Revisiting Join Site Selection in Distributed Database Systems

Haiwei Ye[1], Brigitte Kerhervé[2], and Gregor v. Bochmann[3]

[1] Département d' IRO, Université de Montréal, CP 6128 succ Centre-Ville, Montréal Québec, Canada H3C 3J7
ye@iro.umontreal.ca

[2] Département d'informatique, Université du Québec à Montréal, CP 8888, succ Centre-ville, Montréal Québec, Canada H3C 3P8
Kerherve.Brigitte@uqam.ca

[3] School of Information Technology & Engineering, University of Ottawa P.O. Box 450, Stn A, Ottawa Ontario, Canada K1N 6N5
bochmann@site.uottawa.ca

Abstract. New characteristics for e-commerce applications, such as highly distributed data and unpredictable system nature, require us to revisit query processing for distributed database systems. As join operations involve relations over multiple sites, the site to carry out the join operation can have a significant impact on the performance. In this paper, we propose a new join site selection method. Our method is different from traditional methods in the following ways. First, the candidate sites are not restricted to the operand sites or query site as adopted by most database systems. Second, it considers dynamic system properties such as available bandwidth and system loads. We also conducted some experiments to offer a comparison between our method and the traditional one. The results show the advantage of our method under various system statuses.

1 Introduction

Given the explosive growth of data over the Internet and prevalence of web-based applications such as e-commerce, how to manage large volumes of data and provide fast and timely answers to queries become more challenging in today's information systems. Data are usually distributed across multiple locations in a distributed database system for the purpose of scalability and availability. As a result, a database query issued against a distributed database system generally requires retrieval of data from several locations to compute the final answer. Typical operations to combine the data from different locations are binary operations such as *join*. Even if techniques such as semi-joins have been introduced to reduce the cost of queries [1], where to perform the join is still of interest since it does have a great impact on the overall system performance, especially in today's Internet environment. The research dedicated to join site selection has not received adequate attention. Existing approaches of

H. Kosch, L. Böszörményi, H. Hellwagner (Eds.): Euro-Par 2003, LNCS 2790, pp. 342–347, 2003.

join site selection do not consider the dynamic nature of the Internet environment. We propose an approach to integrate user-defined QoS requirements into a distributed query processing environment [2]. Join site selection is regarded as one of the essential steps in our QoS-aware query processing.

In a recent survey [3], Kossmann summarized three site selection strategies for client-server architectures. Depending on whether to move the query to the data (execution at servers) or to move the data to the query (execution at clients), the strategy is called *query-shipping* or *data-shipping*. A *hybrid* approach of these two solutions is also possible. Traditionally [4, 5], two types of strategies were proposed: *Move-Small* and *Query-Site*. In the query-site strategy, all joins are performed at the site where the query was submitted. In the move-small strategy, for each join operation, the smaller (perhaps temporary) relation participating in the join is always sent to the site of the larger relation. Selinger and Adiba suggested another option in [5]. They mentioned that for a join of two given tables at different sites, they could move both tables to a "third" site yet to be specified. However, this "third" site strategy has not been completely studied and not been adopted by commercial database systems.

The above strategies for join site selection in a distributed environment are inadequate in the sense that they are only suitable in a static environment where network performance and load of the database server are fixed and predictable. This is obviously not the case for today's highly distributed and dynamic systems. In this paper, we address the issue of join site selection in distributed multidatabase systems deployed for e-commerce applications. We propose an approach where dynamic system properties such as the performance of the Internet and the load of the server are taken into account in cost-based decisions for join site selection. In the next section we describe our approach and we provide a performance analysis in Section 3.

2 Considering a Third-Site in Join Site Selection

In order to select the site where the join operation will be processed, we first build the set of candidate sites where we consider both operand sites as well as possible third sites. In this set, we then select the optimal site. This decision is based on the cost model we propose, where dynamic system properties are considered. A QoS monitoring tool is used to periodically collect system status information and to provide dynamic system properties to the query optimizer.

2.1 Building the Set of Candidate Sites

The key issue in site selection is to decide which site is the best (optimal) for each binary operator. The site selection process becomes complicated when several candidate sites are capable of handling the operator. In fact, the crucial question is how many candidates should be considered for the third site. There are three possible approaches to determine candidate sites:

1) Consider all the available sites in the system. This is simple but this will usually incur too much overhead for the optimizer.
2) We can shrink the above set to all the sites involved in this particular query. By considering these sites, we may benefit from the situation where the result of this join needs to be shipped to the selected third site for further calculation. However, if the number of locations involved in the query is large, we may have the same problem as above: too much optimization overhead.
3) We can apply some heuristics to further decrease the size of the candidate sites set. For example, we can restrict the third site for a particular join operator to its "close relatives", such as niece/nephew sites in the join tree.

In our approach, we combine options 2 and 3. A *threshold* for the number of sites is therefore used to describe the situation where option 3 should be used. That is,

Candidate sites for a join = { All the sites in the tree, if N < =*threshold*; Close relative (children and niece), otherwise

where N is the total number of sites involved in the query. The value of threshold should be derived from the experiment. In the following algorithm, we use a procedure *CandidateSitesSelection*() to represent this procedure.

2.2 Algorithm

The procedure of join site selection can be regarded as marking the site for each join node in the tree, usually done in a bottom-up fashion. Thus we can employ one of the standard bottom-up tree traversal algorithms for this purpose. In our algorithm, we use *post order* tree traversal to visit the internal nodes of the tree. We ignore the post order algorithm and only give the algorithm (SiteSelection()) to visit each node.

```
Algorithm: SiteSelection (treenode)
1.  {
2.  if (hasChild(treenode) == true ) {
3.    candidate_set[]  = CandidateSitesSelection (treenode);
4.    for each site_s in candidate_set
5.         cost [s] = Cost-node (treenode, site_s);//Use the cost
           model to compute the cost if the join is performed on this site
6.    min_site = select-min (cost[]);
7.    treenode.join_site = min_site;
8.    if (site_s is also marked as join site for another node m
        in treenode)
9.      Cost-node (m, site_s); //recalculate cost for node m under the new
          added load introduced by choosing site_s
10. }   // end if
11. }
```

The procedure of *SiteSelection*() picks up the join site based on the cost model and records the join site in the root node of the input tree. The procedure of *SiteSelection*() picks up the join site based on the cost model and records the join site in the root node of the input tree. In the algorithm (Line 8 and 9), we consider the impact of this site selection on other nodes of the tree. We then recalculate the cost for node *m*, taking

into account the additional load introduced by choosing *site_s*. Due to the optimization overhead that would increase if we recursively consider the impact, in the current algorithm, we do not check whether the site selection is still optimal. This is part of our future work.

2.3 Cost Model and QoS Information

The *Cost-node*() implements the cost model which provides an evaluation of the cost for each node in the query execution plan and enables the adaptation to the changing conditions in distributed systems. The cost function includes two major parts: the *local part* and the *network part*. The following formulas define our cost models to execute a join at a candidate site **s**:

Cost-node(treenode, site_s) = Local_cost $_{site\_s}$ + Network_cost;
Local_cost $_{site\_s}$ = join-cost (node.left, node.right, load $_{\mathbf{site\_s}}$);
Network_cost =max{ship-cost (node.left, site_s), ship-cost (node.right, site_s)};

ship-cost (node, site_s) = $node.data / bwd_{site\ i,\ site\_s}$ + delay $_{site\ i,\ site\_s}$;

where $bwd_{node.site,\ i}$ and $delay_{node.site,\ i}$ represent the available TCP bandwidth and delay, respectively, from the site of the node to site i.

The accuracy of our cost models relies on the up-to-date information of the current system status. Therefore, to keep track of the current dynamic performance information about the underlying network and database server, we use the QoS monitoring tools developed in our work [6].

3 Experimentation

In this section, we evaluate the performance of our join site selection strategy according to the framework proposed in the previous section. The objective of our experiment is to show that our strategy can adapt to workload changes (both server load and network load) and always provides a better performance as compared to traditional strategy. For the purpose of comparison, we also implement the traditional join site selection strategy where the chosen site is always the site where the larger table resides; we call this strategy "larger-site" strategy.

Two types of system loads are used for our measurement: network load and server load. For network load, we mainly focus on the available bandwidth as the indication of network congestion level. For server load, we concentrate on the CPU utilization as the indication of server load. We observed the TCP traffic using IPERF [7]. As a result, the network congestion levels are classified into 6 levels: level 0 (no congestion) to level 5 (highest congestion). Concerning the server load, we degrade the performance of one server by loading it with additional processes. Each process simply eats up CPU and competes with the database system for CPU utilization. Additional load is

quantified by the number of these processes spawned on a server. We categorize this load into 4 levels: *no load*, *low load*, *medium load*, and *high load.*

It should be noted that as an experimental prototype, our execution engine was designed for ease of implementation and was not tuned for performance. The main purpose is to demonstrate the feasibility of our ideas. We conducted a number of experiments on two-way join and collected performance data for the two sets of experiments identified previously. Figure 1 provides a comparison of various loads.

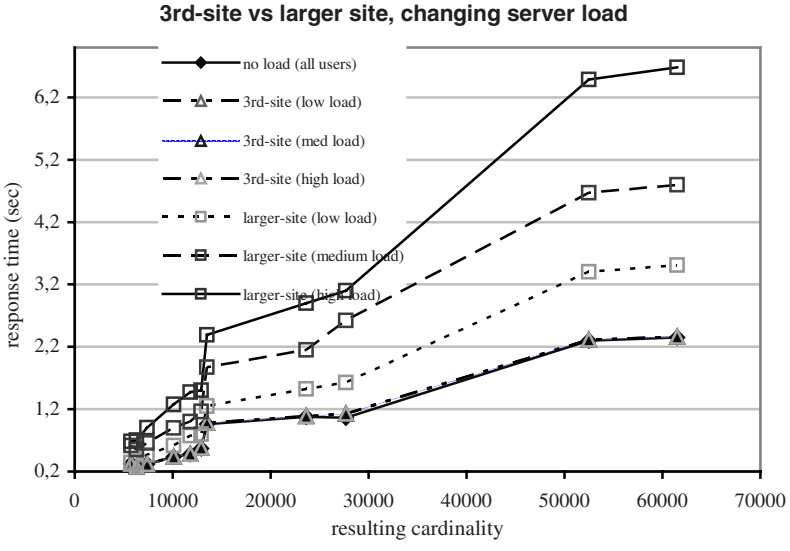

Fig. 1. Third-site vs. Larger-site with various server loads

The number of records in the resulting table (noted as resulting cardinality) varied from 180 to 3075. In the experiment, the network condition is the same for the entire load test, and the result in Figure 1 is collected while the network bandwidth is 5Mbps for all links. We only load the servers taking part in the joins and the rest remains unloaded. This open up the possibility to ship joined tables to other nodes for the purpose of performance gain.

We also conducted some experiments to compare the performance under different network congestion levels. In these tests, we assume that the links among the nodes involved in the join are congested while other links have the normal throughput (5Mbps). In addition, there is no load of the server during the experimental periods. The results show that, under the no load circumstance, the two strategies get the same performance. With the increasing load, the "third-site" selection strategy almost provides the same performance, while the "larger-site" strategy will experience higher response time. The advantage of third-site strategy increases with the increasing of server load. Traditional site selection fixes the join site to the larger site. As the network congestion level increases, the data transfer time will increase too, which will in turn affect the response time of the whole query. However, for the third site selection,

when the network congestion level reaches a certain point (usually at level 3, called shifting point), the algorithm will suggest to avoid the congested link and ship the two tables to a third site. This also explains why the response times (for the third-site algorithm) after the shifting point remain the same. In the performance study of the site selection problem, our third-site strategy not only reduces the response time but also achieves good load balancing among different database servers. According to our algorithm, if a server is heavily loaded, then the cost to perform a join operation on that server might be higher. This leads to the optimizer to avoid using that database server.

Both sets of experiments show the superiority of our "third-site" selection algorithm: it can pick up the fast response plan under different system conditions. In addition, it achieves good load balancing among different database servers. According to our algorithm, if a server is heavily loaded, then the cost to perform a join operation might be higher leading the optimizer to avoid using that server.

4 Conclusion

In this paper, we have proposed a new join site selection strategy for our QoS-aware distributed query processing environment. The strategy is based on cost models which integrate dynamic system properties. The candidate sites set considered in our approach is not restricted to the two operand sites as used in the traditional way. For the moment, we focus on the performance aspect of the QoS parameters provided by the QoS monitor. In the future, we will consider other QoS parameters, such as money cost or data quality.

References

1. Stocker, K., Kossmann, D., Braumandl, R., Kemper, A.: Integrating Semi-Join-Reducers into State of the Art Query Processors. ICDE 2001, pp. 575–584.
2. Ye, H., Kerhervé, B., Bochmann, G.v., Oria, V.,: Pushing Quality of Service Information and Requirements into Global Query Optimization, the Seventh International Database Engineering and Applications Symposium (IDEAS 2003), Hong Kong, China, July 16–18
3. Kossmann, D.: The state of the art in distributed query processing, ACM Computing Surveys (CSUR), Volume 32, Issue 4, December 2000, pp 422–469.
4. Cornell, D. W., Yu, P. S.: Site Assignment for Relations and Join Operations in the Distributed Transaction Processing Environment. ICDE 1988, pp. 100–108.
5. Selinger P. G., Adiba, M.:Access path selection in distributed data base management systems. In Proc. of the International Conference on Data Bases, 1980, pp. 204–215.
6. Ye, H.: Integrating Quality of Service Requirements in a Distributed Query Processing Environment, Ph.D thesis, University of Montreal, May 2003.
7. National Laboratory for Applied Network Research, http://www.nlanr.net/