Query Acceleration in Distributed Database Systems

Ramzi A. Haraty[1] and Roula C. Fany[2]
[1] Lebanese American University, P.O. Box 13-5053
Beirut, Lebanon
Fax: 011-9611-867098
Email: rharaty@beirut.lau.edu.lb
[2] Beirut Riyad Bank, Riyad Soloh Street, Beirut Riyad Bank Building
Beirut, Lebanon,
Email: rolafany@sodetel.net.lb

Abstract

The advent of telecommunication era and the constant development of hardware and network structures have encouraged the decentralization of data while increasing the needs to access information from different sites. Query optimization strategies aim to minimize the cost of transferring data across networks. Many techniques and algorithms have been proposed to optimize queries. Perhaps one of the more important algorithms is the AHY algorithm using semi-joins that is implemented by Apers, Hevner and Yao in [1]. Nowadays, a new technique called PERF (Partially Encoded Record Filters) seems to bring some improvement over semi-joins [12]. PERF joins are two-way semi-joins using a bit vector as their backward phase. Our research encompasses applying PERF joins to two well know algorithms: AHY and W, which both deal with query optimization. Programs were designed to implement both the original and the enhanced algorithms. Several experiments were conducted and the results showed a very considerable enhancement obtained by applying the PERF concept. This major improvement led us to further observations and studies.

Key Words: Query Acceleration, PERF Joins, and Semi Joins.

1 - Introduction

The recent telecommunication boom has encouraged business expansion resulting in the decentralization of data while increasing the needs for instant information access.

A distributed database system is a collection of sites connected on a common high-bandwidth network [9]. Logically, data belongs to the same system but physically it is spread over the sites of the network, making the distribution invisible to the user [5]. Each site is an autonomous database with its processing capability and data storage capacity. The advantage of this distribution resides in achieving availability, modularity, performance, and reliability.

Distributed query processing is the process of retrieving data from different sites. Accessing data from sites involves transmission via communication links, which creates delays. The basic challenge is to design and develop efficient query processing techniques and strategies to minimize the communication cost.

Nowadays, with the explosion of interest in data warehouses and the development of huge applications such as federation and mediation over heterogeneous and object-oriented databases, there is a pressing need for data reduction to minimize data shipping costs. This is the main purpose of query optimization which estimates the cost of alternative query plans in order to choose the best plan to answer quickly and efficiently, complex and expensive queries [11].

The query optimization problem was addressed many times, from different perspectives, and a lot of work has been done. Proposed algorithms and techniques can be categorized in two main approaches [7], [6], [8], [14], [10], [2], [13], and [11]:

1- Minimize the cost of data transferred across the network by reducing the amount of transmitted information, and
2- Minimize the response time of the query by using parallel processing techniques.

Some might even add another category, which is the hybrid approach, merging both data reduction and time reduction.

One of the most popular and important algorithms suggested for query optimization with minimum cost was algorithm GENERAL (total cost) presented by Apers, Hevner and Yao [1]. The advent of AHY was a revolution

in query optimization domain because it introduced semi-joins as reducers in the query optimization process. It uses the three-phased approach method, which consists of the following:

- Local processing to filter unnecessary data.
- Semi-join reduction involving data shipment from one site to another to be reduced.
- Final assembly at the destination site.

A decade later, and with the continuous research and methods developed, a new technique called PERF (Partially Encoded Record Filter) was presented by Kenneth Ross [12]. This method is designed to minimize the cost of the "backward" phase of the two-way semi-join.

A few years later, William T. Bealor proposed a new algorithm called W that uses profitability and gain calculation results to help in choosing reducers [3]. He proved his algorithm to be an enhancement over AHY.

In this paper we present an improvement over AHY and W using PERF joins. The rest of the paper is organized as follows: section 2 overviews the PERF concept and presents our AHYPERF algorithm, section 3 describes our WPERF algorithm, section 4 presents a comparative example solved using the unoptimized, AHY, AHYPERF, W, and WPERF methods, section 5 provides the experimental results, and section 6 concludes the paper.

2- The AHYPERF Algorithm

Partially Encoded Record Filter is a new two-way semi-join implementation primitive. The basic idea of PERF is as follows:
Considering two relations *R* and *S*,
1. Project *R* on a joining attribute and get P_R.
2. Ship P_R to *S*.
3. Reduce *S* by a semi-join with P_R.
4. Send back to *R*, a bit vector (the PERF) that contains one bit for every tuple in P_R
   and in the same order. If the tuple is matching then send a 1 else send a 0.

The fourth step is known as the backward phase. The main utility of PERF is that it minimizes this phase and hence makes the forward phase (step 2) cost greater than the backward phase. PERF joins can be better enhanced by sending back to *R* not all the bit vector corresponding the P_R but only the 0s part or 1s part according to which one is less in size and hence has lower transmission cost.

When applying PERF to the original AHY algorithm the following is performed:
1. Perform all initial local processing.
2. Generate candidate schedules by isolating the attributes and creating simple queries.
3. For each relation R_i,
   a- Use algorithm SERIAL_PERF and create candidate schedules.
   b- Use procedure TOTAL_PERF to integrate candidate schedules.

2.1 - Algorithm SERIAL_PERF

1. Order relations R_i such that $S_1 \leq S_2 \leq \ldots \leq S_m$
2. If no relations are at the result node, then select strategy:
   $R_1 \rightarrow R_2 \rightarrow - - - R_n \rightarrow$ result node
   Or else if R_r is a result at the result node, then there are two strategies:
   $R_1 \rightarrow R_2 \rightarrow - - - \rightarrow R_r \rightarrow - - - R_n \rightarrow R_r$
   Or
   $R_1 \rightarrow R_2 \ldots \rightarrow R_{r-1} \rightarrow R_{r+1} \rightarrow \ldots R_n \rightarrow R_r$
   Select the one with minimum total time.
3. Build a PERF list where $PERF_{Ri\,Ri+1\,j}$ is set to 1 when transmission was done from R_i to R_{i+1} on join attribute *j*.
4. When calculating transmission cost,
   If $PERF_{Ri\,Ri+1\,j} = 1$ then
   Cost = 0
   Else
   $Cost = C_0 + C_1 * b_{ik} + (b_{ik} * \rho_{(i+1)\,k})/8$
   where
   $C_0 + C_1 * b_{ik}$ is the linear function of transmission cost that is equal to the fixed cost per byte transmitted (C_1) multiplied by the size in bytes of the join attribute projected. This is the usual cost of a semi-join known as the forward cost.

(b_{ik} * $\rho_{(i+1)\,k}$)/8 is the backward cost that is the cost of transmitting back to R_i the bit vector consisting of only matching values of the corresponding attribute. For simplicity of this equation, we are considering attribute *k* of width 1 byte.

5. Select the strategy with minimum total time.

2.2 - Algorithm TOTAL_PERF

1. Add candidate schedules: For each relation and candidate schedule, if the schedule contains a transmission of a joining attribute of the relation then add another similar schedule without the transmission of a joining attribute of the relation.
2. Calculate the cost of the newly added schedules as in step 4 of SERIAL_PERF.
3. Select the best candidate schedule to minimize each joining attribute total time.
4. Update the PERF List: Set to 1 the values of all transmissions of $BEST_{ij}$ selected.
5. Candidate Schedule Ordering: For each relation R_i, order the candidate schedules $BEST_{ij}$ on joining attribute d_{ij} so that,
   $ART_{i1}+C\ (s_1*SLT_{i1}) \leq \ldots \leq ART_{i\sigma}+C\ (s_i*SLT_{i\sigma})$
   where
   ART is the arrival time of the best schedule.
   SLT is the accumulated attribute selectivity of the best schedule.
   s is the selectivity of the corresponding relation.
6. Schedule Integration: For each $BEST_{ij}$ construct an integrated schedule to R_i consisting of parallel transmission of $BEST_{ij}$ and all schedules $BEST_{ik}$ where $k < j$.

As it can be seen, the PERF version of AHY algorithm does not eliminate redundant transmissions from the schedules but it makes their cost 0 when they occur. This can be made possible by adding a little overhead on the transmission cost, which is the backward cost. Using this fact, if a transmission was done from site *A* to site *B* using a join attribute *j*, then every other transmission from *A* to *B* using *j* will have a zero cost and every transmission from *B* to *A* using *j* will have also a zero cost. From this point, a PERF join can be seen as a non-redundant symmetric function. This fundamental property allowed us to enhance over the traditional AHY GENERAL algorithm.

We note that the reduction effect of PERF is proportional to the width of the attributes used. In Section 6, we show results from different width selections to clarify this issue.

2.3 - Complexity Analysis of AHYPERF Algorithm

As far as complexity is concerned, there was not a considerable increase in the complexity of AHYPERF algorithm since data will be still scanned in the same way and for the same number of times. Ordering will also be done in the same fashion. What is added is only the maintenance of the PERF list. According to its implementation, PERF list could be very easily maintained and with minimum complexity time. In our case, PERF list was implemented as a three-dimensional array. So, globally, and without loss of generality, we can assume that AHYPERF version of AHY algorithm takes no more than O (σm^2) where

σ is the number of different simple queries.
m is the number of relations in the query.

3 - The WPERF Algorithm

The main aim of W is to minimize total time by using reducers to eliminate unnecessary data. This algorithm is characterized by two distinct phases [3]:

1. Semi-join schedules for constructing each reducer are formed using a cost/benefit analysis based on estimated attribute selectivities and sizes of partial results.
2. Schedule is executed.

Before exposing the details of the heuristic, we note some assumptions and definitions needed. The following corollaries are needed:

A beneficial semi-join is the one for which its benefit exceeds its cost. The cost of a semi-join is expressed as the amount of inter-site data transfers needed to compute the operation while the benefit is the amount of data eliminated [4].
In general, the cost and benefit functions are defined as follows:

Cost: $C(R_i\text{-}d_{ik}\rightarrow R_j) = C_0 + C_1 * b_{ik}$

where

C_0: start-up cost for a transmission.
C_1: fixed cost per byte transmitted.

b_{ik}: size (e.g. in bytes) of the data item in attribute d_{ij}.

Benefit: $B(R_i\text{-}d_{ik} \rightarrow R_j) = S_j - S'_j$

where

S_j: size (e.g. in bytes) of relation R_j.
S'_j: size of relation R_j after reduction.

Hence, the benefit of a semi-join is the amount of data eliminated by reduction.
The profitability of a semi-join is defined as follows:
$P(d_{ij} \ltimes d_{kj}) = B(d_{ij} \ltimes d_{kj}) - C(d_{ij} \ltimes d_{kj})$.
Let $P(d_{ij})$ be the selectivity of d_{ij}, (i.e., the number of distinct values of d_{ij} ($R_i[d_{ij}]$) divided by all possible domain attribute values ($D[d_{ij}]$)). Hence, $\rho(d_{ij}) = R_i[d_{ij}] \ / D[d_{ij}]$.
In terms of $\rho(d_{ij})$,

$C(d_{ij} \ltimes d_{kj}) = C_0 + C_1 \times S(R_k)$
$B(d_{ij} \ltimes d_{kj}) = S(R_k) - S'(R_k) = S(R_k) - (S(R_k) * \rho(d_{ij}))$
$= S(R_k) * (1 - \rho(d_{ij}))$
$P(d_{ij} \ltimes d_{kj}) = B(d_{ij} \ltimes d_{kj}) - C(d_{ij} \ltimes d_{kj})$
$= S(R_k)*(1- \rho(d_{ij}) - (C_0 + C_1 * S(R_k))$
$= S(R_k) * (1- \rho(d_{ij}) -C_1) - C_0$

We define the marginal profit to the extra profit we can achieve by using one reducer instead of the other, as in the following formula:
$MP_{Ri}(d^*_{xj} \ltimes d_{yj}) = P(d_{yj}^* \ltimes R_i) - P(d^*_{xj} \ltimes R_i)$
The gain of semi-join is the sum of the profit and the marginal profit.
$G(d^*_{xj} \ltimes d_{yj}) = P(d^*_{xj} \ltimes d_{yj}) + MP(d^*_{xj} \ltimes d_{yj})$
A semi-join is said cost-effective when its gain > 0; hence, its benefit exceeds its cost and it has a marginal profit.

When applying PERF to W algorithm, the same concept is preserved but semi-joins are replaced by PERF joins. Our enhancement consisted of the following 2 phases that were added to the schedule construction:

1. Build a PERF list where $PERF_{Ri\,Ri+1\,j}$ is set to 1 when transmission was done from R_i to R_{i+1} on join attribute *j*.
2. When calculating transmission cost,

If $PERF_{Ri\,Ri+1\,j} = 1$ then
Cost = 0
Else
$Cost = C_0 + C_1 * b_{ik} + (b_{ik} * \rho_{(i+1)\,k})/8$
where
$C_0 + C_1 * b_{ik}$ is the linear function of transmission cost that is equal to the fixed cost per byte transmitted (C_1) multiplied by the size in bytes of the join attribute projected. This is the usual cost of a semi-join known as the forward cost.
$(b_{ik} * \rho_{(i+1)\,k})/8$ is the backward cost that is the cost of transmitting back to R_i the bit vector consisting of only matching values of the corresponding attribute. For simplicity of this equation, we are considering attribute *k* of width 1 byte.

As it can be seen, the PERF version of W algorithm does not eliminate redundant transmissions from the schedules but it makes their cost 0 when they occur. This can be made possible by adding a little overhead on the transmission cost, which is the backward cost. Note that the reduction effect of PERF is proportional to the width of the attributes used. In section 5, we show results from different width selections.

3.1 - Complexity Analysis of the WPERF Algorithm

As far as complexity is concerned, there was not a considerable increase in the complexity of WPERF algorithm over the original one, since data will be still scanned in the same way and for the same number of times. Ordering will also be done in the same fashion. What is added is only the maintenance of the PERF list. According to its implementation, PERF list could be very easily maintained and with minimum complexity time. In our case, PERF list was implemented as a three-dimensional array. So, globally, and without loss of generality, we can assume that WPERF version of W algorithm takes no more than $O(nm^2)$ where

n is number of common-joint attributes.
m is the number of attributes.

4 - A Comparative Example

Consider the following query: List the product number, name and total quantity for all parts that are currently on order from suppliers who supply that part to jobs 10 or 20.

The database used contains the following relations:
1. **PARTS** (P#, PNAME): This relation contains part numbers and names.
2. **ON_ORDER** (S#, P#, QTY): This relation contains supplier number, part number and quantity on order.
3. **S_P_J** (S#, P#, J#): This relation contains for each job number, the part numbers and from which suppliers they are.

Obviously our database is distributed and each relation resides at a different site. The two joining attributes are: P# and S#. The cost function to be used is: C(X) = 20 + X. It is a linear function in the form of y = aX + b where
a is the cost added per byte transmitted.
b is a fixed cost dependent on the network used. In this example b is taken as 20.

The corresponding size and selectivity relations are given in figure 1.

Ri	|Ri|	Si	di1=P# bi1	ρi1	di2=S# bi2 ρi2	
R1	70	1000	400	0.4	100	0.2
R2	140	2000	400	0.4	450	0.9
R3	150	3000	900	0.9	-	-

Figure 1. Relations Description.

For each relation we have as given:
|Ri|: cardinality of the relation (number of tuples).
Si : size of the relation in bytes.
dii : joining attribute.
bii : for joining attribute, the size, in bytes, of the column in the corresponding relation.
ρii : for each joining attribute, the corresponding selectivity.

4.1 The AHYPERF Algorithm

Applying AHYPERF to this query, two simple queries are formed for attributes d_{i1} and d_{i2}. In step 2, the following serial candidate schedules are formed:

For d_{i1},

d_{11}: d11 — 420 — C(400)

d_{21}: d11 — 420 — C(400); d21 — 180 — C(0.4 * 400)

d_{31}: d11 — 420 — C(400); d21 — 180 — C(0.4 * 400); d31 — 164 — C(0.4*0.4*900)

For d_{i2},

d_{12}: d12 — 120 — C(100)

d_{22}: d12 — 120 — C(100); d22 — 110 — C(0.2 * 450)

We will start the construction of the schedules for each relation.

Relation R_1:
Attribute d_{11}: The following schedules are added:

d'_{21}: d21 — 420 — C(400)

d'_{31}: d21 — 436 — C(400 + 140 * 0.9/8); d31 — 380 — C(0.4 * 900)

Each of the schedules of d_{11} is applied to R_1.

d_{21}: d11 — 424; d21 — 180; R1 — 420

Total time = C(400 + 70 * 0.4 / 8) + C(0.4 * 1000) + C(0.4 *2000)
= 424 + 180 + 420 = 1024

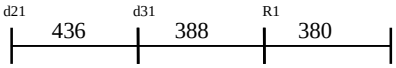
d_{31}: d11 — 424; d21 — 196; d31 — 164; R1 — 380

Total time = C(400+70*0.4/8)+C(0.4*400+140* 0.9/8)+C(0.4*0.4*900)+ C(0.4*0.9*
1000) = 424 + 196 + 164 + 380 = 1164

d'_{21} :

Total time = C(400 + 140 * 0.418) + C(0.4 * 1000)
= 427 + 420 = 847

d'_{31} :

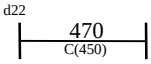


Total time = C(400+140*0.9/8)+C(0.4*900+150*0.4/8)+ C(0.4*0.9*1000)
= 436 + 388 + 380 = 1204

Choosing the minimum time schedule, we find that $BEST_{11}$ is d'_{21} with time 847.

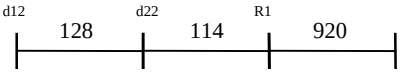
Attribute d_{12}: The following candidate schedule added:

d'_{22} :



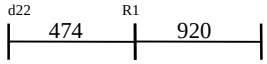
Each of the schedules of d_{12} is applied to R_1.

d_{22} :



Total time = C(100+70*0.9/8)+ (0.2*450+140*0.2/8)+C(0.9*1000)
= 128 + 114 + 920 = 1162

d'_{22} :

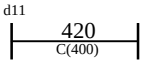


Total time = C(450 + 140 * 0.2/8) + C(0.9 * 1000)
= 474 + 920 = 1394

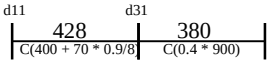
We find $BEST_{12}$ is d_{22} with time 1162.
Finally, for R_1, we choose $BEST_{11}$ with time 847.
At this stage we update $PERF_{2-1,\,1} = 1$ and $PERF_{1-2\,,\,1} = 1$.

Relation R2:
Attribute d_{11}:

d'_{21} :

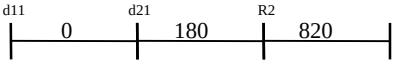


d'_{31} :



Each of the schedules of d_{11} is applied to R_2.

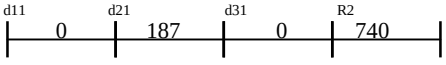
d_{21} :



Total time = 0 + C(0.4 * 400) + C(0.4 * 2000)
= 180 + 820 = 1000

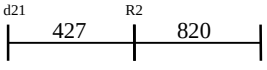
Note that transmission from 1–2 on attribute 1 is 0 because $PERF_{1-2,\,1} = 1$.

d_{31} :

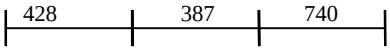


Total time = 0+C(0.4*400+140*0.4*0.9/8)+0+C(0.4*0.9*2000)
= 187 + 740 = 927

d'_{21} :



Total time = C(400 + 140 * 0.4/8) + C(0.4 * 2000)
= 427 + 820 = 1247

d'_{31} 

Total time = C(400+70*0.9/8)+C(0.4*900+150*0.4*0.9/8)+C(0.4*0.9*2000)
= 428 + 387 + 740 = 1555

We find $BEST_{21}$ is d_{31} with total time 927 and $BEST_{22}$ d'_{22} with total time 1948.
Finally, for R_2 we choose d_{31} with total time 927.
$PERF_{2-3,1} = 1$ and $PERF_{3-2,1} = 1$
Applying for relation R_3 we get $BEST_{31}$ with total time 740.
Hence, the most optimal total time with PERF algorithm for this query is:
847 + 927 + 740 = 2514.
Contribution: (2880 – 2514) / 2880 = 12.7% where
Contribution = (Initial time - Enhanced time) / Initial time
and in our case the initial time is the AHY time and the enhanced time is AHYPERF time.

4.2 The WPERF algorithm

We first establish schedules for construction of the two reducers:
Reducer for d_{i1}: The first semi-join is considered: $d_{11} \ltimes d_{21}$ with cost = $S(d_{11})$ = 420 and benefit = $S(R_2)$ – ($S(R_2)$ * $\rho(d_{11})$) = 1220.
Hence, the marginal profit for R_3:
MP_{R3} = $S(R_3)$ * ($\rho(d_{11})$ – $\rho(d_{21})$ + $S(d_{11})$–$S(d_{21})$
= 3000 * (0.4 – 0.4 * 0.4) + 420 – 180
= 3000 * 0.24 + 240 = 720 + 240 = 960

Since both profit and marginal profit are positive, this semi-join is added to the schedule. Next, the semi-join d^*_{21} d_{31} is examined with cost 180 and benefit 2540. The marginal profit of this semi-join with respect to R_1 is:
MP_{R1} = 1000 * (0.4 – 0.4 * 0.8) + 180 – 164
= 1000 * 0.04 + 16 = 56

Again, both profit and marginal profit are positive, so the semi-join is added.

So the reducer is d^*_{31}. It is constructed b the following schedule.

$$d_{11} \xrightarrow{400} d_{21} \xrightarrow{160} d_{31}$$

Reducer for d_{i2}: The only semi-join to be considered is $d_{12} \rtimes d_{22}$ where the cost is 120 and the benefit is 1620. The marginal profit with respect to R_1 is:
MP_{R1} = 1000 * (1 – 0.9) – 90
= 1000 * 0.1 – 90 = 100 – 90 = 10

So the schedule for constructing d^*_{22} is:

$$d_{12} \xrightarrow{100} d_{22}$$

The second phase considers the reduction effect of the reducers starting with the smallest. Calculations are based on the fact that the construction of the reducer will reduce certain relations. The reduction $d^*_{22} \rightarrow R_1$ is considered first with cost 128 and benefit 120 (because of the backward cost added) and then we do not need again to send $d^*_{22} \rightarrow R_1$ as we already have this information in the bit vector. So, the cost of the previous transmission will become 0. This semi-join is appended to the schedule. Next, the reducer d^*_{33} is considered but now we have the following:
- $S(R_1)$=900 : because R_1 has been reduced by d^*_{22}.
- $S(R_2)$=160 : reduced during the construction of the reducers.
- $S(R_3)$=480 : reduced during the construction of the reducers.

Hence, the reduction $d^*_{33} \rightarrow R_2$ has a cost of 164 and a benefit of 596. It is also appended to the schedule.
Therefore, the final schedule for execution is:

$d_{11} \xrightarrow{420} d_{21} \xrightarrow{180} d_{31}$ $\quad$ $d^*_{31} \xrightarrow{164} R_1$ $\quad$ $R_1 \xrightarrow{344} QS$

$R_2 \xrightarrow{180} QS$

$d_{12} \xrightarrow{128} d_{22}$ $\quad$ $d^*_{22} \xrightarrow{0} R_1$ $\quad$ $R_3 \xrightarrow{500} QS$

Contribution: (2018 – 1928) / 2018 = 4.46% where:
Contribution = (Initial time - Enhanced time) / Initial time
and in our case the initial time is the W time and the enhanced time is WPERF time.

5 - Experimental Results

In order to conduct experiments, many were built:

- A parametric program to generate the dataset definitions randomly with different domain ranges and different attribute selectivities. The user enters the maximum number of relations and the maximum number of attributes per relation. From this data, the program uses random functions to define the domain ranges and attribute selectivities (noting that for attribute selectivities the range should be between 0.1 and 0.9).
- A parametric program to generate the relation files from the dataset definitions above. This program creates a file for each relation and inserts into it a number of tuples equals the random cardinality of this relation. The tuple is a series of numbers generated randomly.
- A program implementing the discussed algorithms that will read the dataset definition files and the relation files and will simulate an inter-site data transfer and will calculate the cost of each transfer and then the total cost.

Because datasets are build randomly but within our specific ranges, we could consider that the runs for each scenario were able to represent a meaningful study sample for the cases.

Different scenarios were conceived in order to evaluate the performance of the different algorithms and for each scenario programs were run 1500 times. Note that all programs were developed using Visual C++ 4.0 under Windows 95. Experiments were conducted on a Pentium V PC with 64 MB RAM.

5.1 - Scenario 1:

In this scenario the attribute width is taken as 1 byte for all attributes. Table 1 contains the results where the TYPE field represents the number of relations involved in the join and the number of maximum join attributes, the Unoptimized field represents the case where all the data is shipped from one site to the other, and the rest of the fields represent the algorithms under study.

TYPE	Unoptimized	AHY	AHYPERF	W	WPERF
2-2	15166.54	11281.77	10244.36	10555.78	10045.63
2-3	20176.99	12514.91	11210.69	11309.97	10512.57
2-4	25324.95	13195.31	11518.01	11111.44	10027.33
3-2	20675.39	15610.47	14097.76	14015.49	13628.02
3-3	25983.89	16727.14	15110.53	14835.10	14170.03
3-4	30977.84	17211.87	15422.78	14457.87	13581.98
4-2	26656.58	18105.75	16167.82	15243.94	15022.64
4-3	33149.04	20203.26	18039.28	16939.22	16472.85
4-4	39281.91	21067.12	18746.14	16971.68	16319.74
5-2	32613.54	19919.32	17448.54	15306.96	15229.38
5-3	41224.43	22975.32	20212.85	18000.69	17713.47
5-4	48771.57	24180.39	21166.99	18512.63	18039.93
Average:	30000.22	17749.39	15782.15	14771.73	14230.30

Table 1. Comparison Table.

Graphically, the results are shown in figure 2. We notice that WPERF outperforms the other methods in all cases.

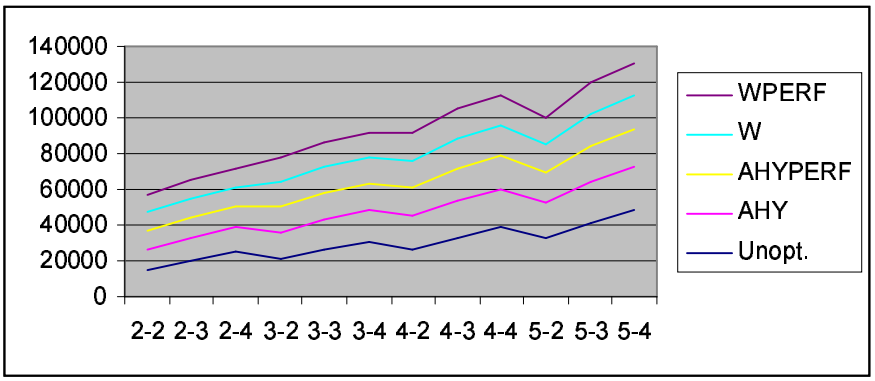


Figure 2. Comparison Results.

5.2- Scenario 2:

In this scenario the attribute width is taken as 5 bytes for all attributes. The results for this experiment are displayed in Table 2.

TYPE	Unoptimized	AHY	AHYPERF	W	WPERF
2-2	77094.85	57045.23	51343.36	55590.24	52984.34
2-3	100824.51	61996.65	54589.02	58207.49	54231.05
2-4	128113.97	66652.65	57004.79	57770.61	52240.99
3-2	102374.75	77600.45	69121.66	71389.21	69357.14
3-3	126861.52	83015.71	73938.19	74782.07	71954.99
3-4	154160.29	85602.33	75057.11	72123.73	67829.88
4-2	133156.13	91216.04	79985.85	77390.21	76317.44
4-3	165297.06	101571.73	89211.25	86298.51	83836.67
4-4	194790.69	105177.11	91628.56	84845.83	81522.69
5-2	162878.19	100808.91	85699.60	77760.06	77336.34
5-3	204638.73	114924.46	98263.82	89970.74	88519.65
5-4	244435.78	121233.30	103244.07	92136.69	89856.33
Average:	149552.2	88903.71	77423.94	74855.45	72165.63

Table 2. Comparison Table.

Graphically, the results are depicted in figure 3. We notice that AHYPERF and WPERF outperform their counterparts in all cases.

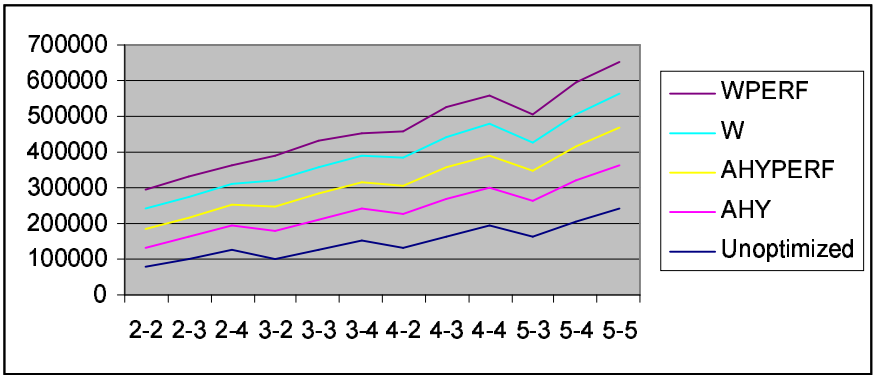


Figure 3. Comparison Results.

5.3 - Scenario 3:

In this scenario the attribute width is taken as 50 bytes for all attributes. The results are shown in Table 3.

TYPE	Unoptimized	AHY	AHYPERF	W	WPERF
2-2	774307.23	571559.9	512075.76	553557.3	526940.4
2-3	1025722.9	631131.25	553042.67	587713.6	547917.1
2-4	1286922.7	654111.03	555093.58	553746.6	499728.7
3-2	1034533.1	782855.95	695615.08	718546.9	698591.4
3-3	1272349.4	814956.83	719179.11	727393.5	694341.1
3-4	1558745	870137.82	758737.1	734627.2	691109.8
4-2	1343785.1	919758.2	800893.21	776510.1	765846.9
4-3	1636465.9	1007484	882273.47	852440.9	829994.7
4-4	1954307.2	1056676.8	914048.56	852391.6	818450.5
5-2	1622776.1	1005206.8	849381.99	768641.9	764375.7
5-3	2025497	1142050.7	975639.24	893408.6	878125.6
5-4	2471104.4	1227620.7	1041665.4	923914.8	901240.8
Average:	1500543	890295.84	771470.43	745241.1	718055.2

Table 3. Comparison Table.
Graphically, the results are shown in figure 4.

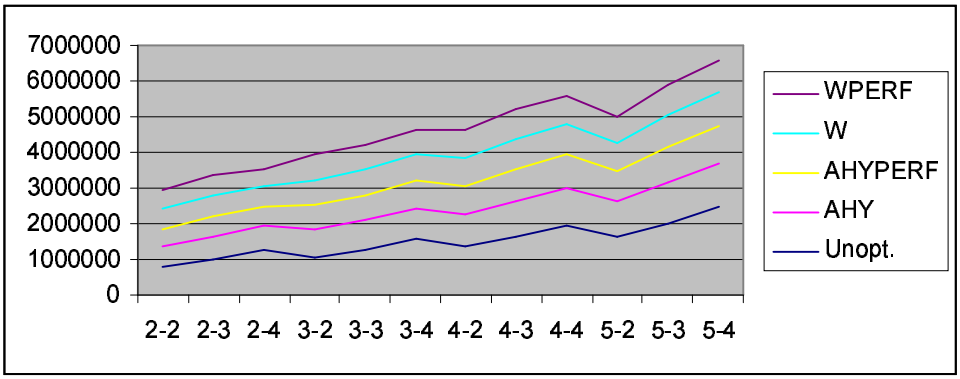


Figure 4. Comparison Results.

We used three different scenarios in order to study the performance of the mentioned algorithms from different perspectives. For each scenario, we compared the performance of the algorithms with respect to each other and with respect to the unoptimized solution. Using different scenarios we studied better the behavior of all algorithms under a variety of circumstances. We could be able to note that WPERF has the best performance for a field width of 50 bytes. This result was expected because of the overhead added by PERF to the backward phase. Remember that PERF concept consists of returning back to the original site a bit vector representing the matching tuples. This overhead is somehow more considerable when the original field width is less than 1 byte because it might be more profitable sometimes not to send back this data. But when having a width of 50 bytes, the backward cost becomes negligible as compared to the forward cost.

Finally, we can conclude that the results of our experiments were up to the expectations and proved the power of PERF joins and their advantage in optimizing the total time of distributed queries.

6 – Conclusion

In this paper, two algorithms have been presented as our contribution to the query optimization problem using semi-joins. We have fully exposed both concepts of semi-joins and PERF joins and then, we have taken two optimization algorithms using semi-joins (AHY and W) and enhanced them by applying PERF joins.

Theoretically, we have discussed the advantages of PERF joins over semi-joins which mainly consist of removing the cost associated with redundant transmissions by adding a relatively negligible cost to the backward phase of each PERF join.

Experimental results confirmed our expectations by showing a considerable enhancement over the original algorithms. Different series of experiments were conducted, allowing us to study even better the efficiency of PERF joins from different perspectives and to consider the best case for which PERF joins perform at most.

However, based on the fact that during the query processing, data in the relations should not be updated without updating the PERF accordingly and because not much work has been done until now to deal with this problem, we

view PERF joins as the best solution for distributed query optimization that can be adapted for huge, static warehouses where data is not changed very frequently.

Finally, we would like to mention that the optimization field is still an active research field and many new other techniques are being proposed. We studied thoroughly one new strategy and we applied it but there are still many others waiting for further study to prove their unique characteristics and advantages as well as their drawbacks.

References

[1] Apers, P., Hevner, A., and Yao, A. Optimization Algorithms For Distributed Queries in *IEEE Transactions on Software Engineering*, Vol. Se-9, No.1. 1983. pp. 57-68.

[2] Barbara, D., DuMouchel, W., Faloustos, C., Haas, P. J., Hellerstein, J., Iaonnidies, Y., Jagadish, H., Johnson, T., Ng, R., Poosala, N., Ross, K., and Sevcik, K. The New Jersey Data Reduction Report. *Bulletin of the Technical Committee on Data Engineering*, 1997. pp. 3-45.

[3] Bealor, T. Semi-Join Strategies For Total Cost Minimization in Distributed Query Processing. *Master Thesis*, University of Windsor, Canada. 1995.

[4] Bernstein, P., Goodman, N., Wong, E., Reeve, C., and Rothnie, J. Query Processing in a System For Distributed Databases (SDD-1) in *ACM Transactions on Database Systems*, Vol. 6, No. 4. 1981. pp. 602-625.

[5] Chatziantoniou, D., and Ross, K. GroupWise Processing of Relational Queries in *Proceedings of the 1997 VLDB Conference*, 1997. pp. 476-485.

[6] Chen, A., and Li, V. Improvement Algorithms For Semi-join Query Processing Programs In Distributed Database Systems in *IEEE Transactions on Computers*, Vol. C-33, No.11, 1984. pp. 959-967.

[7] Hevner, A., Wu, O., and Yao, S. Query Optimization on Local Area Networks in *ACM Transactions on Office Information*, Vol. 3, No. 1, 1985. pp. 35-62.

[8] Kang, H., and Roussopoulos, N. Using 2-Way Semi-joins in Distributed Query Processing in *Proceedings of the Third International Conference on Data Engineering*, 1987. pp. 644-651.

[9] Karwin, B. InterBase Server Configuration And Optimization. *Borland Developer's Conference*. 1996.

[10] Lei, H., and Ross, K. Faster Joins, Self-Joins and Multi-Way Joins Using Join Indices in *International Workshop on Next Generation Information Technologies and Systems*. 1997.

[11] Levy, A., Srivastava, D., and Kirk, T. *Data Model and Query Evaluation in Global Information Systems*. AT&T Bell Laboratories. 1991.

[12] Li, Z., and Ross, K. PERF Join: An Alternative to Two-Way Semi-Join and Bloomjoin. *Technical Report*. *Columbia University*, New York. 1995.

[13] Li, Z., and Ross, K. Fast Joins Using Join Indices in *VLDB Journal*, Vol. 8, No. 1, 1999. pp. 6-12.

[14] Liu, C., and Chen, H. A Hash Partition Strategy for Distributed Query Processing. *Technical Report. De Paul University*, Chicago. 1995.