UNIVERSITE D'ORLEANS
Faculté des Sciences
LIFO

Laboratoire d'Informatique Fondamentale d'Orléans
4, rue Léonard de Vinci, BP 6759
F-45067 Orléans Cedex 2
FRANCE

Rapport de Recherche

www : http://www.univ-orleans.fr/SCIENCES/LIFO/

An optimal and skew-insensitive join algorithm for Shared Nothing machines

M. Bamha
LIFO, Université d'Orléans.
bamha@lifo.univ-orleans.fr

Rapport N^o **2000-05**

An optimal and skew-insensitive join algorithm for Shared Nothing machines

M. Bamha

LIFO, Université d'Orléans,
BP 6759, 45067 Orléans cedex 2, France.
bamha@lifo.univ-orleans.fr

Abstract. Joins constitute a computation-intensive part of database request and their parallelization is necessary for dealing with very large data sets. However, effectiveness of parallel joins depends on the ability to evenly divide load among processors. Research has shown that the join operation is parallelizable with near-linear speed-up on shared nothing machines only under ideal balancing conditions. Data skew can have a disastrous effect on performance. Although many skew-handling algorithms have been proposed they remain generally inefficient in the case of multi-joins due to join product skew, costly and unnecessary redistribution and communication costs. In this paper, we introduce a new parallel join algorithm with *optimal complexity* and *deterministic perfect balancing.* Its predictably low join-product and attribute-value skews makes it suitable for repeated use in multi-join operations. Its tradeoff between balancing overhead and speedup is analyzed using the scalable and portable BSP [1] cost model which predicts a negligible join product skew and a near-linear speed-up. The algorithm presented here, improves our fa_join and sfa_join algorithms by reducing their communication and synchronisation cost to a minimum while offering the same load balancing properties.

Key words: *PDBMS* [2], Intra-transaction parallelism, Parallel joins, Multi-joins, Data-skew, Join-product skew, Dynamic load balancing.

1 Introduction

The appeal of parallel processing becomes very strong in applications which require ever higher performance and particularly in applications such as : data-warehousing, decision support and OLAP [3]. Parallelism can greatly increase processing power in such applications [DMT98, BBH99]. However parallelism can only maintain acceptable performance through efficient algorithms realizing complex queries on dynamic, irregular and distributed data. Such algorithms must be designed to fully exploit the processing power of multi-processor machines and the ability to evenly divide load among processors while minimizing local computation and communication costs inherent to multi-processors machines.

Research has shown that the join operation is parallelizable with near-linear speed-up on shared nothing machines only under ideal balancing conditions. Data skew can have a disastrous effect on performance [BH00a, BH98, SY90, KO90, HL91, WDYT94, HTY95, DNSS92, PI96, HK95, MMSY94, Mär98].

Many algorithms have been proposed to handle data skew for a simple join operation, but little is known for the case of complex queries leading to multi-joins [LOT94, DNSS92, HL91, PI96]. In particular, the performance of PDBMS has generally been estimated on queries involving one or two

1. BSP : Bulk-synchronous parallelism.
2. PDBMS : Parallel Database Management Systems.
3. OLAP : On-Line Analytical Processing.

join operations only [WFA95]. However the problem of data skew is more acute with multi-joins because the imbalance of intermediate results is unknown during static query optimization [LOT94].

Join algorithms described in previous works induce costly and unnecessary communications. In this paper we show that the communications involved in computing the join $R \bowtie S$ of two relations R and S can be reduced to a minimum by :

- an efficient computation of the semi-joins $R \ltimes S$ and $S \ltimes R$. This reduces the redistribution cost to only the tuples of relations R and S which will effectively be present in the join result.
- a dynamic choice of the build [4] and probe [5] relation for each bucket (sub-relation) of the two relations allowing to reduce the communications costs while avoiding attribute and join product skews.

Our present contributions to parallel relational query processing are :

1. A new parallel join algorithm called *Osfa_join* (Optimal symmetric frequency adaptive join algorithm) for shared nothing machines. This algorithm has optimal complexity, near-perfect balancing properties and supports flexible control of communications induced by intra-transaction parallelism. The Osfa_join algorithm is based on an improved version of the redistribution algorithm of *fa-join* [BH98, BH00a] which efficiently avoids the problem of attribute value- and join product skews while reducing the communication and synchronisation costs to a minimum.
2. An analysis of Osfa_join using the scalable and portable BSP cost model. It predicts a negligible join product skew and a near linear speed-up for our algorithm, independently of the data and of the (shared nothing) architecture's bandwidth, latency and number of processors.

The organization of this paper is as follows : We first recall the notions of PDBMS, joins operations and load balancing in section 2. Section 3 introduces the scalable and portable BSP cost model. Section 4 presents a new data redistribution approach for parallel join operations. Section 5 presents the Osfa_join algorithm and its complexity. We then conclude this work in section 6.

2 PDBMS : join operations and load balancing

2.1 PDBMS, join operations and data skew

Join is an expensive and frequently used operation whose parallelization is highly desirable. The *θ-join* of two relations R and S on attribute A of R and attribute B of S (A and B of same domain) is the relation T, written $R \bowtie S$, obtained by concatenating the pairs of tuples from R and S for which $R.A\ \theta\ S.B$ where $\theta \in \{=\ ,\neq\ ,<\ ,>\ ,\leq\ ,\geq\}$. When θ is $=$, the operation is called *equi-join*. The *semi-join* of S by R is the relation $S \ltimes R$ composed of the tuples of S which occur in the join of R and S. Semi-join reduces the size of relations to be joined and $R \bowtie S = R \bowtie (S \ltimes R) = (R \ltimes S) \bowtie (S \ltimes R)$.

In PDBMS, relations are generally partitioned among processors by *horizontal fragmentation* with the values of a chosen attribute.Three methods are used [DG92]: hash partitioning, block-partitioning (also called range partitioning) and cyclic (also called round-robin) partitioning. Relation indices are also partitioned and partitioning is expected to be balanced. Fragmentation is physical for shared nothing (SN) machines and logical for shared disk (SD) machines [RS95].

Parallel join usually proceeds in two phases: a redistribution phase by join attribute hashing and then sequential join of local fragments. Many such algorithms have been proposed. The principal ones are: *Sort-merge join*, *Simple-hash join*, *Grace-hash join* and *Hybrid-hash join* [SD89]. All of them (called hashing algorithms) are based on hashing functions which redistribute relations so that tuples having the same attribute value are forwarded to the same node. Local joins are then computed and their union is the output relation. Their major disadvantage is to be vulnerable to both *attribute value skew* (imbalance of the output of the first phase) and *join product skew* (imbalance of the output of local joins) [SY90, PI96, LOT94]. The former affects immediate performance and the latter affects the efficiency of output or pipelined operations in the case of a multi-join.

4. Build relation : the relation used to create the hash table.
5. Probe relation : the relation used to probe the hash table.

2.2 Load balancing in parallel join operations

The authors of [PI96] have identified the two best proposed solutions in conventional- and sampling-based parallel join algorithms. All are based on the notion of parallel hashing according to the value of the join attribute.

In the first category, the *extended adaptive load balancing parallel hash join* of [HL91] sends all tuples with the same attribute value to the same node. As a result, the algorithm may fail to balance the load for attribute-value distributions where a few values have a large weight. Moreover, the algorithm ignores the attribute value skew (AVS) of the *probe* relation and the join product skew (JPS). In other words, both the intermediate result of hashing (before local join computations) and the output relations (a dynamic and data-dependent subset of this intermediate result) can be unbalanced over the network. In the second category of algorithms, *virtual process partitioning* [DNSS92] improves on the previous category of algorithms but fails to handle AVS in the probe relation once the build relation is chosen. It also ignores JPS.

To improve on these methods, the authors of [PI96] have then introduced an algorithm which minimizes the *expected* AVS. It precomputes histograms of the expected attribute-value distribution (i.e. statistically predicts the eventual result of hashing). However, it fails to correct JPS as follows. The method is based as before on hashing the set of attribute values into buckets, each one being held by a single node. The algorithm is therefore sensitive to the same effect as earlier methods: if a few attribute value account for most of the output relation, then most records of the *output* relation may end-up on the same node. We conclude that all existing methods are sensitive to imbalance when applied multiple times because of JPS.

To address this problem, we introduced in [BH98, BH00a] a deterministic data-redistribution algorithm with near-perfect balancing properties. It is adaptable to θ-join and efficient for multi-join. It dynamically computes *exact* frequency histograms to avoid *join product skew* (JPS). A scalable and portable cost analysis was made with the BSP model, leading to general predictions about the effect of relation histograms on performance. The analysis suggests a hybrid *frequency-adaptive* algorithm (*fa_join* algorithm) [BH98, BH00a], dynamically combining histogram-based balancing with standard hashing methods. The *fa_join* algorithm avoids the slowdown usually caused by attribute value skew (AVS) and the imbalance of the size of local joins processed by the standard algorithms. It avoids AVS and JPS at the cost of extra processing time. We analyzed this overhead both theoretically and experimentally and concluded that it does not penalize overall performance, even in the absence of skew.

The *fa_join* algorithm was thus designed to efficiently avoid the problem of AVS and JPS. However, its performance is less than ideal when computing the join of highly skewed relations because of unnecessary redistribution and communication costs. We introduce here a new parallel algorithm called *Osfa_join* (Optimal symmetric frequency adaptive join algorithm) to perform such joins. The algorithm Osfa_join improves on the algorithm Sfa_join algorithm introduced in [BH00b] by its optimal complexity and in the semi-joins computation. Its predictably low join-product and attribute-value skew make it suitable for repeated use in multi-join operations. Its performance is analyzed using the scalable BSP cost model which predicts for it a near-linear speedup.

3 The BSP cost model

Bulk-Synchronous Parallelism (BSP) is a parallel programming model introduced by Valiant [Val90] to offer a high degree of abstraction like PRAM models and yet allow portable and predictable performance on a wide variety of architectures. 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 and the time g for collectively delivering a 1-relation (communication phase where every processor receives/sends at most one word). The network is assumed to deliver an h-relation in time $g * h$ for any arity h.

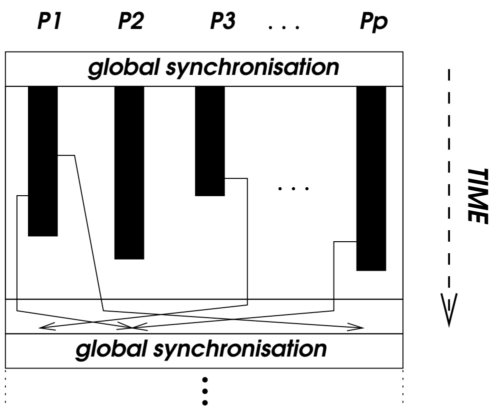

FIG. 1 – *A BSP superstep.*

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 its local data (only) 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 $w_i^{(s)}$ = 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).

4 Data redistribution : A new approach

In this section, we present a variant of the algorithm *sfa_join*[BH00b] called Osfa_join (*Optimal symmetric frequency adaptive join algorithm*) with an optimal complexity. The major difference between the two algorithms lies in the manner of computing the semi-joins. We point out that, for the computation of the semi-joins, the algorithm sfa_join broadcasts the histograms $Hist_i(R \bowtie S)_{i=1..,p}$ to all processors. Thus, each processor has a local access to the whole the histogram $Hist(R \bowtie S)$ [6] to compute local semi-joins. In the algorithm *Osfa_join* the semi-joins computation is carried out in an optimal way without this stage from diffusion.

Before carrying out the description of the various stages of the algorithm Osfa_join, we present some results related to histograms and semi-joins computations which we will use thereafter.

6. We recall that the histogram $Hist(R \bowtie S)$ is of size generally very small compared to the sizes of the histograms $Hist(R)$ and $Hist(S)$.

4.1 Histograms and semi-joins processing: Some preliminary results

We first assume that 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 every processor, i.e. $|R_i| \simeq \frac{|R|}{p}$ where p is the number of processors. In the rest of this paper we use the following terminology for each relation $T \in \{R,S\}$:

- T_i denotes the fragment of relation T placed on processor i,
- $Hist(T)$ denotes the histogram of relation T with respect to the join attribute value, i.e. a list of pairs (v,n_v) where $n_v \neq 0$ is the number of tuples having the value v for the join attribute. The histogram is often much smaller and never larger than the relation it describes,
- $Hist(T_i)$ denotes the histogram of fragment T_i
- $Hist_i(T)$ is processor i's fragment of the histogram of T,
- $Hist(T)(d)$ is the frequency (n_d) of value d in relation T,
- $Hist(T_i)(d)$ is the frequency of value d 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.

Proposition 4.1. *Let $Hist(R_i)_{i=1,..,p}$ the histograms, according to some attribute A, of the fragments R_i of a relation R. If h is a hash function allowing to distribute evenly the distinct values of the attribute A, then the fragments $Hist_i(R)_{i=1,..,p}$ of the histogram of the relation R, can be computed, by hashing the local histograms $Hist(R_i)_{i=1,..,p}$, in time:*

$$O\Big(\min(g * |Hist(R)| + \|Hist(R)\|,\, g * \frac{|R|}{p} + \frac{\|R\|}{p}) + l\Big).$$

□

Proof:
The proof is done according to frequencies' of the values of the attribute A of the relation R.

- **1$^{\text{fst}}$ case:** All the values of the attribute A appear in the relation R with a frequency equal to 1, i.e. the number of tuples of the relation R is equal to the number of distinct values of the attribute A (and thus $\|Hist(R)\| = \|R\|$).
  In this case, the application of the hash function h, distributes the tuples relation R (and consequently the histogram of R) evenly on the various processors. Consequently, the redistribution of the histograms of $Hist(R_i)$, by using the function h, is done in time:

  $$O\Big(g * \frac{|Hist(R)|}{p} + \frac{\|Hist(R)\|}{p} + l\Big) \quad \leq \quad O\Big(g * \frac{|R|}{p} + \frac{\|R\|}{p} + l\Big). \tag{1}$$

  The term $g * \frac{|Hist(R)|}{p}$ is due to the fact that each processor receives a fragment of size about equal to $\frac{|Hist(R)|}{p}$ and the term $\frac{\|Hist(R)\|}{p}$ is due to time necessary to form the fragments, $Hist_i(R)_{i=1,..,p}$, starting from the received blocks, by each processor, during the redistribution of the local histograms $Hist(R_i)_{i=1,..,p}$.

  The inequality 1 is generally true, owing to the fact that $|Hist(R)| \leq |R|$ since the histogram contains only couples of the form $(d,freq(d))$ where d are the distinct values of some attribute of R and $freq(d)$ the corresponding frequencies (i.e. the number of tuples having the value d for this attribute). What shows that the histograms, $Hist_i(R)$, creation can be carried out in time:

  $$O\Big(\min(g * \frac{|Hist(R)|}{p} + \frac{\|Hist(R)\|}{p},\, g * \frac{|R|}{p} + \frac{\|R\|}{p}) + l\Big)$$
  $$\leq O\Big(\min(g * |Hist(R)| + \|Hist(R)\|,\, g * \frac{|R|}{p} + \frac{\|R\|}{p}) + l\Big). \tag{2}$$

- **2^{snd} case :** All the values of the attribute A appear in the relation R with a frequency lower than p. In this case the relation R can be partitioned into p relations R^1, R^2, ..., R^p so that the frequency of each value d in a relation R^j is exactly equal to 1. By applying the results of the 1^{fst} case to each relation R^j, we deduce that the creation of the fragments $Hist_i(R^j)$ of a relation R^j for $j = 1,..,p$ requires at most :

$$O\Big(\min(g * \frac{|Hist(R^j)|}{p} + \frac{\|Hist(R^j)\|}{p}, g * \frac{|R^j|}{p} + \frac{\|R^j\|}{p}) + l\Big),$$

and thus the creation of the fragments $Hist_i(R)$ of relation R requires therefore :

$$O\Big(\sum_{j=1,..p} \min(g * \frac{|Hist(R^j)|}{p} + \frac{\|Hist(R^j)\|}{p}, g * \frac{|R^j|}{p} + \frac{\|R^j\|}{p}) + l\Big)$$
$$\leq O\Big(\min(g * |Hist(R)| + \|Hist(R)\|, g * \frac{|R|}{p} + \frac{\|R\|}{p}) + l\Big), \quad (3)$$

because the fragment $Hist_i(R)$ can be created starting from the fragments $Hist_i(R^j)_{j=1,..,p}$ by a simple fusion of the histograms $Hist_i(R^j)_{1\leq j\leq p}$ in a time about $O(\sum_{j=1}^{p} \frac{\|Hist_i(R^j)\|}{p}) = O(\|Hist(R)\|)$, and that $\sum_i \min(a_i,b_i) \leq \min(\sum_i a_i, \sum_i b_i)$. We thus combine the p phases of communication in only one (the term is not multiplied by p).

The distribution of the relation R into p relations is carried out only to show the validity of the result. In practice, if a value d appears in the relation R_i with a frequency k, the application of the hash function to the histogram sends the couple (d,k) to the processor destination in only one sending and not in k messages of the form $(d,1)$. This reduces the amount of the data exchanged between the various processors.

- **3^{rd} case :** All the values of the attribute A appear in the relation R with frequencies higher than p. In this case, for a given value d, the tuples associated with d can be distributed on several processors and in the worst case distributed on p the processors. Thus the application of the hash function, h, to the histograms $Hist(R_i)_{i=1,..,p}$, sends couples of the form $(d,freq_i(d))_{i=1,..,p}$ to the same processor.
As each processor deals with block of size $\frac{\|Hist(R)\|}{p}$, it receives at most $p * \frac{|Hist(R)|}{p} = |Hist(R)|$ data from the other processors, and the cost of the redistribution of the histograms $Hist(R_i)$ is about :

$$O\Big(g * |Hist(R)| + l\Big) \quad \leq \quad O\Big(g * \frac{|R|}{p} + l\Big). \quad (4)$$

Inequality 4 holds because all the values of the attribute appear in R with a frequency higher than p (by assumption) and thus

$$\|Hist(R)\| \leq \frac{\|R\|}{p} \quad \text{and} \quad |Hist(R)| \leq \frac{|R|}{p}.$$

The term $\|Hist(R)\|$ is time necessary to form the histogram, $Hist_i(R)$, on the processor i starting from the fragments of the histograms $Hist(R_j)_{j=1,..,p}$ received after the stage of redistribution using the hash function h. And thus the creation of the histograms $Hist_i(R)$ requires :

$$O\Big(\min(g * |Hist(R)| + \|Hist(R)\|, g * \frac{|R|}{p} + \frac{\|R\|}{p}) + l\Big).$$

- **4^{rd} case :** the general case follows immediately from the three preceding cases. To note it, it is enough to partitionne relation R into two disjoined sub-relations R' and R'' so that the tuples of the relation R' (resp. R'') are associated with frequencies lower (resp. higher) than p for the attribute A. Thus, using the first two cases, the histogram R' can be carried out in time :

$$O\Big(\min(g * |Hist(R')| + \|Hist(R')\|, g * \frac{|R'|}{p} + \frac{\|R'\|}{p}) + l\Big), \quad (5)$$

and the creation of the histogram of R'', (cf. 3rd case), in time :

$$O\Big(\min(g * |Hist(R'')| + \|Hist(R'')\|,\, g * \frac{|R''|}{p} + \frac{\|R''\|}{p}) + l\Big) \tag{6}$$

By making the sum of the terms 5 and 6, one shows that the creation of the histograms $Hist_i(R)$ requires a at most :

$$O\Big(\min(g * |Hist(R)| + \|Hist(R)\|,\, g * \frac{|R|}{p} + \frac{\|R\|}{p}) + l\Big). \tag{7}$$

This is due to the fact that $Hist(R) = Hist(R') \cup Hist(R'')$ and that $\min(a,b) + \min(c,d) \leq \min(a + c, b + d)$, which proves the required result

■

Remark 4.1. *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 [CW79],*
- *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 a clever hash function to avoid load imbalance that result from these repeated values [DNSS92]. This case cannot arise here owing to the fact that the histogram contains only distinct values of the join attribute.*

Proposition 4.2. *Let $Hist(R_i)_{i=1,..,p}$ (resp. $Hist(S_i)_{i=1,..,p}$) the histograms, according to some attribute A, of the fragments R_i (resp. S_i) of a relation R (resp. S).*
If h is hash function allowing to distribute in an equitable way, the distinct values of the attribute A, then the two semi-joins, $R \ltimes S$ and $S \ltimes R$, can be computed in time of the order :

$$\begin{aligned} Time_{sj} \;=\; O\Bigg(& \min\left(g * |Hist(R)| + \|Hist(R)\|,\, g * \frac{|R|}{p} + \frac{\|R\|}{p}\right) \\ & + \min\left(g * |Hist(S)| + \|Hist(S)\|,\, g * \frac{|S|}{p} + \frac{\|S\|}{p}\right) + \max_{i=1,..,p}(\|R_i\| + \|S_i\|) + l\Bigg). \end{aligned}$$

□

Proof :
The semi-joins computation, $R \ltimes S$ and $S \ltimes R$, is done in five stages.

a. A stage of creation of the fragments $Hist_i(R)_{i=1,..,p}$ and $Hist_i(S)_{i=1,..,p}$ of the histograms of R and S respectively.

On each processor i, the application of the hash function distributes the histogram $Hist(R_i)$ (resp. $Hist(S_i)$) into p buckets $Hist^j(R_i)_{i=1,..,p}$ (resp. $Hist^j(S_i)_{i=1,..,p}$) where $Hist^j(R_i)$ (resp. $Hist^j(S_i)$) is the fragment of $Hist(R_i)$ (resp. $Hist(S_i)$) which will be sent to the processor j using the hash function h. By using the proposition 4.1, we have

$$\begin{cases} |\bigcup_i Hist^j(R_i)| & \leq & \min(|Hist(R)|, \frac{|R|}{p}), \\ |\bigcup_i Hist^j(S_i)| & \leq & \min(|Hist(S)|, \frac{|S|}{p}). \end{cases}$$

The fragments $Hist^j(R_i)_{i=1,..,p}$ (resp. $Hist^j(S_i)_{i=1,..,p}$) are then gathered, on each processor j, to form $Hist^j(R)$ (resp. $Hist^j(S)$) in time :

$$O(\min(\|Hist(R)\|, \tfrac{\|R\|}{p})) \text{ (resp. } O(\min(\|Hist(S)\|, \tfrac{\|S\|}{p}))).$$

Hence, the creation of the fragments $Hist_i(R)_{i=1,..,p}$ and $Hist_i(S)_{i=1,..,p}$ of the histograms of R and S requires:

$$\begin{aligned} Time_{\text{step.a}} &= O\Bigg(\min\Big(g * |Hist(R)| + \|Hist(R)\|,\, g * \frac{|R|}{p} + \frac{\|R\|}{p}\Big) \\ &\quad + \min\Big(g * |Hist(S)| + \|Hist(S)\|,\, g * \frac{|S|}{p} + \frac{\|S\|}{p}\Big) + l\Bigg). \end{aligned}$$

b. A stage of creation of the fragments $Hist_i(R \bowtie S)_{i=1,..,p}$.
The fragment $Hist_i(R \bowtie S)$ is then computed on each processor i, by "intersecting" $Hist_i(R)$ and $Hist_i(S)$, in time:

$$Time_{\text{step.b}} = O\Big(max_{i=1,..,p}(min(\|Hist_i(R)\|,\|Hist_i(S)\|))\Big).$$

Remark 4.2. *During the creation of the fragments of $Hist(R \bowtie S)$ we store, for each value $d \in Hist_i(R \bowtie S)$, an extra information called index ($index(d) \in \{1,2,3\}$). This information will allow us to decide if, for a given value d, the frequencies of tuples of relations R and S having the value d are greater (resp. lesser) than a threshold frequency f_0. It also permit us to choose dynamically the probe and 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 the same threshold frequency as in the redistribution algorithm of fa_join [BH98, BH00a], i.e. $f_0 = p * log(p)$. For a given value $d \in Hist(R \bowtie S)$,

- *the value $index(d) = 3$, means that the frequency of tuples of relations R and S having the value d are less than the threshold frequency. $Hist(R)(d) < f_0$ and $Hist(S)(d) < f_0$.*
- *the value $index(d) = 1$, means that $Hist(R)(d) \geq f_0$ and $Hist(R)(d) \geq Hist(S)(d)$.*
- *the value $index(d) = 2$, means that $Hist(S)(d) \geq f_0$ and $Hist(S)(d) > Hist(R)(d)$.*

The cost of this operation, does not change in anything the total cost the creation of the fragments, $Hist_i(R \bowtie S)_{i=1,..,p}$ of the histogram $Hist(R \bowtie S)$.

c. Once the stage b finished, we compute on each processor j, the intersections:

$$\begin{cases} \overline{Hist}^j(R_i) &= Hist^j(R_i) \cap Hist_j(R \bowtie S) \textbf{ for } i = 1,..,p \textbf{ and} \\ \overline{Hist}^j(S_i) &= Hist^j(S_i) \cap Hist_j(R \bowtie S) \textbf{ for } i = 1,..,p \end{cases}$$

in time:

$$O\Big(\sum_i \|Hist^j(R_i)\| + \sum_i \|Hist^j(S_i)\|\Big),$$

indeed, in accordance with stage a., one has :

$$\begin{cases} \sum_i \|Hist^j(R_i)\| &= \| \cup_i Hist^j(R_i)\| \leq \min\Big(\|Hist(R)\|, \frac{\|R\|}{p}\Big) \\ \sum_i \|Hist^j(S_i)\| &= \| \cup_i Hist^j(S_i)\| \leq \min\Big(\|Hist(S)\|, \frac{\|S\|}{p}\Big), \end{cases}$$

and thus the total cost of this stage is:

$$Time_{\text{step.c}} = O\Big((min(\|Hist(R)\|, \frac{\|R\|}{p}) + min(\|Hist(S)\|, \frac{\|S\|}{p})\Big).$$

Let us note that, during the computation of the fragments $\overline{Hist}^j(R_i)$, the value $index(d)$ for each $d \in \overline{Hist}^j(R_i)$ is also transmitted to $\overline{Hist}^j(R_i)$ without changing the order of magnitude of the cost of this stage.

d. A stage of communication where each processor j sends the fragment $\overline{Hist}^j(R_i)$ to the processor i. Thus, during this stage of communication, each processor i, receives $\sum_j |\overline{Hist}^j(R_i)|$ pages of data from the other processors. The various stages of the creation of the histograms $\overline{Hist}(R_i)_{i=1,..,p}$ are illustrated in the figure Fig. 2. We point out that :$Hist(R_i) = \cup_j Hist^j(R_i)$ and that :

$$|Hist(R_i)| = \sum_j |Hist^j(R_i)| \geq \sum_j |Hist^j(R_i) \cap Hist(R \bowtie S)|$$

and thus $|Hist(R_i)| \geq \sum_j |\overline{Hist}^j(R_i)|$ and consequently the total cost of this stage is at most :

$$Time_{\text{step.d}} = O\Big(g * |\overline{Hist}(R_i)| + l\Big).$$

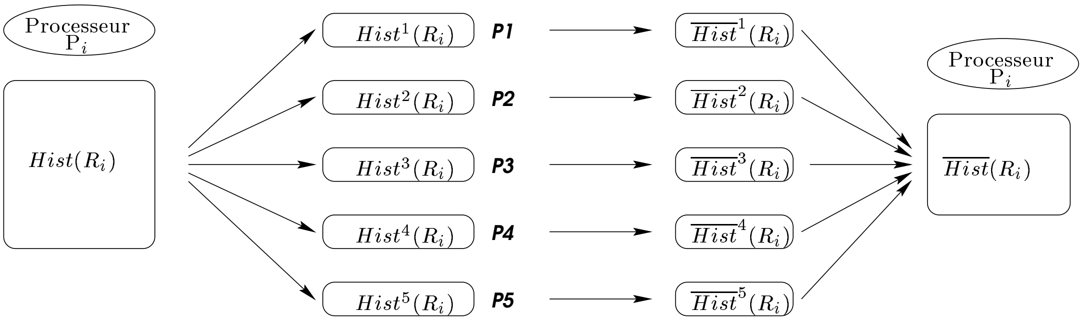

FIG. 2 – *Various stages of the creation of the histograms $\overline{Hist}(R_i)$.*

Remark 4.3. $\cup_j \overline{Hist}^j(R_i)$ *is simply the intersection of $Hist(R_i)$ and the histogram $Hist(R \bowtie S)$ which will be noted :*

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

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

e. A stage of the local semi-joins $R_i \ltimes S$ and $S_i \ltimes R$ computation.

Owing to the fact that, at the end of stage d., each processor i has the fragment $\overline{Hist}(R_i) = Hist(R_i) \cap Hist(R \bowtie S)$, the semi-join $R_i \ltimes S$ (resp. $S_i \ltimes R$) can be computed, in parallel on each processor i, by a scan of the fragment R_i (resp. S_i) and trying the access to the histogram $\overline{Hist}(R_i)$ in time $O(\|R_i\|)$ (resp. $O(\|S_i\|)$) [7]. Thus, the total cost of this stage is :

$$Time_{\text{step.e}} = O\Big(\max_{i=1,..,p} (\|R_i\| + \|S_i\|)\Big).$$

The total cost of the two semi-joins $R \ltimes S$ and $S \ltimes R$ computation, is thus the sum of the above five stages :

$$\begin{aligned} Time_{sj} &= O\Bigg(\min\Big(g * |Hist(R)| + \|Hist(R)\|,\ g * \frac{|R|}{p} + \frac{\|R\|}{p}\Big) \\ &\quad + \min\Big(g * |Hist(S)| + \|Hist(S)\|,\ g * \frac{|S|}{p} + \frac{\|S\|}{p}\Big) + \max_{i=1,..,p} (\|R_i\| + \|S_i\|) + l \Bigg). \end{aligned}$$

7. In the case where the relation R (resp. S) is indexed on the join attribute, the semi-join $R_i \ltimes S$ (resp. $S_i \ltimes R$) can be computed, on each processor i, by a scan of the histogram $\overline{Hist}(R_i)$ and trying the access to the index of the relation R_i (resp. S_i)

■

Remark 4.4. *During the computation of the semi-joins $R_i \ltimes S$ (resp. $S_i \ltimes R$) in the stage e. of the preceding proof, each generated tuple is immediately inserted into :*

- $\widetilde{R'_i}$ *(resp.* $\widetilde{S'_i}$*) when the value* $index(d)$ *of this tuple is equal to* 1 *(resp.* 2*),*
- $\widetilde{R''_i}$ *(resp.* $\widetilde{S''_i}$*) when the value* $index(d)$ *of this tuple is equal to* 2 *(resp.* 1*),*
- $\widetilde{R'''_i}$ *(resp.* $\widetilde{S'''_i}$*) when the value* $index(d)$ *of this tuple is equal to* 3 *(resp.* 3*).*

This partitioning of the semi-joins $R_i \ltimes S$ (resp. $S_i \ltimes R$) into three sub-relations is used to reduce the cost of communication and to deal with the problems of attribute value and join product skews. The cost of this partitioning does not change in anything the total cost of the semi-joins computation.

4.2 Data redistribution in Osfa_join algorithm

We will describe the algorithm while giving an upper bound on the BSP execution time of each phase. The $O(\ldots)$ notation only hides small constant factors: they depend on the implementation program but neither on data nor on the BSP machine parameters.

Our redistribution algorithm is the basis for efficient and scalable join processing. It proceeds in 4 phases:

Phase 1 : Creating local histograms

Local histograms $Hist(R_i)_{i=1,..,p}$ (resp. $Hist(S_i)_{i=1,..,p}$) of blocks R_i (resp. S_i) are created in parallel by a traversal of R_i (resp. S_i) in time $c_{i/o} * \max_{i=1,..,p} |R_i|$ (resp. $c_{i/o} * \max_{i=1,..,p} |S_i|$) where $c_{i/o}$ is a constant factor for disk input/output operations. In principle, this phase costs :

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

but in practice, the extra cost for this operation is negligible because the histograms can be computed on the fly while creating local hash tables.

Phase 2 : Local semi-joins computation

In order to minimize the redistribution cost and thus the communication time between the various processors, we compute the following local semi-joins : $\widetilde{R_i} = R_i \ltimes S$ (resp. $\widetilde{S_i} = S_i \ltimes R$) using proposition 4.2 in time :

$$\begin{aligned} Time_{phase2} &= O\Big(\max_{i=1,..,p} (\|R_i\| + \|S_i\|) + \min\left(g * |Hist(R)| + \|Hist(R)\|,\, g * \frac{|R|}{p} + \frac{\|R\|}{p}\right) \\ &\quad + \min\left(g * |Hist(S)| + \|Hist(S)\|,\, g * \frac{|S|}{p} + \frac{\|S\|}{p}\right) + l\Big). \end{aligned}$$

Note that, unlike the hash-based algorithms where both relation R and S are redistributed, and as an improvement to the *fa_join* algorithm where only relations R and $S \ltimes R$ are redistributed, we will only redistribute $R \ltimes S$ and $S \ltimes R$ to perform the join operation $R \bowtie S$. This will reduce communication costs to a minimum.

At the end of this phase, on each processor i, the semi-join $\widetilde{R_i} = R_i \ltimes S$ (resp. $\widetilde{S_i} = S_i \ltimes R$) is divided into three sub-relations in the following way :

$$\widetilde{R_i} = \widetilde{R'_i} \cup \widetilde{R''_i} \cup \widetilde{R'''_i} \quad \text{and} \quad \widetilde{S_i} = \widetilde{S'_i} \cup \widetilde{S''_i} \cup \widetilde{S'''_i} \text{ where :}$$

- All the tuples of relation $\widetilde{R'_i}$ (resp. $\widetilde{S'_i}$) are associated to values d such that $index(d) = 1$ (resp. $index(d) = 2$),

- All the tuples of relation $\widetilde{R''_i}$ (resp. $\widetilde{S''_i}$) are associated to values d such that $index(d) = 2$ (resp. $index(d) = 1$),
- All the tuples of relation $\widetilde{R'''_i}$ (resp. $\widetilde{S'''_i}$) are associated to values d such that $index(d) = 3$ (resp. $index(d) = 3$), i.e. the tuples associated to values which occur with frequencies less than a threshold frequency f_0 in both relations R and S.

The tuples of the relations $\widetilde{R'_i}$ and $\widetilde{S'_i}$ is associated to high frequencies for the join attribute. These tuples have an important effect on attribute value and join product skews. They will be redistributed using an appropriate redistribution algorithm to efficiently avoid both AVS and JPS. However the tuples of relations $\widetilde{R'''_i}$ and $\widetilde{S'''_i}$ (are associated to very low frequencies for the join attribute) have no effect neither on AVS nor JPS. These tuples will be redistributed using a hash function.

Phase 3 : Creation of communication templates

The attribute values which could lead to attribute value skew (those having high frequencies) are also those which may cause join product skew in standard algorithms. To avoid the slowdown usually caused by attribute value skew and the imbalance of the size of local joins processed by the standard algorithms, an appropriate treatment for high attribute frequencies is needed.

3.a To this end, we partition the histogram $Hist(R \bowtie S)$ into two sub-histograms : $Hist^{(1,2)}(R \bowtie S)$ and $Hist^{(3)}(R \bowtie S)$ in the following manner :

- the values $d \in Hist^{(1,2)}(R \bowtie S)$ are associated to high frequencies of the join attribute (i.e. $index(d) = 1$ or $index(d) = 2$),
- the values $d \in Hist^{(3)}(R \bowtie S)$ are associated to low frequencies of the join attribute (i.e. $index(d) = 3$).

this partition step is performed in parallel, on each processor i, by a local traversal of the histogram $Hist_i(R \bowtie S)$ in time :

$$Time_{phase3.a} = O(\max_{i=1,..,p} \|Hist_i(R \bowtie S)\|).$$

3.b Communication templates for high frequencies

We first create a communication template : the list of messages which constitute 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**.

Processor i computes a set of necessary messages relating to the values d it owns in $Hist_i^{(1,2)}(R \bowtie S)$. The communication template is derived from the following mapping, its intended result. For relation $T \in \{\widetilde{R'},\widetilde{S'}\}$, tuples of T are mapped to multiple nodes as follows :

---

if $\left(Hist(T)(d) \quad mod(p) = 0\right)$ **then**

each processor j will hold an equal number $block_j(d) = \frac{Hist(T)(d)}{p}$ of tuples of value d.

else

begin

- Pick a random value j_0 between 0 and $(p-1)$
- **if** processor index j is between j_0 and $j_0 + (Hist(T)(d) \quad mod(p))$ **then**

  the processor of index j will hold a block of size : $block_j(d) = \lfloor \frac{Hist(T)(d)}{p} \rfloor + 1$

  else

  processor of index j will hold a block of size : $block_j(d) = \lfloor \frac{Hist(T)(d)}{p} \rfloor$.

end.

---

where $\lfloor x \rfloor$ is the largest integral value not greater than x and $block_j(d)$ be the number of tuples of value d which 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 which processor j must send (if $Rest_j(d) > 0$) or receive (if $Rest_j(d) < 0$).

For $d \in Hist_i^{(1,2)}(R \bowtie S)$, 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. < 0) send (resp. receive) tuples of value d. This step is thus completed in time :

$$Time_{phase3.b} = O(\|Hist^{(1,2)}(R \bowtie S)\|).$$

The tuples associated to low frequencies (i.e. tuples having $d \in Hist_i^{(3)}(R \bowtie S)$) have no effect neither on the AVS nor the JPS. These tuples are simply mapped to processors using a hash function and thus no communication template computation is needed.
The creation of communication templates has therefore taken the sum of the above two steps :

$$\begin{aligned} Time_{phase3} &= Time_{phase3.a} + Time_{phase3.b} \\ &= O\Big(\max_{i=1,..,p} \|Hist_i(R \bowtie S)\| + \|Hist^{(1,2)}(R \bowtie S)\|\Big). \end{aligned}$$

Phase 4 : Data redistribution

4.a Redistribution of tuples having $d \in Hist_i^{(1,2)}(R \bowtie S)$:
Every processor i holds, for every one of its local $d \in Hist_i^{(1,2)}(R \bowtie S)$, 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 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 in 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 :

$$\begin{cases} Rest_{Proc(j)}(d) > 0 & for \quad j = 1,..,\alpha \\ Rest_{Proc(j)}(d) < 0 & for \quad j = (\alpha+1),..,\beta \end{cases}$$

A sequential traversal of $Proc(k)_{k=1,..\alpha+\beta}$ determines the number of tuples each processor j will send. The sending orders concerning attribute value d are computed by the following procedure :

```
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; fi
            * if Rest_Proc(j)(d) = 0 then j := j + 1; fi
        end.
```

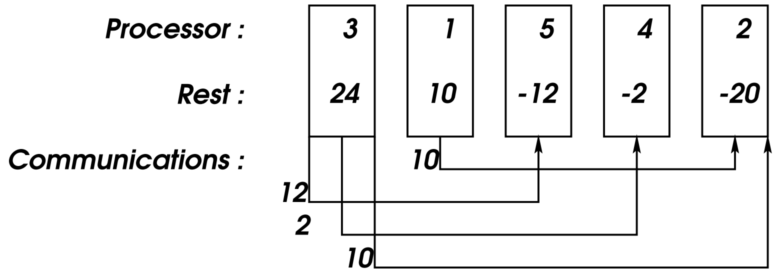

FIG. 3 – *Example of communication template as a function of Rest values.*

of complexity $O(\|Hist^{(1,2)}(R \bowtie S)\|)$ 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 $d \in Hist_i^{(1,2)}(R \bowtie S)$.
For every i and $d \in Hist_i^{(1,2)}(R \bowtie S)$, all the orders `order_to_send`($j$,$i$,...) are sent to processor j when $j \neq i$ in time $O\left(g * |Hist^{(1,2)}(R \bowtie S)| + l\right)$. Figure 3 gives an example of *Rest* values for a certain d and corresponding sending orders.

In all, this step costs at most : $O\left(g * |Hist^{(1,2)}(R \bowtie S)| + \|Hist^{(1,2)}(R \bowtie S)\| + l\right)$.

4.b Redistribution of tuples with values $d \in Hist_i^{(3)}(R \bowtie S)$:
We recall that tuples of relations $\widetilde{R_i'''}$ and $\widetilde{S_i'''}$ (i.e. tuples having $d \in Hist_i^{(3)}(R \bowtie S)$) have no effect neither on the AVS nor the JPS. These tuples 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 $\widetilde{R_i}$ and $\widetilde{S_i}$ will be redistributed. Redistribution is then performed, in time :

$$O(g * (|\widetilde{R_i}| + |\widetilde{S_i}|) + l).$$

Phase 4, has therefore taken the sum of the above two costs :

$$Time_{phase4} = O\Big(g * \max_{i=1,..,p} (|\widetilde{R_i}| + |\widetilde{S_i}| + |Hist^{(1,2)}(R \bowtie S)|) + \|Hist^{(1,2)}(R \bowtie S)\| + l\Big),$$

and the complete redistribution algorithm costs :

$$\begin{aligned} Time_{\text{Osfa-redist}} &= Time_{phase1} + Time_{phase2} + Time_{phase3} + Time_{phase4} \\ &= O\Big(c_{i/o} * \max_{i=1,..,p} (|R_i| + |S_i|) + \min\left(g * |Hist(R)| + \|Hist(R)\|,\, g * \frac{|R|}{p} + \frac{\|R\|}{p}\right) \\ &\quad + \min\left(g * |Hist(S)| + \|Hist(S)\|,\, g * \frac{|S|}{p} + \frac{\|S\|}{p}\right) + \max_{i=1,..,p} (\|R_i\| + \|S_i\|) \\ &\quad + g * (|\widetilde{R_i}| + |\widetilde{S_i}| + |Hist^{(1,2)}(R \bowtie S)|) + \|Hist^{(1,2)}(R \bowtie S)\| + l\Big). \end{aligned}$$

We mention that, we only redistribute the semi-joins $\widetilde{R_i}$ and $\widetilde{S_i}$. Note that $|\widetilde{R_i}|$ (resp. $|\widetilde{S_i}|$) is generally very small compared to $|R_i|$ (resp. $|S_i|$) and $|Hist(R \bowtie S)|$ is generally very small compared to $|Hist(R)|$ and $|Hist(S)|$. Thus we reduce the communication cost to a minimum.

5 Osfa_join : An optimal and skew-insensitive join algorithm

To perform the join of two relations R and S, we first redistribute relations R and S using the above redistribution algorithm at the cost of :

$$Time_{\text{Osfa-redist}} = O\Big(c_{i/o} * \max_{i=1,..,p} (|R_i| + |S_i|) + \min\left(g * |Hist(R)| + \|Hist(R)\|,\, g * \frac{|R|}{p} + \frac{\|R\|}{p}\right)$$

$$+\min\Big(g * |Hist(S)| + \|Hist(S)\|,\, g * \frac{|S|}{p} + \frac{\|S\|}{p}\Big) + \max_{i=1,..,p}(\|R_i\| + \|S_i\|)$$
$$+g * (|\widetilde{R_i}| + |\widetilde{S}_i| + |Hist^{(1,2)}(R \bowtie S)|) + \|Hist^{(1,2)}(R \bowtie S)\| + l\Big).$$

Once the redistribution phase is completed, the semi-joins $\widetilde{R_i}$ (resp. $\widetilde{S}_i$) are partitioned into three disjoint relations as follow: $\widetilde{R_i} = \widetilde{R_i'} \cup \widetilde{R_i''} \cup \widetilde{R_i'''}$ (resp. $\widetilde{S}_i = \widetilde{S_i'} \cup \widetilde{S_i''} \cup \widetilde{S_i'''}$) as described in phase 2.

Taking advantage of the identities:

$$\begin{aligned}
R \bowtie S &= \widetilde{R} \bowtie \widetilde{S} = (\widetilde{R'} \cup \widetilde{R''} \cup \widetilde{R'''}) \bowtie (\widetilde{S'} \cup \widetilde{S''} \cup \widetilde{S'''}) \\
&= (\widetilde{R'} \bowtie \widetilde{S''}) \cup (\widetilde{R''} \bowtie \widetilde{S'}) \cup (\widetilde{R'''} \bowtie \widetilde{S'''}) \\
&= (\widetilde{R'} \bowtie \widetilde{S''}) \cup (\widetilde{R''} \bowtie \widetilde{S'}) \cup (\widetilde{R'''} \bowtie \widetilde{S'''}) \\
&= \Big(\bigcup_i \widetilde{R_i'} \bowtie \widetilde{S''}\Big) \cup \Big(\bigcup_i \widetilde{R''} \bowtie \widetilde{S_i'}\Big) \cup \Big(\bigcup_i \widetilde{R_i'''} \bowtie \widetilde{S_i'''}\Big).
\end{aligned} \tag{8}$$

Frequencies of tuples of relations $\widetilde{R_i'}$ (resp. $\widetilde{S_i'}$) are by definition greater than the corresponding (matching) tuples in relations $\widetilde{S_i''}$ (resp. $\widetilde{R_i''}$). Fragments $\widetilde{R_i'}$ (resp. $\widetilde{S_i'}$) will be thus chosen as *build* relations and $\widetilde{S_i''}$ (resp. $\widetilde{R_i''}$) as *probe* relations to be duplicated on each processor. This improves over fa_join where all the semi-join $S \rtimes R$ is duplicated. It reduces communications costs significantly in asymmetric cases where both relations contain frequent & infrequent values.

To perform the join $R \bowtie S$, it is sufficient to compute the three following local joins $\widetilde{R_i'} \bowtie \widetilde{S''}$, $\widetilde{R''} \bowtie \widetilde{S_i'}$ and $\widetilde{R_i'''} \bowtie \widetilde{S_i'''}$ (cf. equation 8). To this end, we first broadcast the fragments $\widetilde{R_i''}$ (resp. $\widetilde{S_i''}$) to all processors in time:

$$Time_{step.a} = O\Big(g * (|\widetilde{R''}| + |\widetilde{S''}|) + l\Big).$$

Local joins $\widetilde{R_i'} \bowtie \widetilde{S''}$, $\widetilde{R''} \bowtie \widetilde{S_i'}$ and $\widetilde{R_i'''} \bowtie \widetilde{S_i'''}$ could be done in time:

$$\begin{aligned}
Time_{step.b} &= c_{i/o} * O\Big(\max_{i:1,..,p}(|\widetilde{R_i'}| + |\widetilde{S''}| + |\widetilde{R_i'} \bowtie \widetilde{S''}|) + \max_{i:1,..,p}(|\widetilde{R''}| + |\widetilde{S_i'}| + |\widetilde{R''} \bowtie \widetilde{S_i'}|) \\
&\qquad + \max_{i:1,..,p}(|\widetilde{R_i'''}| + |\widetilde{S_i'''}| + |\widetilde{R_i'''} \bowtie \widetilde{S_i'''}|)\Big) \\
&= c_{i/o} * O\Big(\max_{i:1,..,p}(|\widetilde{R_i'} \bowtie \widetilde{S''}| + |\widetilde{R_i'} \bowtie \widetilde{S''}| + |\widetilde{R_i'''} \bowtie \widetilde{S_i'''}|)\Big).
\end{aligned} \tag{9}$$

The equation 9 holds due to the fact that the join size is at least equal to the maximum of the sizes of the semi-joins.

The cost of the local join computation is thus the sum of the two costs above:

$$\begin{aligned}
Time_{local-join} &= Time_{step.a} + Time_{step.b} \\
&= c_{i/o} * O\Big(g * (|\widetilde{R''}| + |\widetilde{S''}|) + \max_{i:1,..,p}(|\widetilde{R_i'} \bowtie \widetilde{S''}| + |\widetilde{R''} \bowtie \widetilde{S_i'}| + |\widetilde{R_i'''} \bowtie \widetilde{S_i'''}|) + l\Big).
\end{aligned}$$

The global cost of the join of relations R and S using the Osfa_join algorithm is the sum of the redistribution cost with local join computation cost. It is of the order:

$$\begin{aligned}
Time_{\text{Osfa-join}} &= O\Big(c_{i/o} * \max_{i=1,..,p}(|R_i| + |S_i|) + \min\Big(g * |Hist(R)| + \|Hist(R)\|, g * \frac{|R|}{p} + \frac{\|R\|}{p}\Big) \\
&\quad + \min\Big(g * |Hist(S)| + \|Hist(S)\|,\, g * \frac{|S|}{p} + \frac{\|S\|}{p}\Big) + \max_{i=1,..,p}(\|R_i\| + \|S_i\|) \\
&\quad + g * (|\widetilde{R_i}| + |\widetilde{S}_i| + |Hist^{(1,2)}(R \bowtie S)|) + \|Hist^{(1,2)}(R \bowtie S)\| + l\Big).
\end{aligned}$$

The join of relations R and S using the Osfa_join algorithm, avoid JPS because the values which could lead to attribute value skew (those having high frequencies) are those which often cause the join product skew. This values are mapped to multiple nodes so that local joins have almost the same sizes. It avoids thus, the slowdown usually caused by attribute value skew and the imbalance of the size of local joins processed by the standard algorithms. We recall that, the redistribution cost is minimal because redistribution concerns only tuples which are effectively present in the join result.
Note that, local joins have almost the same size $\simeq \frac{|R \bowtie S|}{p}$.

Remark: Observe that sequential θ-join processing of two relations R and S requires at least the following lower bound: $bound_{inf_1} = \Omega\left(c_{i/o} * (|R| + |S| + |R \bowtie_\theta S|)\right)$.
Parallel processing with p processors therefore requires $bound_{inf_p} = \frac{1}{p} * \Omega\left(|R| + |S| + |R \bowtie_\theta S|\right)$, and the *Osfa_join* algorithm has *optimal* asymptotic complexity when:

$$|Hist^{(1,2)}(R \bowtie S)| \leq c_{i/o} * \max(\frac{|R|}{p}, \frac{|S|}{p}, \frac{|R \bowtie S|}{p}) \qquad (10)$$

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

6 Conclusion

In this paper, we have introduced an parallel join algorithm with *optimal complexity* based on a "symmetric" sub-set replication allowing to reduce the communication costs while guaranteeing near perfect balancing properties. Its predictably low join product skew makes it suitable for multi-join operations.
The *Osfa_join* algorithm is verified to improve on *fa_join* in three ways:

1. The negative effect of non-correlated join attribute value distributions is completely eliminated at *no extra cost.*
2. The JPS, already very low for *fa_join*, is now completely eliminated which implies even faster multi-joins.
3. The algorithm *Osfa-join* is proved to have optimal complexity.

The performance of this algorithm was analyzed using the BSP cost model which predicts a linear speedup. The $O(\ldots)$ notation only hides small constant factors: they depend only on the implantation but neither on data nor on the the BSP machine. Our experience with the join operation [BH98, BH00a, BH00b] is evidence that the above theoretical analysis is accurate in practice.

Acknowledgment

The author thanks FECIT [8] for free access to an AP-3000 system.

Références

[BBH99] M. Bamha, F. Bentayeb, and G. Hains. An efficient scalable parallel view maintenance algorithm for shared nothing multi-processor machines. In *the 10th International Conference on Database and Expert Systems Applications DEXA'99*, volume 1677 of *Lecture Notes in Computer Science*, pages 616–625, Florence, Italy, 1999. Springer-Verlag.

[BH98] M. Bamha and G. Hains. A self-balancing join algorithm for SN machines. In Y. Pan, S. G. Akl, and K. Li, editors, *Proceedings of International Conference on Parallel and Distributed Computing and Systems (PDCS)*, pages 285–290, Las Vegas, Nevada, USA, October 1998. IASTED/ACTA Press.

8. FECIT: Fujitsu European Center for Information and Technology.

[BH00a] M. Bamha and G. Hains. A frequency adaptive join algorithm for Shared Nothing machines. *Journal of Parallel and Distributed Computing Practices (PDCP)*, 2000. To appear.

[BH00b] M. Bamha and G. Hains. A symmetric frequency-adaptive join algorithm for shared nothing machines. Research Report RR-LIFO-2000-03, LIFO, Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans, BP 6759, F-45067 Orléans Cedex 2, 2000.

[CW79] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, April 1979.

[DG92] D. J. DeWitt and J. Gray. Parallel database systems: The future of high performance database systems. *Communications of the ACM*, 35(6):85–98, June 1992.

[DMT98] A. Datta, B. Moon, and H. Thomas. 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, 1998.

[DNSS92] D. J. DeWitt, J. F. Naughton, D. A. Schneider, and S. Seshadri. Practical Skew Handling in Parallel Joins. In *Proceedings of the 18th VLDB Conference*, pages 27–40, Vancouver, British Columbia, Canada, 1992.

[HK95] L. Harada and M. Kitsuregawa. Dynamic join product skew handling for hash-joins in shared-nothing database systems. In *Fourth International Conference on Database Systems for Advanced Applications*, pages 246–255, 1995.

[HL91] K. A. Hua and C. Lee. Handling data skew in multiprocessor database computers using partition tuning. In G. M. Lohman, A. Sernadas, and R. Camps, editors, *Proc. of the 17th International Conference on Very Large Data Bases*, pages 525–535, Barcelona, Catalonia, Spain, 1991. Morgan Kaufmann.

[HTY95] K. A. Hua, W. Tavanapong, and H. C. Young. A Performance Evaluation of Load Balancing Techniques for Join Operations on Multicomputer Database systems. In P. S. Yu and A. L. P. Chen, editors, *Proceedings of the 11th International Conference on Data Engineering*, pages 44–51, Los Alamos, CA, USA, 1995. IEEE Computer Society Press.

[KO90] M. Kitsuregawa and Y. Ogawa. Bucket spreading parallel hash: A new, robust, parallel hash join method for skew in the super database computer (SDC). *Very Large Data Bases: 16th International Conference on Very Large Data Bases, August 13–16, Brisbane, Australia*, pages 210–221, 1990.

[LOT94] H. Lu, B.-C. Ooi, and K.-L. Tan. *Query Processing in Parallel Relational Database Systems.* IEEE Computer Society Press, Los Alamos, California, 1994.

[Mär98] H. Märtens. Skew-insensitive join processing in shared-disk database systems. *Proc. of Issues and Applications of Database Technology (IADT '98), Berlin*, pages 17–24, July 1998.

[MMSY94] A. N. Mourad, R. J. T. Morris, A. Swami, and H. C. Young. Limits of parallelism in hash join algorithms. *Performance evaluation*, 20(1/3):301–316, May 1994.

[PI96] V. Poosala and Y. E. Ioannidis. Estimation of query-result distribution and its application in parallel-join load balancing. *In: Proc. 22th Int. Conference on Very Large Database Systems, VLDB'96, pp. 448-459, Bombay, India*, September 1996.

[RS95] E. Rahm and T. Stöhr. Analysis of parallel scan processing in shared disk database systems. *Proc. of EURO-PAR'95 Conference, Stockholm, Springer-Verlag, LNCS*, pages 485–500, August 1995.

[SD89] D. Schneider and D. DeWitt. A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment. In J. Clifford, B. Lindsay, and D. Maier, editors, *Proceedings of the 1989 ACM SIGMOD International Conference on the Management of Data, Portland, Oregon*, pages 110–121, New York, NY 10036, USA, 1989. ACM Press.

[SY90] M. Seetha and P. S. Yu. Effectiveness of parallel joins. *IEEE, Transactions on Knowledge and Data Enginneerings*, 2(4):410–424, December 1990.

[Val90] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.

[WDYT94] Joel L. Wolf, Daniel M. Dias, Philip S. Yu, and John Turek. New algorithms for parallelizing relational database joins in the presence of data skew. *IEEE Transactions on Knowledge and Data Engineering*, 6(6):990–997, 1994.

[WFA95] A. N. Wilschut, J. Flokstra, and P. M.G. Apers. Parallel evaluation of multi-join queries. *Procedings of the ACM-SIGMOD*, 24(2):115–126, 1995.