distribute the subrelations to the extra sites. The algorithm determines for each relation R_i the partitioning level, denoted by l_i, that should be used. The partitioning level indicates the number of fragments into which R_i should be divided. Each iteration of the algorithm considers the addition of one extra site if the savings obtained in local processing costs outweigh the additional data partitioning costs. The complete partitioning algorithm is given below.

Partitioning Algorithm(PART):

for $i = 1$ to r do
  $l_i = 1$; /* l_i is the partitioning level of R_i */
for $k = 1$ to $(n - r)$ do begin
  for $i = 2$ to r do
    compute Gain($R_i^{l_i}$);
  Max_Gain = $\max_i$(Gain($R_i^{l_i}$));
  if (Max_Gain > 0) then
    $l_j = l_j + 1$; /* R_j is the relation for which Max_Gain is achieved */
end;
for $i = 1$ to r do
  partition R_i into l_i subrelations of equal size and distribute them to the extra sites;

where $\text{Gain}(R_i^{l_i}) = (S_{i,f}^{l_i} + S_{i,b}^{l_i}) - (S_{i,f}^{l_i+1} + S_{i,b}^{l_i+1}) + (PT_i^{l_i} - PT_i^{l_i+1})$.
$S_{i,f}^{l_i}$ and $S_{i,b}^{l_i}$ are the local processing costs at site S_i in forward and backward reduction given that R_i is at partitioning level l_i; $PT_i^{l_i}$ stands for the partitioning cost.

Given n sites and r relations (with $n > r$), the CPU time complexity of the algorithm is $O((n - r)^* r)$.

Example 6. Let us consider the query $R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4$, and let us assume that there are 6 sites in the network. Initially all relations are at partition level 1. The gains to be computed are $(S_{i,f}^1 + S_{i,b}^1) - (S_{i,f}^2 + S_{i,b}^2) - PT_i^2$ for i = 2,3,4. Note that PT_i^2 denotes the cost to obtain a partition containing half the tuples of R_i at a new site. Let us assume that Max-gain is the gain obtained by partitioning R_3 into two equal sized partitions, one at its original site S_3 and the second at the extra site S_5. Since we have one additional site available, the actual partitioning does not occur yet, but we update the partitioning level of R_3 to 2. In the next iteration the gains obtained by partitioning R_2 and R_4 are unchanged, but Gain(R_3^2) becomes: $(S_{3,f}^2 + S_{3,b}^2) - (S_{3,f}^3 + S_{3,b}^3) + (PT_3^2 - PT_3^3)$. Now PT_3^3 denotes the cost of obtaining a partition of R_3 containing one third of the tuples of R_3. It is important to observe here that we should only consider subdividing a relation into fragments of equal size. A partitioning into unequal fragments will imply that the largest fragment becomes the bottleneck with regard to local processing costs. Let us assume that Gain(R_3^2) is largest among the gains considered. Thus the algorithm terminates with R_3 at partitioning level 3 and all other relations at partitioning level 1. This means that the additional sites S_5 and S_6 will each receive a fragment of R_3 whose cardinality is 1/3 of the original cardinality of R_3.

The modified pipeline algorithm, which we shall denote by ***partitioned pipeline***, is almost identical to the original, except that it makes use of broadcasting in both reduction phases. Thus, for example, if R_{i+1} has to be reduced by R_i and R_{i+1} is fragmented among a number of sites, then R_i will send its tuples identifiers to all these sites. The cost model for the response time of the modified pipeline algorithm is given in Appendix B.

The ***adaptive*** algorithm for response optimization chooses the algorithm best suited for a particular system and data configuration as summarized below.

Algorithm adaptive response time:
Case 1: no extra sites available
    if ($Response_{PIPE} > Response_{PAR}$)
    then use Parallel Algorithm;
    else use Pipeline Algorithm;
Case 2: extra sites available
    if ($Response_{PIPE} < Response_{PAR}$)
    then use Partitioned Pipeline Algorithm;
    else if ($Response_{PART} < Response_{PAR}$)
        then use Partitioned Pipeline Algorithm;
        else use Parallel Algorithm;

5. Experimental results

We have implemented our experiments on a SUN SPARC-IPX workstation having 16 MB main memory. The test relations were created from the Wisconsin benchmark database [4], with some modifications as explained in Section 5.1 below. Our simulation experiments are based on actual runs which compute the I/O time and CPU time explicitly by making use of the UNIX 'time' facility. In the absence of an actual distributed database system, the communication time was calculated by measuring the amount of data being transmitted and dividing it by the actual bandwidth. In our experiments we assumed a local area network with a bandwidth of 10 Mb/sec. The cost model developed in Appendix B is used only in Section 5.2.3 in order to determine the partitioning levels of the relations in PART. Hence, it is important to distinguish between the estimated values of the output parameters which are used in the cost model and the output parameters given in the simulation experiments which are based on actual runs. The estimated values of the output parameters are used by the cost model for response time optimization in order to decide which version of the algorithm to use.

Our workstation was used when the load on the server was low. In order to discount the I/O delay caused by competing users, we replicated each experiment five times and took the best timing value as the result since the best timing value is the closest to the actual time taken when the experiment is run stand-alone. Each run was treated as a separate process; hence all the buffers were flushed before the start of the next run.

The experiments were performed with chain queries. They were further subdivided into two major categories: one, for which we assumed that the bipartite graphs fit in main memory and a second, geared toward disk-based systems. Since the Wisconsin benchmark