

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.

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

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