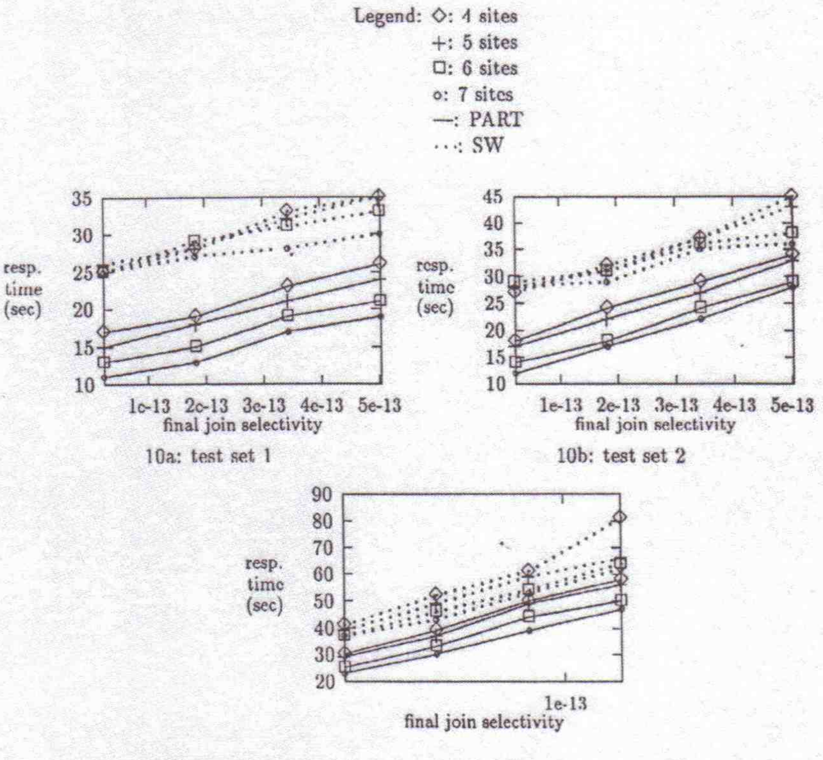


10a: test set 1

10b: test set 2

10c: test set 3

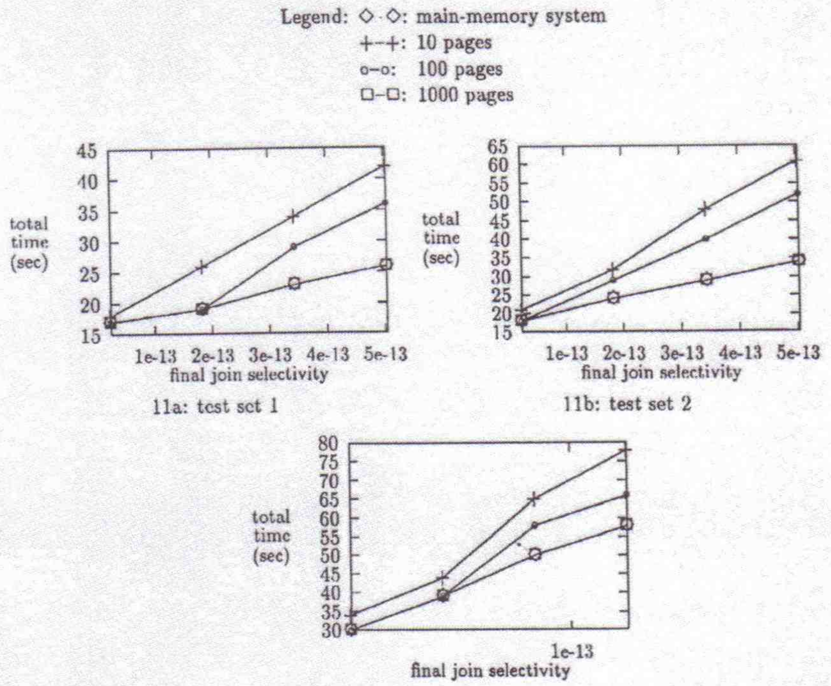
Figure 10. Response time of PART algorithm compared to SW.

The experiments were performed with the same 4 relations of test sets 1–3, but we varied the number of total sites available from 4 to 7. Thus, with 4 sites, PART does not perform any partitioning. The results of these experiments are summarized in figure 10. In comparison with SW, we achieved the following response time improvements: 18–33% for 4 sites, 15–42% for 5 sites, 19–52% for 6 sites, and 24–57% for 7 sites. Notice that increasing the number of sites does not guarantee a lower response time for the SW method for certain join selectivity values. The improvement of response time in PART versus PIPE_CHQ, i.e., the gains obtained by additional sites, were as follows: 3–10% for 5 sites, 13–24% for 6 sites, and 19–37% for 7 sites. It is clear that the speed-up in the response time of PART is not optimal with respect with the number of processors, but the performance of PART is more stable than that of SW.

5.3. *Disk-based systems*

In order to assess the effectiveness of our algorithm for disk-based systems where graph partitioning needs to be performed, we repeated the experiments based on test sets 1–3

23



11a: test set 1

11b: test set 2

11c: test set 3

Figure 11. Total time comparison: main-memory vs. disk-based PIPE_CHQ.

while varying the amount of main memory available for the graphs. We experimented with three different settings for the size of the main memory buffer available at each site (including the query site): 10, 100 and 1000 pages respectively, each page being of size 1 Kbyte. In Section 2 we presented two variants of PIPE_CHQ, intended for the extreme cases where all graphs fit in main memory or none of them fit. In practice, we also encounter a number of hybrid cases. One such case occurs when each bipartite graph BG_{R_{i-1},R_i} fits in main memory at site S_i, but at the query site there is not enough space to store all reduced bipartite graphs. In this case, step 4 of the disk-based PIPE_CHQ is modified, since no page identifiers are present. A second hybrid configuration occurs when some bipartite graphs fit in main memory at their respective sites. At each site S_i the inclusion of $page(id_{i-1})$'s in BG_{R_{i-1},R_i} depends upon whether the graph at S_{i-1} is main memory resident or not.

Figure 11 compares the total processing times of PIPE_CHQ in a main memory system with those for disk-based systems with various buffer sizes. In all test cases run, a disk-based system with buffers of size 1000 pages behaved identically to a main memory system: all graphs fit in memory; hence, no deterioration in performance occurred. In the case of buffers of size 10 pages, on the average, 31% of the graphs fit in main memory, while for 100 pages the figure increased to 92%. For buffers of size 10 pages we observed a performance deterioration in the range of 6–79% as compared to a main memory system, while for buffers of size 100 pages, this figure dropped to 0–53%. The 53% deterioration occurred for the highest join selectivity in test case 2 where 52% of the graphs fit in the main memory. In [24] we also compare the performance of our disk-based PIPE_CHQ with the RK and SJ algorithms and show that our algorithm maintains its superiority.

6. Discussion and conclusion

We have developed efficient algorithms for distributed join processing that make use of bipartite graphs in order to represent the joinability of two relations with respect to a given reduction set. Our algorithms achieve substantial savings in local processing time by performing a semijoin only once at each site and avoiding the generation of reduced relations. In addition, at the query site we can traverse these graphs easily in order to construct the implicit join. This also eliminates the need to send reduced relations to the query site and to store them there temporarily in order to perform an explicit join. All our algorithms are full reducers and can be applied to all cases, even to the pathological ones for which the forward and backward reduction phases do not eliminate any tuples.

Our algorithms are best suited for local area networks where the sites have large amounts of main memory and hence the bipartite graphs do not need to be written out to disk. However, we have shown that these algorithms can be extended for disk-based systems with only a minimal cost incurred for I/O.

Our algorithms are adaptive in number of contexts. Depending upon the objective minimization criteria desired, we can use a pipeline version for minimizing the total processing time or, we can choose between a partitioned pipeline version and a parallel version for response time minimization. Furthermore, the partitioned pipeline algorithm considers the number of additional sites available which gives it an adaptive flavor.

Our cost models developed in Appendix B relied on the knowledge of selectivity factors in order to estimate the amounts of data being transmitted and the sizes of the bipartite graphs. However, it is important to observe that these selectivity factors are not necessary for total processing time minimization. The cost models and hence, the selectivity factors, are only used for response time minimization in order to choose between PART and PAR_CHQ. In addition, even if the selectivity factors are not known in advance of running the response time optimization algorithm, they can be estimated quite efficiently and accurately via a sampling method, like the adaptive sampling method reported in [16].

In contrast to most partitioning schemes, PART divides the relations into equal fragments without considering the identity of the tuples. Most partitioning schemes assume the existence of a hash function that evenly distributes the tuples among the sites. This type of partitioning is difficult to achieve due to the skew in attribute values. We have reported on the results of simulation experiments that show that PART outperforms the partitioning method of [25] proposed for response time minimization; similarly PIPE_CHQ achieves substantial savings in total time over traditional semi- join methods or the tuple connector scheme of [21].

25

Appendix A: Parallel chain query algorithm(PAR_CHQ)

```
Step 1. parbegin
            At $S_i (i = 1, \ldots, n)$ do:
                if $(i < n)$ then
                  begin
                    $Temp_i = R_i(ID_i, A_i)$;
                    send $Temp_i$ to $S_{i+1}$;
                  end;
                if $(i > 1)$ then
                  begin
                    wait_for($Temp_{i-1}$);
                    perform $R_{i-1/\{i-1\}} \rtimes_{A_{i-1}} R_{i/\{i\}}$;
                    build $BG_{R_{i-1}/\{i-1\}, R_i/\{i-1,i\}}$;
                  end;
        parend;
Step 2. At $S_2$ do: /* Right_messages */
            Remove all $id_1$'s from $BG_{R_1,R_2}$ that do not have any edges incident to them;
            $Right\_messages_2 =$ set of $id_2$'s from $BG_{R_1,R_2}$ that do not have any edges
               incident to them;
            Remove $Right\_messages_2$ from $BG_{R_1,R_2}$ and send it to $S_3$;
        At $S_n$ do: /* Left_messages */
            Remove all $id_n$'s from $BG_{R_{n-1},R_n}$ that do not have any edges incident to them;
            $Left\_messages_n =$ set of $id_{n-1}$'s from $BG_{R_{n-1},R_n}$ that do not have any edges
               incident to them;
            Remove $Left\_messages_n$ from $BG_{R_{n-1},R_n}$ and send it to $S_{n-1}$;
Step 3. parbegin
            At $S_2$ do:
                wait_for($Left\_messages_3$);
                Remove $id_2$'s contained in $Left\_messages_3$ from $BG_{R_1,R_2}$ and remove the
                   edges incident to them;
                Remove $id_1$'s that do not have any edges incident to them from $BG_{R_1,R_2}$;
            At $S_i (i \neq 2, i \neq n)$ do:
                parbegin
                    begin
                        wait_for($Left\_messages_{i+1}$);
                        Remove $id_i$'s contained in $Left\_messages_{i+1}$ from $BG_{R_{i-1},R_i}$ and
                           remove the edges incident to them;
                        $Left\_messages_i =$ set of $id_{i-1}$'s from $BG_{R_{i-1},R_i}$ that do not have
                           any edges incident to them;
                        Remove $Left\_messages_i$ from $BG_{R_{i-1},R_i}$;
                        if ($Right\_messages_{i-1}$ processed)
                        then send $Left\_messages_i$ minus $Right\_messages_{i-1}$ to $S_{i-1}$;
                        else send $Left\_messages_i$ to $S_{i-1}$;
                    end;
```

26

```
                begin
                    wait_for($Right\_messages_{i-1}$);
                    Remove $id_{i-1}$'s contained in $Right\_messages_{i-1}$ from $BG_{R_{i-1},R_i}$
                      and remove the edges incident to them;
                    $Right\_messages_i =$ set of $id_i$'s from $BG_{R_{i-1},R_i}$ that do not have
                      any edges incident to them;
                    Remove $Right\_messages_i$ from $BG_{R_{i-1},R_i}$;
                    if ($Left\_messages_{i+1}$ processed)
                    then send $Right\_messages_i$ minus $Left\_messages_{i+1}$ to $S_{i+1}$;
                    else send $Right\_messages_i$ to $S_{i+1}$;
                end;
            parend;
        At $S_n$ do:
            wait_for($Right\_messages_{n-1}$);
            Remove $id_{n-1}$'s contained in $Right\_messages_{n-1}$ from $BG_{R_{n-1},R_n}$ and
              remove the edges incident to them;
            Remove $id_n$'s that do not have any edges incident to them from $BG_{R_{n-1},R_n}$;
    parend;
Step 4. The same as Step 4 in the algorithm PIPE_CHQ.
```

Appendix B: Computation of processing times and response times

As we mentioned earlier, we assume that an external merge-sort is used to perform the semijoin at each site. The local processing times in the formulae given below can be modified correspondingly if a different semijoin method is used. The parameters used in the Appendix are summarized in Table 4.

A. Pipeline algorithm

We shall compute the following components of the total processing time:

$S_{i,f}$ = time to construct BG_{R_{i-1},R_i}
$S_{i,b}$ = time to reduce the graph BG_{R_{i-1},R_i}
$T_{i,f}$ = data transmission time from S_i to S_{i+1} in forward reduction
$T_{i,j,b}$ = data transmission time from S_i to S_j in backward reduction
$T_{i,g}$ = graph transmission time from S_i to the query site
G_i = size of the graph after the forward reduction at S_i
GT = graph traversal time
$I_{i-1,f}$ = size of input data received at S_i from S_{i-1} in pages (in forward phase)
$I_{i+1,b}$ = size of input data received at S_i from S_{i+1} in pages (in backward phase)
T_{tar} = target attributes transmission time
$Total_{PIPE}$ = total time for pipeline algorithm
$Response_{PIPE}$ = response time for pipeline algorithm

In the forward reduction the processing times at each site are:

$$
\begin{aligned}
S_{1,f} &= (n_1 \cdot |R_1|)/P \times C_{i/o} \\
S_{i,f} &= I_{i-1,f} \times C_{i/o} + 2 \times I_{i-1,f} \times C_{i/o} + I_{i-1,f}/m \times C_{\text{sort}} \\
&\quad + \log_2(I_{i-1,f}/m) \times (n_{i-1} \times C_{\text{compare}} + 2 \times I_{i-1,f} \times C_{i/o}) \\
&\quad + I_{i-1,f} \times C_{i/o} + n_{i-1} \times C_{\text{compare}} + 2 \times P_i \times C_{i/o} \\
&\quad + P_i/m \times C_{\text{sort}} + \log_2(P_i/m) \times (n_i \times C_{\text{compare}} + 2 \times P_i \times C_{i/o}) \\
&\quad + P_i \times C_{i/o} + n_i \times C_{\text{compare}} + G_i/m \times C_{\text{sort}},
\end{aligned}
$$

where $I_{i-1,f} = (n_{i-1}(\prod_{k=1}^{i-2} f_{k,k+1}) \cdot |R_{i-1}(ID_{i-1}, A_{i-1})|)/P$, $P_i = (n_i \cdot |R_i|)/P$, $G_i = n_{i-1}n_i \prod_{k=1}^{i-1} g_k \times (|R_{i-1}(ID_{i-1})| + |R_i(ID_i)|)/P$, $(i = 2, \ldots, n)$.

At S_1, we need to read R_1 and send to S_2 its *ID*s and join attributes. The first term in $S_{i,f}$ $(i = 2, \ldots, n)$ denotes the I/O cost to store the incoming data received from S_{i-1}, while the next two terms account for the I/O costs and CPU costs in the sorting of the incoming data. The fourth term accounts for the merge costs in completing the external sort of the incoming data: a total of $\log_2(I_{i-1,f}/m)$ passes are required since we have $(I_{i-1,f}/\mathrm{m})$ runs after the sort step. The next two terms in $S_{i,f}$ account for the component of the time spent in finding matching tuples in the incoming data. The next 5 terms repeat the same calculation for R_i, the relation stored at S_i. The last term accounts for the sort cost of the graph assuming that the graph fits in main memory. If the graph does not fit in main memory, the sort cost can be represented similar to terms 3 and 4. In the backward stage the processing times at each site are:

$$
\begin{aligned}
S_{i,b} = 2 \times I_{i+1,b} \times C_{i/o} &+ \left(n_i \prod_{k=1}^{i-1} f_{k,k+1} - n_i \prod_{k=1}^{n-1} f_{k,k+1}\right) \\
&\times \log_2\left(n_{i-1}n_i \prod_{k=1}^{i-1} g_k\right) \times C_{\text{compare}},
\end{aligned}
$$

where $I_{i+1,b} = (n_i \prod_{k=1}^{i-1} f_{k,k+1} - n_i \prod_{k=1}^{n-1} f_{k,k+1}) \cdot |R_i(ID_i)|/P$, $(i = 2, \ldots, n-1)$.

The first term in $S_{i,b}$ accounts for storing and retrieving the incoming data from S_{i+1}. The second term represents the cost of doing a binary search for all ID's received from S_{i+1} to reduce the graph whose size is $n_{i-1}n_i \prod_{k=1}^{i-1} g_k$. The data transmission times in the forward and backward phases are:

$$
\begin{aligned}
T_{i,f} &= n_i\left(\prod_{k=1}^{i-1} f_{k,k+1}\right) \cdot |R_i(ID_i, A_i)|/TR, \ (i = 1, \ldots, n-1) \\
T_{i+1,i,b} &= \left(n_i \prod_{k=1}^{i-1} f_{k,k+1} - n_i \prod_{k=1}^{n-1} f_{k,k+1}\right) \cdot |R_i(ID_i)|/TR, \ (i = 2, \ldots, n-1)
\end{aligned}
$$

The time to transmit the final graph to the query site is:

$$
T_{i,g} = n_{i-1}n_i \times \prod_{k=1}^{n-1} g_k \times (|R_{i-1}(ID_{i-1})| + |R_i(ID_i)|), \ (i = 2, \ldots, n)
$$

The number of edges in each bipartite graph after the reduction is $n_{i-1}n_i \times \prod_{k=1}^{n-1} g_k$. The graph traversal time is:

$$GT = \left(\prod_{k=1}^{n} n_k \times \prod_{k=1}^{n-1} g_k\right)\left(\sum_{k=2}^{n-1} \log_2\left(n_k n_{k+1} \prod_{l=1}^{n-1} g_l\right) + n \times C_{i/o}\right)$$

For each tuple in the join result we need to perform $(n-2)$ binary search operations and, in the worst case, n I/Os to retrieve the target attributes. The number of tuples in the join result is $\prod_{k=1}^{n} n_k \times \prod_{k=1}^{n-1} g_k$. The total time $Total_{PIPE}$ is:

$$\begin{aligned} Total_{PIPE} = Response_{PIPE} = {} & \sum_{k=1}^{n} S_{k,f} + \sum_{k=2}^{n-1} S_{k,b} + \sum_{k=1}^{n-1} T_{k,f} + \sum_{k=2}^{n-1} T_{k+1,k,b} \\ & + \sum_{k=2}^{n} T_{k,g} + GT + T_{\text{tar}} \end{aligned}$$

B. Parallel Algorithm

In order to compute the response time of the parallel algorithm, we need to compute a number of additional processing terms as well as elapsed times. The elapsed time for an operation performed at S_i or for a message received at S_i accounts for the delays due to other operations/messages that need to be completed/received earlier.

$S_{i,rl}$ = time to reduce the graph BG_{R_{i-1},R_i} using *Left_messages*
$S_{i,rr}$ = time to reduce the graph BG_{R_{i-1},R_i} using *Right_messages*
$I_{i+1,rl}$ = size of input data received at S_i from S_{i+1} in left reduction
$I_{i-1,rr}$ = size of input data received at S_i from S_{i-1} in right reduction
$E_{i,s}$ = elapsed time to finish the semijoin at site S_i.
$E_{j,rm}$ = elapsed time to receive *Right_messages* from S_2 at S_j
$E_{j,lm}$ = elapsed time to receive *Left_messages* from S_n at S_j

The processing times for the semijoins performed during forward reduction are:

$$\begin{aligned} S_{1,f} &= (n_1 \cdot |R_1|)/P \times C_{i/o} \\ S_{i,f} &= I_{i-1,f} \times C_{i/o} + 2 \times I_{i-1,f} \times C_{i/o} + I_{i-1,f}/m \times C_{\text{sort}} + \log_2(I_{i-1,f}/m) \\ &\quad \times (n_{i-1} \times C_{\text{compare}} + 2 \times I_{i-1,f} \times C_{i/o}) + I_{i-1,f} \times C_{i/o} \\ &\quad + n_{i-1} \times C_{\text{compare}} + 2 \times P_i \times C_{i/o} + P_i/m \times C_{\text{sort}} \\ &\quad + \log_2(P_i/m) \times (n_i \times C_{\text{compare}} + 2 \times P_i \times C_{i/o}) \\ &\quad + P_i \times C_{i/o} + n_i \times C_{\text{compare}} + G_i/m \times C_{\text{sort}}, \end{aligned}$$

where $I_{i-1,f} = (n_{i-1} \cdot |R_{i-1}(ID_{i-1}, A_{i-1})|)/P$, $P_i = (n_i \cdot |R_i|)/P$, $G_i = n_{i-1}n_i g_{i-1} \times (|R_{i-1}(ID_{i-1})| + |R_i(ID_i)|)/P$, $(i = 2, \ldots, n)$

29

Note that the formula above is the same as the one used in the pipeline algorithm except for the different definitions of $I_{i-1.f}$ and G_i. In the backward phase, the processing times for *Left_messages* at each site are:

$$S_{i.rl} = 2 \times I_{i+1.rl} \times C_{i/o} + \left(n_i - n_i \prod_{k=i}^{n-1} f_{k+1.k}\right) \log_2(n_{i-1} n_i g_{i-1}) \times C_{\text{compare}},$$

where $I_{i+1.rl} = (n_i - n_i \prod_{k=i}^{n-1} f_{k+1.k}) \cdot |R_i(ID_i)|/P, (i = 2, \ldots, n-1)$

Similar to the pipeline algorithm computation, the first term in $S_{i,rl}$ accounts for the storing and retrieving of the incoming data from S_{i+1} and the next term stands for the cost of the binary search to be performed for all IDs received from S_{i+1} to reduce further the graph whose size is $n_{i-1} n_i g_{i-1}$. Similarly, processing time for *Right_messages* at each site is:

$$S_{i,rr} = 2 \times I_{i-1,rr} \times C_{1/O} + \left(n_{i-1} - n_{i-1} \prod_{k=1}^{i-2} f_{k,k+1}\right) \log_2(n_{i-1} n_i g_{i-1}) \times C_{\text{compare}},$$

where $I_{i-1,rr} = (n_{i-1} - n_{i-1} \prod_{k=1}^{i-2} f_{k,k+1}) \cdot |R_{i-1} i(ID_{i-1})|/P, (i = 3, \ldots, n)$

The elapsed times to finish the semijoin at each site are:

$$E_{i,s} = \max(n_{i-1} \cdot |R_{i-1}(ID_{i-1}, A_{i-1})|/TR,\ n_i \cdot |R_i|/P \times C_{i/o}) + S_{i.f}, \quad (i = 2, \ldots, n-1)$$

$$E_{n,s} = n_{n-1} \cdot |R_{n-1}(ID_{n-1}, A_{n-1})|/TR + S_{n,f}$$

We observe here that a semijoin at site S_i cannot be performed until we receive the incoming data from S_{i-1} and we send the output data to S_{i+1}. This is the reason for taking the maximum of these operations in $E_{i,s}$. Note that at S_n there is no data to be sent out in forward reduction. Let S_j $(2 \leq j \leq n)$ denote the meeting point of the left and right messages, i.e., the first site to receive both types of messages. Then $E_{j.lm}$ and $E_{j,rm}$ can be computed as follows:

$$E_{j.rm} = E_{j-1.s} + \left(n_{j-1} - n_{j-1} \prod_{k=1}^{j-2} f_{k.k+1}\right) \cdot |R_{j-1}(ID_{j-1})|/TR$$

$$E_{j.lm} = E_{j+1.s} + \left(n_j - n_j \prod_{k=j}^{n-1} f_{k+1.k}\right) \cdot |R_j(ID_j)|/TR$$

Let us denote by T_r and T_l the elapsed times for the right and left messages to travel to S_n and S_2 and be processed there, respectively. T_r and T_l can be computed by the following recurrence formulae:

```
Case 1: $E_{j,rm} < E_{j,lm}$
    for $i = j$ to $n - 1$ do
        begin
            $T_r = \max(E_{i,rm}, E_{i,s}) + S_{i,rr} + (n_i - n_i \prod_{k=1}^{i-1} f_{k,k+1}) \cdot |R_i(ID_i)|/TR$;
            $E_{i+1,rm} = T_r$;
        end;
    $T_r = \max(E_{n,rm}, E_{n,s}) + S_{n,rr}$;
    $E_{j,lm} = \max(E_{j,rm}, E_{j,s}) + S_{j,rr}$;
    for $i = j$ to 3 do
        begin
            $T_l = \max(E_{i,lm}, E_{i,s}) + S_{i,rl} + (n_{i-1} - n_{i-1} \prod_{k=j-1}^{n-1} f_{k+1,k}) \cdot$
            $|R_{i-1}(ID_{i-1})|/TR$;
            $E_{i-1,lm} = T_l$;
        end;
    $T_l = \max(E_{2,lm}, E_{2,s}) + S_{2,rl}$;
```

The above calculations are based on the following observations. Each site S_i can start processing its incoming *Right_messages*$_{i-1}$ only after it has received the message from S_{i-1} and it has finished its semijoin operation. Only after it has finished processing the incoming message *Right_messages*$_{i-1}$, can S_i proceed to send *Right_messages*$_i$ to site S_{i+1}. This explains the use of the maximum in the calculation of T_r. *Left_messages*$_{i+1}$ can be processed asynchronously from *Right_messages*$_{i-1}$; in this case, the left message is processed after its right counterpart. The calculations for case 2 are almost identical with those for case 1, with only one exception. Since in case 2, $E_{j,rm} > E_{j,lm}$, the right message is processed after the left message; the statement above $E_{j,lm} = \max(E_{j,rm}, E_{j,s}) + S_{j,rr}$ should be removed and the statement $E_{j,rm} = \max(E_{j,lm}, E_{j,s}) + S_{j,rl}$ should be inserted at the beginning of the algorithm. The response time *Response*$_{PAR}$ is:

$$Response_{PAR} = \max(T_r, T_l) + \sum_{k=2}^{n} T_{k,g} + GT + T_{\text{tar}}$$

C. Partitioned pipeline algorithm

Let l_i denote the partitioning level of relation R_i.

$$\begin{aligned} Total\_PT_i^{l_i} &= \text{ total partitioning cost for } R_i^{l_i} \\ PT_i^{l_i} &= \text{ cost to obtain one partition of } R_i^{l_i} \text{ at an extra site} \end{aligned}$$

These costs can be computed as follows:

$$\begin{aligned} Total\_PT_i^{l_i} &= (l_i - 1) \times PT_i^{l_i} \\ PT_i^{l_i} &= P_i \frac{1}{l_i} \times C_{i/o} + \frac{1}{l_i} n_i \cdot |R_i(ID_i, A_{i-1}, A_i)|/TR \\ &\quad + \frac{1}{l_i} n_i \cdot |R_i(ID_i, A_{i-1}, A_i)|/P \times C_{i/o}, \end{aligned}$$

where $P_i = n_i \cdot |R_i|/P$, $(l_i \neq 1)$

$$PT_i^1 = 0, (i = 1, \ldots, r)$$

The first term in $PT_i^{l_i}$ accounts for the cost of retrieving one partition at level l_i from disk; the second term stands for the data transmission cost and the third one for cost of storing this partition at the new site. The modified local processing costs during the forward reduction phase are:

$$\begin{aligned} S_{i,f}^{l_i} = {} & I_{i-1,f} \times C_{i/o} + 2 \times I_{i-1,f} \times C_{i/o} + I_{i-1,f}/m \times C_{\text{sort}} \\ & + \log_2(I_{i-1,f}/m) \times (n_{i-1} \times C_{\text{compare}} + 2 \times I_{i-1,f} \times C_{i/o}) \\ & + I_{i-1,f} \times C_{i/o} + n_{i-1} \times C_{\text{compare}} + 2 \times P_i^{l_i} \times C_{i/o} \\ & + P_i^{l_i}/m \times C_{\text{sort}} + log_2(P_i^{l_i}/m) \times \left(n_i \times C_{\text{compare}} + 2 \times P_i^{l_i} \times C_{i/o}\right) \\ & + P_i^{l_i} \times C_{i/o} + n_i \times C_{\text{compare}} + G_i^{l_i}/m \times C_{\text{sort}}, \end{aligned}$$

where $I_{i-1,f} = (n_{i-1}(\prod_{k=1}^{i-2} f_{k,k+1}) \cdot |R_{i-1}(ID_{i-1},\ A_{i-1})|)/P$, $P_i^{l_i} = (\frac{1}{l_i} n_i \cdot |R_i|)/P$, $G_i^{l_i} = \frac{1}{l_i} n_{i-1} n_i \prod_{k=1}^{i-1} g_k \times (|R_{i-1}(ID_{i-1})| + |R_i(ID_i)|)/P$, $(i = 2, \ldots, n)$.

Observe that this formula is almost identical to that of $S_{i,f}$ in the pipeline algorithm except for the terms $P_i^{l_i}$ and $G_i^{l_i}$, which take into account that the cardinality of R_i is $\frac{1}{l_i}$ of the original cardinality. The backward processing costs become:

$$\begin{aligned} S_{i,b}^{l_i} = {} & 2 \times I_{i+1,b} \times C_{i/o} + \frac{1}{l_i}\left(n_i \prod_{k=1}^{i-1} f_{k,k+1} - n_i \prod_{k=1}^{n-1} f_{k,k+1}\right) \log_2\left(\frac{1}{l_i} n_{i-1} n_i \prod_{k=1}^{i-1} g_k\right) \\ & \times C_{\text{compare}}, \end{aligned}$$

where$I_{i+1,b} = \frac{1}{l_i}(n_i \prod_{k=1}^{i-1} f_{k,k+1} - n_i \prod_{k=1}^{n-1} f_{k,k+1}) \cdot |R_i(ID_i)|/P$, $(i = 2, \ldots, n-1)$.

S_i receives only $\frac{1}{l_i}$ of the data in backward reduction and the size of its graph is also reduced by $\frac{1}{l_i}$. The response time of the partitioning algorithm $Response_{PART}$ is:

$$\begin{aligned} Response_{PART} = {} & \sum_{k=1}^{n} S_{k,f}^{l_k} + \sum_{k=2}^{n-1} S_{k,b}^{l_k} + \sum_{k=1}^{r} Total\_PT_i^{l_k} + \sum_{k=1}^{n-1} T_{k,f} \\ & + \sum_{k=2}^{n-1} T_{k+1,k,b} + \sum_{k=2}^{n} T_{k,g} + GT + T_{\text{tar}} \end{aligned}$$

Acknowledgment

We would like to thank Nick Roussopoulos for providing us with the relation generator for the Wisconsin benchmark.

32

References

1. P. Apers, A.R. Hevner, and S.B. Yao, "Optimization algorithms for distributed queries," IEEE Trans. on Software Engineering, vol. SE-9, no. 1, pp. 57–68, Jan. 1983.
2. P. Bernstein and D. Chiu, "Using semijoins to solve relational queries," JACM, vol. 28, no. 1, pp. 25–40, Jan. 1981.
3. P. Bernstein, N. Goodman, E. Wong, C. Reeve, and J.B. Rothnie, "Query processing in a system for distributed database(SDD-1)," ACM Trans. on Database Systems, vol. 6, no. 4, pp. 602–625, Dec. 1981.
4. D. Bitton, D. DeWitt, and C. Turbyfill, "Benchmarking database systems: A systematic approach," Proc. 9th Intl. Conf. on VLDB, pp. 8–19, Oct. 1983.
5. J. Chang, "A heuristic approach to distributed query processing," Proc. Intl. 8th Conf. on VLDB, Mexico City, 1982, pp. 54–61.
6. A.L.P. Chen and V.O.K. Li, "Improvement algorithms for semijoin query processing programs in distributed database systems," IEEE Trans. on Computers, vol. C-33, pp. 959–967, Nov. 1984.
7. J.S.J. Chen and V.O.K. Li, "Optimizing joins in fragmented database systems on a broadcast local network," IEEE Trans. on Software Engineering, vol. 15, no. 1, pp. 26–38, Jan. 1989.
8. M.S. Chen and P.S. Yu, "Interleaving a join sequence with semijoins in distributed query processing," IEEE Trans. on Parallel and Distributed Systems, vol. 3, no. 5, pp. 611–621, Sept. 1992.
9. D.M. Chiu and Y.C. Ho, "A method for interpreting tree queries into optimal semi-join expressions," Proc. 9th ACM SIGMOD Intl. Conf. on Management of Data, pp. 169–178, 1980.
10. D.M. Chiu, P.A. Bernstein, and Y.C. Ho, "Optimizing chain queries in a distributed database system," SIAM Journal on Computing, vol. 13, no. 1, pp. 116–134, Feb. 1984.
11. E.I. Chong, "Query optimization in distributed database systems and multidatabase systems," Ph.D. Dissertation, Dept. of Elec. Eng. and Computer Science, Northwestern University, Evanston, IL, June 1994.
12. R. Epstein, M. Stonebraker, and E. Wong, "Distributed query processing in a relational database system," Proc. 7th ACM SIGMOD Intl. Conf. on Management of Data, Austin, TX, 1978, pp. 169–180.
13. A.R. Hevner and S.B. Yao, "Query processing in distributed database systems," IEEE Trans. on Software Engineering, vol. SE-5, no. 3, pp. 177–187, May 1979.
14. Y. Kambayashi, M. Yoshikawa, and S. Yagima, "Query processing for distributed databases using generalized semijoins," Proc. 11th ACM SIGMOD Intl. Conf. on Management of Data, Orlando, FL, 1982, pp. 151–160.
15. S. Lafortune and E. Wong, "A state transition model for distributed query processing," ACM Trans. on Database Systems, vol. 11, no. 3, pp. 294–322, Sept. 1986.
16. R.J. Lipton, J.F. Naughton, and D.A. Schneider, "Practical selectivity estimation through adaptive sampling," Proc. 19th ACM SIGMOD Intl. Conf. on Management of Data, pp. 1–11, 1990.
17. W. Litwin, M.-A. Neimat, and D. Schneider, "LH-Linear hashing for distributed files," Proc. 22nd ACM SIGMOD Intl. Conf. on Management of Data, pp. 327–336, 1993.
18. H. Lu and M. Carey, "Some experimental results on distributed join algorithms in a local area network," Proc. 11th Intl. Conf. on VLDB, pp. 292–304, Aug. 1985.
19. P. Mishra and M.H. Eich, "Join processing in relational databases," ACM Computing Surveys, vol. 24, no. 1, pp. 63–113, March 1992.
20. M.T. Özsu and P. Valduriez, Principles of Distributed Database Systems, Prentice-Hall, 1991.
21. N. Roussopoulos and H. Kang, "A pipeline N-way join algorithm based on the 2-way semijoin program," IEEE Trans. on Knowledge and Data Engineering, vol. 3, no. 4, pp. 486-495, Dec. 1991.
22. P. Scheuermann and G. Gursel, "Asserting the optimality of serial SJRPs in processing simple queries in chain networks," Information Processing Letters, vol. 19, pp. 255–260, Nov. 1984.
23. P. Scheuermann and E.I. Chong, "Distributed join processing using bipartite graphs," Proc. The 15th IEEE Intl. Conf. on Distributed Computing Systems, pp. 387–394, May 1995.
24. P. Scheuermann and E.I. Chong, "Using bipartite graphs for efficient data reduction in distributed join algorithms," Technical Report, Northwestern University, Dept. of Electrical and Computer Engineering, 1996.
25. D. Shasha and T. Wang, "Optimizing equijoin queries in distributed databases where relations are hash partitioned," ACM Trans. on Database Systems, vol. 16, no. 2, pp. 279–308, June 1991.
26. P. Valduriez, "Join indices," ACM Trans. on Database Systems, vol. 12, no. 2, pp. 218–246, June 1987.

33

27. C.B. Walton, A.G. Dale, and R.M. Jenevein, "A taxonomy and performance model of data skew effects in parallel joins," Proc. 17th Intl. Conf. on Very Large Data Bases, pp. 537–548, Sept. 1991.
28. C. Wang, A.L.P. Chen, and S.-C. Shyu, "A parallel execution method for minimizing distributed query response time," IEEE Trans. on Parallel and Distributed Systems, vol. 3, no. 3, pp. 325–333, May 1992.
29. C.T. Yu, C. Chang, M. Templeton, D. Brill, and E. Lund, "On the design of a distributed query processing algorithm," Proc. 12th ACM SIGMOD Intl. Conf. on Management of Data, San Jose, CA, 1983, pp. 30–39.
30. C.T. Yu and C.C. Chang, "Distributed query processing," Computing Surveys, vol. 16, no. 4, pp. 399–433, Dec. 1984.
31. C.T. Yu, K. Guh, D. Brill, and A.L.P. Chen, "Partition strategy for distributed query processing in fast local networks," IEEE Trans. on Software Engineering, vol. 15, no. 6, pp. 780–793, June 1989.

34