

# Distributed Query Optimization by Query Trading

Fragkiskos Pentaris and Yannis Ioannidis

Dept. of Informatics and Telecommunications, University of Athens  
Ilisia, Athens 15784, Hellas(Greece)  
{frank,yannis}@di.uoa.gr

## Abstract

*Large-scale distributed environments, where each node is completely autonomous and offers services to its peers through external communication, pose significant challenges to query processing and optimization. Autonomy is the main source of the problem, as it results in lack of knowledge about any particular node with respect to the information it can produce and its characteristics, e.g., cost of production or quality of produced results. Inter-node competition is another source of the problem, as it results in potentially inconsistent behavior of the nodes at different times. In this paper, inspired by e-commerce technology, we recognize queries (and query answers) as commodities and model query optimization as a trading negotiation process. Query parts (and their answers) are traded between nodes until deals are struck with some nodes for all of them. We identify the key parameters of this framework and suggest several potential alternatives for each one. In comparison to trading negotiations for e-commerce, query optimization faces unique new challenges that stem primarily from the complex structure of queries. We address these challenges through a particular instantiation of our framework focusing on the negotiation strategy, the contents of the negotiation messages, the evaluation metrics of the queries, and the optimization algorithm run on a “buying” node. Finally, we conclude with some experiments that demonstrate the scalability and performance characteristics of our approach compared to those of traditional query optimization.*

## 1 Introduction

The success of the peer-to-peer (P2P) paradigm has clearly proven the advantages of outsourcing processing/searching workloads to thousands of nodes across the Internet. This approach to distribution satisfies the scalability and autonomy requirements of future data management environments, where large amounts of data are distributed among hundreds or even thousands of autonomous and possibly heterogeneous Database Management Systems (DBMS). In such an environment, finding the answer of a query will require splitting it into parts (sub-queries), retrieving the answers of these parts from remote “black-box” nodes, and merging the results together to calculate the answer of the initial query [14].

Query-processing techniques employed by P2P information retrieval systems cannot be used in databases federations, as these systems are limited to keyword-based searches. Furthermore, node autonomy and diversity result in lack of knowledge about any particular node with respect to the information it can produce and its characteristics, e.g., query capabilities, cost of production, or quality of produced results. Finally, if inter-node competition exists, it results in potentially inconsistent node behavior at different times. All these problems make traditional query optimization techniques [8, 9, 16] inappropriate [6, 11, 25].

In this paper, we consider a new approach to distributed query optimization in large federations of autonomous DBMSs. Inspired from microeconomics, we adapt e-commerce trading negotiation methods

to the problem. The result is a query-answers trading mechanism, where instead of trading goods, nodes trade answers of (parts of) queries in order to find the best possible query execution plan.

**Motivating example:** Consider the case of a telecommunications company with thousands of regional offices. Each of them has a local DBMS, holding customer-care (CC) data of millions of customers. The schema includes the relations `customer(custid, custname, office)`, holding customer information such as the regional office responsible for them, and `invoiceline(invid, linenum, custid, charge)`, holding the details (charged amounts) of customers' past invoices. For performance and robustness reasons, each relation may be horizontally partitioned and/or replicated across the regional offices. Consider now a manager at the Athens office asking for the total amount of issued bills in the offices in the islands of Corfu and Myconos:

```
SELECT SUM(charge)
FROM invoiceline i, customer c
WHERE i.custid=c.custid AND office in ('Corfu', 'Myconos');
```

The Athens node will ask the rest of the company's nodes whether or not they can evaluate (some part of) the query. Assume that the Myconos and Corfu nodes reply positively about the part of the query dealing with their own customers with a cost of 30 and 40 seconds, respectively. These offers could be based on the nodes actually processing the query, or having the offered result pre-computed already, or even receiving it from yet another node; whatever the case, it is no concern of Athens. It only has to compare these offers against any other it may have, and whatever has the least cost wins.

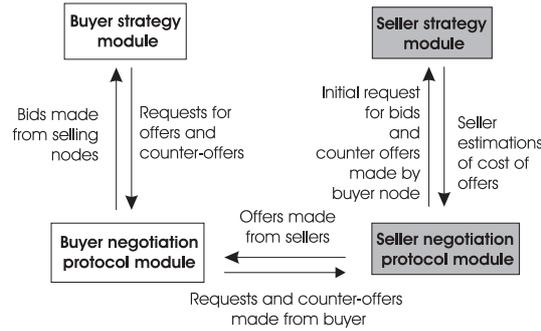
In this example, Athens effectively *purchases* the two answers from the Corfu and Myconos nodes at a cost of 30 and 40 seconds, respectively. That is, queries and query-answers are commodities and query optimization is a common trading negotiation process. The buyer is Athens and the potential sellers are Corfu and Myconos. The cost of each query-answer is the time to deliver it. In the general case, the cost may involve many other properties of the query-answers, e.g., freshness and accuracy, or may even be monetary. Moreover, the participating nodes may not be in a cooperative relationship (parts of a company's distributed database) but in a competitive one (nodes in the internet offering data products). In that case, the goal of each node would be to maximize its private benefits (according to the chosen cost model) instead of the joint benefit of all nodes.

In this paper, we present a complete query and query-answers trading negotiation framework and propose it as a query optimization mechanism that is appropriate for a large-scale distributed environment of (cooperative or competitive) *autonomous* information providers. It is inspired by traditional e-commerce trading negotiation solutions, whose properties have been studied extensively within B2B and B2C systems [2, 4, 17, 21, 26, 28], but also for distributing tasks over several agents in order to achieve a common goal (e.g., Contract Net [23]). Its major differences from these traditional frameworks stem primarily from two facts:

- A query is a complex structure that can be cut into smaller pieces that can be traded separately. Traditionally, only atomic commodities are traded, e.g., a car; hence, buyers do not know *a priori* what commodities (query answers) they should buy.
- The *value* of a query answer is in general multidimensional, e.g., system resources, data freshness, data accuracy, response time, etc. Traditionally, only individual monetary values are associated with commodities.

In this paper, we focus on the first difference primarily and provide details about the proposed framework with respect to the overall system architecture, negotiation protocols, and negotiation contents. We also present the results of an extended number of simulation experiments that identify the key parameters affecting query optimization in very large autonomous federations of DBMSs and demonstrate the potential efficiency and performance of our method.

To the best of our knowledge, there is no other work that addresses the problem of distributed query optimization in an environment of purely autonomous systems. Nevertheless, our experiments include a comparison of our technique with some of the currently most efficient techniques for distributed query optimization [6, 25].



**Figure 1. Modules used in a general trading negotiations framework.**

The rest of the paper is organized as follows. In section 2 we examine the way a general trading negotiation frameworks is constructed. In section 3 we present our query optimization technique. In section 4 we experimentally measure the performance of our technique and compare it to that of other relevant algorithms. In section 5 we discuss the results of our experiments and conclude.

## 2 Trading Negotiations Framework

A trading negotiation framework provides the means for buyers to request items offered by seller entities. These items can be anything, from plain pencils to advanced gene-related data. The involved parties (buyer and sellers) assign private valuations to each traded item, which in the case of traditional commerce, is usually their cost measured using a currency unit. Entities may have different valuations for the same item (e.g. different costs) or even use different indices as valuations, (e.g. the weight of the item, or a number measuring how important the item is for the buyer).

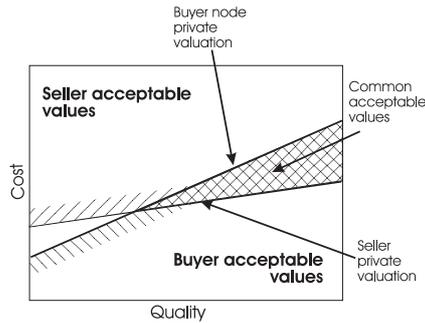
Trading negotiation procedures follow rules defined in a negotiation protocol. In each step of the procedure, the protocol designates a number of possible actions (e.g., make a better offer, accept offer, reject offer, etc.). Entities choose their actions, based on the strategy they follow and the expected surplus(utility) from this action, which is defined as the difference between the values agreed in the negotiation procedure and these held privately. Traditionally, strategies are classified as either *cooperative* or *competitive (non-cooperative)*. *In the first case, the involved entities aim to maximize the joint surplus of all parties, whereas in the second case, they simply try to individually maximize only their personal utility.*

Figure 1 shows the modules required for implementing a distributed electronic trading negotiation framework among a number of network nodes. Each node uses two separately modules, a *negotiation protocol* and a *strategy* module (white and gray modules designate buyer and seller modules respectively). The first one handles the inter-nodes message exchanges and monitors the current status of the negotiation, while the second one selects the contents of each offer/counter-offer. There are many different parameters affecting the behavior of these modules, including which negotiation protocol is used, which mechanism is used to reduce network congestions, what strategy is selected, how the private valuation of the traded items is calculated and, what the contracting details are. We briefly examine each of these parameters separately below.

### 2.1 Negotiation Protocol

The Negotiation protocol designates to the bargaining nodes, the valid actions in each stage of the negotiation (i.e., the rules of the negotiation), including the valid network messages, and how the winner(s) of the negotiations are selected. In [26], three principal forms of negotiation are defined: *bidding*, *auction*, and *bargaining*.

Bidding is the most simple negotiation protocol. The node interested in purchasing specific items asks for bids from the nodes that may be able to supply them. The selling nodes reply with offers that



**Figure 2. Negotiation solution space.**

include the properties (e.g. quantity, quality) of the items they can provide. The winning node(s) are the one(s) that make the best offers. The Contract Net Protocol (CNP) [23] is a good example of bidding.

Auctions are the second type of negotiation protocol. There are many different auction protocols, which differ in the way prices are quoted and in the manner in which bids are tendered. The most known one is the English (first-price) auction. Auctions are much more complex than direct bidding but have better efficiency in terms of accumulated sellers profit and buyers surplus, and smaller computational cost of the strategy used by the selling nodes [28].

Finally, bargaining is the third and probably the oldest negotiation mechanism. The buyer and selling nodes make alternating offer/counter-offers until an agreement is reached. The bargaining protocol is not as simple as the bidding protocol, and the strategy used is more complex than that of auctions, yet it is the only one allowing the negotiation of every property of an offer using counter-offers, like for instance, how many items will be purchased, their quality, and the delivery date. Bargaining with a large number of nodes is considered inefficient.

Distributed implementation of the previous negotiation protocols in e-commerce solutions have run into message congestion problems [23] caused by offers flooding. These problems can be solved using several different approaches such as agent-based architectures, focused addressing, audience restriction, use-based communication charges and, mutual monitoring [17, 23].

## 2.2 Strategies and Evaluation Metrics

The strategy that each node follows is *the set of rules that designate the exact action the node will choose, depending on the knowledge it has about the rest of the nodes*. As previously mentioned, strategies can either be cooperative, or competitive. Figure 2 shows the common acceptable values (solution) for an example of a two-issues (cost and quality) competitive bilateral bargaining. Both the buyer and the seller nodes have their private valuation of the bargained items. The buyer/seller node will not accept any offer or make any counter-offer above/bellow its private valuation. A proper strategy should ensure that the seller and buyer offers converge towards the common acceptable values.

In order for a strategy to work, there must be some way to rank two offers. The designer of the trading mechanism can define certain functions that map the values of the qualitative properties of the offers into real numbers, in such a way that better values are mapped into larger (or, depending on the parameter, smaller) numbers, and then specify a weighting aggregation function that maps these numbers into a single one measuring how good is an offer.

## 3 Distributed query optimization framework

Using the e-commerce negotiations paradigm, we have constructed an efficient algorithm for optimizing queries in large disparate and autonomous environments. Although our framework is more general, in this paper, we limit ourselves on select-project-join queries. This section presents the details of our technique, focusing on the parts of the trading framework that we have modified. The reader can find additional information on parts that are not affected by our algorithm, such as general competitive strategies

and equilibriums, message congestion protocols, and details on negotiation protocol implementations in [2, 4, 5, 15, 17, 21, 26, 28] and on standard e-commerce and strategic negotiations textbooks (e.g., [10, 12, 20]). Furthermore, there are possibilities for additional enhancements of the algorithm that will be covered in future work. These enhancements include the use of *contracting* to model partial/adaptive query optimization techniques, the design of a scalable *subcontracting* algorithm, the selection of advanced *cost functions*, and the examination of various *competitive and cooperative* strategies.

### 3.1 Overview

The idea of our algorithm is to consider queries and query-answers as commodities and the query optimization procedure as a trading of query answers between nodes holding information that is relevant to the contents of these queries. Buying nodes are those that are unable to answer some query, either because they lack the necessary resources (e.g. data, I/O, CPU), or simply because outsourcing the query is better than having it executed locally. Selling nodes are the ones offering to provide data relevant to some parts of these queries. Each node may play any of those two roles (buyer and seller) depending on the query been optimized and the data that each node locally holds.

Before going on with the presentation of the optimization algorithm, we should note that no query or part of it is physically executed during the whole optimization procedure. The buyer nodes simply ask from seller nodes for assistance in evaluating some *queries* and seller nodes make offers which contain their *estimated* properties of the answer of these queries (*query-answers*). These properties can be the total time required to execute and transmit the results of the query back to the buyer, the time required to find the first row of the answer, the average rate of retrieved rows per second, the total rows of the answer, the freshness of the data, the completeness of the data, and possibly a charged amount for this answer. The query-answer properties are calculated by the sellers' query optimizer and strategy module, therefore, they can be extremely precise, taking into account the available *network resources* and the *current workload* of sellers.

The buyer ranks the offers received using an administrator-defined weighting aggregation function (see sub-section 2.2) and chooses those that minimize the total cost/value of the query. In the rest of this section, the valuation of the offered query-answers will be the total execution time (cost) of the query, thus, we will use the terms cost and valuation interchangeably. However, nothing forbids the use of a different cost unit, such as the total network resources used (number of transmitted bytes) or even monetary units.

### 3.2 The Query-Trading Algorithm

The execution plans produced by the query-trading (QT) algorithm, consist of the query-answers offered by remote seller nodes together with the processing operations required to construct the results of the optimized queries from these offers. The algorithm task is to find the combination of offers and local processing operations that minimizes the valuation (cost) of the final answer. For this reason, it runs iteratively, progressively selecting the best execution plan. In each iteration, the buyer node asks (Request for Bids -RFBs) for some queries and the sellers reply with offers that contain the estimations of the properties of these queries (query-answers). Since sellers may not have all the data referenced in a query, they are allowed to give offers for only the part of the data they actually have. At the end of each iteration, the buyer uses the received offers to find the best possible execution plan, and then, the algorithm starts again with a possibly new set of queries that might be used to construct an even better execution plan.

The optimization algorithm is actually a kind of bargaining between the buyer node and the seller nodes. The buyer asks for certain queries and the sellers counter-offer to evaluate some (modified parts) of these queries at different values. The difference between our approach and the general trading framework, is that in each iteration of this bargaining the negotiated queries are different, as the buyer **and** the sellers progressively identify additional queries that may help in the optimization procedure. This difference, in turn, makes necessary to change selling nodes in each step of the bargaining, as these

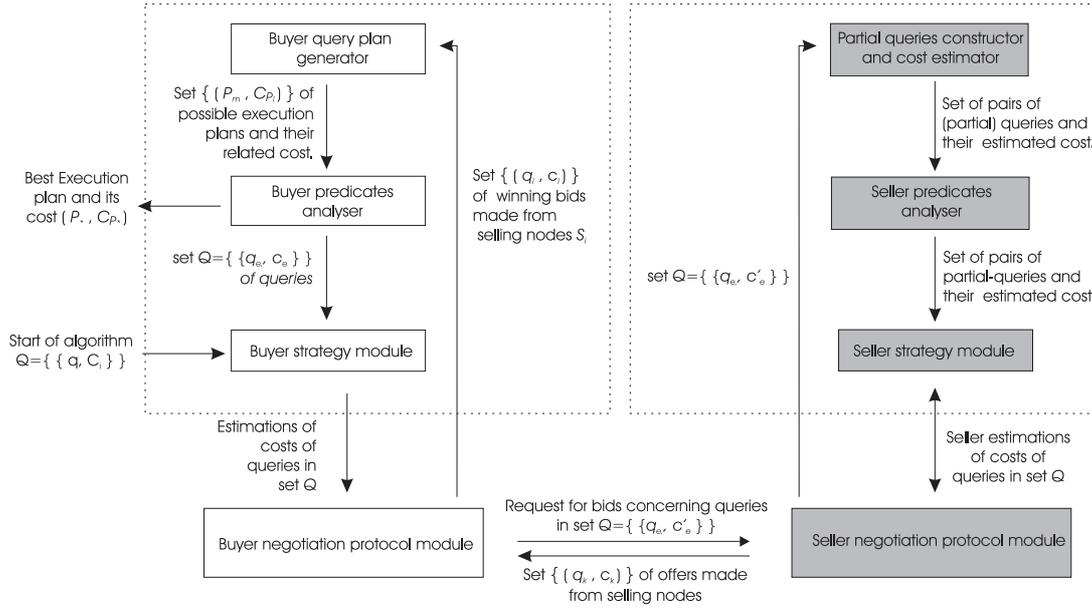
Buyer-side algorithm	Sellers-side algorithm
<p>B0. Initialization, set <math>Q = \{\{q, C_i\}\}</math></p> <p>B1. Make estimations of the values of the queries in set <math>Q</math>, using a trading strategy.</p> <p>B2. Request offers for the queries in set <math>Q</math></p> <p>B3. Select the best offers <math>\{q_i, c_i\}</math> using <b>one of the three</b> methods (bidding, auction, bargaining) of the query trading framework</p> <p>B4. Using the best offers, find possible execution plans <math>P_m</math> and their estimated cost <math>C_m</math></p> <p>B5. Find possible sub-queries <math>q_e</math> and their estimated cost <math>c_e</math> that, if available, could be used in step 4.</p> <p>B6. Update set <math>Q</math> with sub-queries <math>\{q_e, c_e\}</math>.</p> <p>B7. Let <math>P_*</math> be the best of the execution plans <math>P_m</math>. If <math>P_*</math> is better than that of the previous iteration of the algorithm, or if step B6 modified the set <math>Q</math>, then go to step B1.</p> <p>B8. Inform the selling-nodes, which offered queries used in the best execution plan <math>P_*</math>, to execute these queries.</p>	<p>S1. For each query <math>q</math> in set <math>Q</math> do the following:</p> <p>S2.1. Find sub-queries <math>q_k</math> of <math>q</math> that can be answered locally.</p> <p>S2.2. Estimate the cost <math>c_k</math> of each of these sub-queries <math>q_k</math>.</p> <p>S2.3. Find other (sub-)queries that may be of some help to the buyer node.</p> <p>S3. Using the query trading framework, make offers and try to sell some of the subqueries of step S2.2 and S2.3.</p>

**Figure 3. The distributed optimization algorithm.**

additional queries may be better offered by other nodes. This is in contrast to the traditional trading framework, where the participants in a bargaining remain constant.

Figure 3 presents the details of the distributed optimization algorithm. The input of the algorithm is a query  $q$  with an initially estimated cost of  $C_i$ . If no estimation using the available local information is possible, then  $C_i$  is a predefined constant (zero or something else depending on the type of cost used). The output is the estimated best execution plan  $P_*$  and its respective cost  $C_*$  (step B8). The algorithm, at the buyer-side, runs iteratively (steps B1 to B7). Each iteration starts with a set  $Q$  of pairs of queries and their estimated costs, which the buyer node would like to purchase from remote nodes. In the first step (B1), the buyer strategically estimates the values it should ask for the queries in set  $Q$  and then asks for bids (RFB) from remote nodes (step B2). The seller nodes after receiving this RFB make their offers, which contain query-answers concerning parts of the queries in set  $Q$  (step S2.1 - S2.2) or other relevant queries that they think it could be of some use to the buyer (step S2.3). The winning offers are then selected using a small nested trading negotiation (steps B3 and S3). The buyer uses the contents of the winning offers to find a set of candidate execution plans  $P_m$  and their respective estimated costs  $C_m$  (step B4), and an enhanced set  $Q$  of queries-costs pairs  $(q_e, c_e)$  (steps B5 and B6) which they could possibly be used in the next iteration of the algorithm for further improving the plans produced at step B4. Finally, in step B7, the best execution plan  $P_*$  out of the candidate plans  $P_m$  is selected. If this is not better than that of the previous iteration (i.e., no improvement) and if step B6 did not find any new query, then the algorithm is terminated.

As previously mentioned, our algorithm looks like a general bargaining with the difference that in each step the sellers and the queries bargained are different. In the steps B2, B3 and S3 of each iteration of the algorithm, a complete (nested) trading negotiation is conducted to select the best seller nodes and



**Figure 4. Modules used by the optimization algorithm.**

offers. The protocol used can be any of the ones discussed in sub-section 2.1.

### 3.3 Algorithm Details

Figure 4 shows the modules required for an implementation of our optimization algorithm (grayed boxes concern modules running at the seller nodes) and the processing workflow between them. As Figure 4 shows, the buyer node initially assumes that the value of query  $q$  is  $C_i$ , and asks its *buyer strategy* module to make a (strategic) estimation of its value using a traditional e-commerce trading reasoning. This estimation is given to the *buyer negotiation protocol* module that asks for bids (RFB) from the selling nodes. The seller, using its *seller negotiation protocol* module, receives this RFB and forwards it to the *partial query constructor and cost estimator* module, which builds pairs of a possible part of query  $q$  together with an estimate of its respective value. The pairs are forwarded to the *seller predicates analyser* to examine them and find additional queries (e.g., materialized views) that might be useful to the buyer. The output of this module (set of (sub-)queries and their costs) is given to the *seller strategy module* to decide (using again an e-commerce trading reasoning) which of these pairs is worth attempting to sell to the buyer node, and in what value. The *negotiation protocols* modules of both the seller and the buyer then run through the network a predefined trading protocol (e.g. bidding) to find the winning offers  $(q_i, c_i)$ . These offers are used by the buyer as input to the *buyer query plan generator*, which produces a number of candidate execution plans  $P_m$  and their respective buyer-estimated costs  $C_m$ . These plans are forwarded to the *buyer predicates analyser* to find a new set  $Q$  of queries  $q_e$  and then, the workflow is restarted unless the set  $Q$  was not modified by the *buyer predicates analyser* and the *buyer query plan generator* failed to find a better candidate plan than that of the previous workflow iteration.

It is worth comparing Figure 1, which shows the typical trading framework, to Figure 4, which describes our query trading framework. These figures show that the buyer strategy module of the general framework is enhanced in the query trading framework with a query plan generator and buyer predicates analyser. Similarly, the seller strategy module is enhanced with a partial query constructor and a seller predicates analyser. These additional modules are required, since in each bargaining step the buyer and seller nodes make (counter-)offers concerning a different set of queries, than that of the previous step.

To complete the analysis of the distributed optimization algorithm, we examine in detail each of the modules of Figure 4 below.

### 3.4 Partial query constructor and cost estimator

The role of the partial query constructor and cost estimator of the selling nodes is to construct a set of queries  $q_k$  that will be offered to the buyer node. It works by examining the set  $Q$  of queries asked by the buyer and identifying the parts of these queries that the seller node can contribute to. Sellers may not have all necessary base relations, or relations' partitions, to process all elements of  $Q$ . Therefore, they initially examine each query  $q$  of  $Q$  and rewrite it (if possible), using the following algorithm, which removes all non-local relations and restricts the base-relation extents to those partitions available locally:

---

<b>Query rewriting algorithm</b>	
1.	While there is a relation $R$ in query $q$ that is not available locally, do
1.1.	Remove $R$ from $q$ .
1.2.	Update the SELECT-part of $q$ adding the attributes of the rest relations of $q$ that are used in joins with $R$ .
1.3.	Update the WHERE-part of $q$ removing any operators referencing relation $R$ .
1.	End While
2.	For each remaining relation $R$ in query $q$ , do
2.1	Update the WHERE-part of $q$ adding the restriction operators of the partitions of $R$ that are stored locally.
2.	End For

---

As an example of how the previous algorithm works, consider the example of the telecommunications company and consider again the example of the query asked by that manager at Athens. Assume that the Myconos node has the whole invoiceline table but only the partition of the customer table with the restriction `office='Myconos'`. Then, after running the query rewriting algorithm at the Myconos node and simplifying the expression in the WHERE part, the resulting query will be the following:

```
SELECT SUM(charge)
FROM invoiceline i, customer c
WHERE i.custid=c.custid AND office='Myconos';
```

The restriction `office='Myconos'` was added to the above query, since the Myconos node has only this partition of the customer table.

After running the query rewrite algorithm, the sellers use their local query optimizer to find the best possible local plan for each (rewritten) query. This is needed to estimate the properties and cost of the query-offers they will make. Conventional local optimizers work progressively pruning sub-optimal access paths, first considering two-way joins, then three-way joins, and so on, until all joins have been considered [22]. Since, these partial results may be useful to the buyer, we include the optimal two-way, three-way, etc. partial results in the offer sent to the buyer. The modified dynamic programming (DP) algorithm [7] that runs for each (rewritten) query  $q$  is the following (The queries in set  $D$  are the result of the algorithm):

---

<b>Modified DP algorithm</b>	
1.	Find all possible access paths of the relations of $q$ .
2.	Compare their cost and keep the least expensive.
3.	Add the resulting plans into set $D$ .
3.	For $i=2$ to number of joins in $q$ , do
3.1.	Consider joining the relevant access paths found in previous iterations using all possible join methods.
3.2.	Compare the cost of the resulting plans and keep the least expensive.
3.3.	Add the resulting plans into set $D$ .
3.	End For

---

If we run the modified DP algorithm on the output of the previous example, we will get the following queries:

1. SELECT custid  
FROM customer  
WHERE office='Myconos';
2. SELECT custid,charge  
FROM invoiceline;
3. SELECT SUM(charge)  
FROM invoiceline i, customer c  
WHERE i.custid=c.custid AND office='Myconos';

The first two SQL queries are produced at steps 1-3 of the dynamic programming algorithm and the last query is produced in its first iteration ( $i=2$ ) at step 3.3.

### 3.5 Seller Predicates Analyser

The seller predicates analyser works complementarily to the partial query constructor to further find queries that might be of some interest to the buyer node. The latter is based on a traditional dynamic programming optimizer and therefore does not necessarily find all queries that might be of some help to the buyer. If there is a materialized view that might be used to quickly find a superset/subset of a query asked by the buyer, then it is worth offering (in small value) the contents of this materialized view to the buyer. For instance, continuing the example of the previous section, if Myconos node had the materialized view:

```
CREATE VIEW invoice AS
  SELECT custid, SUM(charge)
  FROM invoiceline
  GROUP by custid;
```

then it would worth offering it to the buyer, as the grouping asked by the manager at Athens is more coarse than that of this materialized view. There are a lot of non-distributed algorithms concerning answering queries using materialized views with or without the presence of grouping, aggregation and multi-dimensional functions, like for instance [29]. All these algorithms can be used in the seller predicates analyser to further enhance the efficiency of the QT algorithm and enable it to consider using remote materialized views. The potential of improving the distributed execution plan by using materialized views is substantial, especially in large databases, data warehouses and OLAP applications.

The seller predicates analyser has another role, useful when the seller does not hold the whole data requested. In this case, the seller, apart from offering only the data it already has, it may try to find the rest of these data using a subcontracting procedure, i.e., purchase the missing data from a third seller node. In this paper, due to lack of space, we do not consider this possibility.

### 3.6 Buyer Query Plan Generator

The query plan generator combines the queries  $q_i$  that won the bidding procedure to build possible execution plans  $P_m$  for the original query  $q$ . The problem of finding these plans is identical to the answering queries using materialized views [18] problem. In general, this problem is NP-Complete, since it involves searching through a possibly exponential number of rewritings.

The most simple algorithm that can be used is the dynamic programming (DP) algorithm that we describe below. The output of the algorithm is the set  $D$  of candidate Plans  $P_m$ . If query  $q$  is the one been optimized, then candidate plans are those representing an equivalent re-writing of query  $q$  using some of the queries  $q_i$  that won the bidding procedure (see Figure 4):

<b>DP buyer plan generator</b>	
1.	Find all candidate execution Plans and move them to set $D$ .
2.	Compare the cost of the remaining (partial) plans and keep those for which there is no other partial plan with the same or greater contribution towards query $q$ and equal or smaller value.
3.	While there are still some partial plans left, do
3.1.	Consider joining or unioning all remaining partial plans using all possible ways.
3.2.	Find all candidate execution Plans and move them to set $D$ .
3.3.	Compare the cost of the remaining (partial) plans and keep those for which there is no other partial plan with the same or greater contribution towards query $q$ and equal or smaller value. The value of two plans can be compared only if they run on the same set of nodes and their results are delivered on the same node.
3.	End While

Other more advanced algorithms that may be used in the buyer plan generator include those proposed for the Manifold System (bucket algorithm [13]), the InfoMaster System (inverse-rules algorithm [19]) and recently the MiniCon [18] algorithm. These algorithms are more scalable than the SP algorithm

and thus, they should be used if the complexity of the optimized queries, or the number of horizontal partitions per relation are large. In the experiments presented at section 4, apart from the DP algorithm, we have also considered the use of the Iterative Dynamic Programming IDP-M(2,5) algorithm proposed in [6]. This algorithm is similar to DP. Its only difference is that after evaluating all 2-way join sub-plans, it keeps the best five of them throwing away all other 2-way join sub-plans, and then it continues processing like the DP algorithm.

### 3.7 Buyer Predicates Analyser

The buyer predicates analyser enriches the set  $Q$  (see Figure 4) with additional queries, which are computed by examining each candidate execution plan  $P_m$  (see previous subsection). If the queries used in these plans provide redundant information, it updates the set  $Q$  adding the restrictions of these queries which eliminate the redundancy. Other queries that may be added to the set  $Q$  are simple modifications of the existing ones with the addition/removal of sorting predicates, or the removal of some attributes that are not used in the final plan.

To make more concrete to the reader the functionality of the buyer predicate analyser, consider again the telecommunications company example, and assume that someone asks the following query:

```
SELECT custid
FROM customer
WHERE office in ('Corfu', 'Myconos', 'Santorini');
```

Assume that one of the candidate plans produced from the buyer plan generator contains the union (distinct) of the following queries:

```
1a. SELECT custid
    FROM customer
    WHERE office in ('Corfu', 'Myconos');
2a. SELECT custid
    FROM customer c
    WHERE office in ('Santorini', 'Myconos');
```

The buyer predicates analyser will see that this union has redundancy and will produce the following two queries:

```
1b. SELECT custid
    FROM customer
    WHERE office='Corfu';
2b. SELECT custid
    FROM customer
    WHERE office='Santorini';
```

In the next iteration of the algorithm, the buyer will also ask for bids concerning the above two SQL statements, which will be used in the next invocation of the buyer plan generator, to build the same union-based plan with either query (1a) or (2a) replaced with the cheaper queries (1b) or (2b) respectively.

### 3.8 Negotiation Protocols

All of the known negotiation protocols, such as bidding, bargaining and auctions can be used in the query trading environment. Bidding should be selected if someone wishes to keep the implementation as simple as possible. However, if the expected number of selling nodes is large, bidding will lead to flooding the buying nodes with too many bids. In this case, a better approach is to use an auction mechanism, since it reduces the number of bids. This is because nodes that cannot offer a better bid than the currently winning one, cannot make any new bids. On the other hand, an auction may produce sub-optimal results if the bidders are few or if some malicious nodes form a *ring* agreeing not to compete against each other. The latter is possible, if our query trading framework is used for commercial purposes. This problem affects all general e-commerce frameworks and is solved in many different ways (e.g., contract specifications, use of second-price auctions [27], etc.).

Parameter type	Parameter	Value
Network	Total size of Network (WAN)	5,000 nodes
	Minimum duration of each network message	1 ms
	WAN network packet latency	10 - 240ms (120 ms average)
	WAN interconnection speed	0.5 - 4Mbits/s
RDBMs	Join capabilities	Nested-loops, merge-scan and Hash-join
	Operators pipelining support	Yes
	CPU resources	One 700-1500 MHz CPU
	Sorting/Hashing buffer size	10,000 tuples
Dataset	I/O Speed per node	5-20Mbytes
	Size of relations	1-400,000 tuples
	Number of attributes per relation	20 attributes
	Number of partitions per relation	1-8
Workload	Number of mirrors per partition	1-3
	Indices per partition per node	3 single-attribute indices
	Joins per query	0 - 7
	Partitions per relation	1 - 8

**Table 1. Simulation parameters.**

Bargaining, instead of bidding, is worth using only when the number of expected offers is small and minor modifications in the offers are required (e.g., a change of a single query-answer property), since this will avoid the cost of another workflow iteration. If major modifications in the structure of the offered queries are required, the workflow (by definition) will have to run at least one more iteration and since each iteration is actually a generalized bargaining step, using a nested bargaining within a bargaining will only increase the number of exchanged messages.

## 4 Experimental study

In order to assert the quality of our algorithm (QT), we simulated a large network of interconnected RDBMs and run a number of experiments to measure the performance of our algorithm. To the best of our knowledge, there is no other work that addresses the problem of distributed query optimization in purely autonomous nodes. Hence, there is no approach to compare our algorithm against on an equal basis with respect to its operating environment.

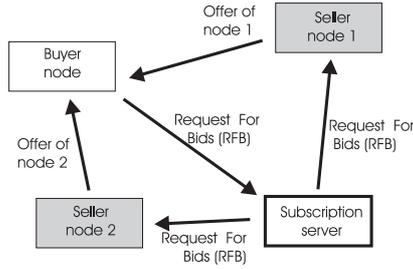
As a matter of comparison, however, and in order to identify the potential "cost" for handling true autonomy, we have also implemented the SystemR algorithm, an instance of the optimization algorithm used by the Mariposa distributed DBMS [25] and a variant of the Iterative Dynamic Programming (IDP) algorithm [6]. These are considered three of the most effective algorithms for distributed query optimization and serve as a solid basis of our evaluation.

### 4.1 Experiments setup

#### Simulation parameters

We used C++ to build a simulator of a large Wide Area Network (WAN). The parameters of this environment, together with their possible values, are displayed in Table 1. The network had 5,000 nodes that exchanged messages with a simulated latency of 10-240ms and a speed of 0.5-4Mbits. Each node was equipped with a single CPU at 1200Mhz (on average) that hosted an RDBMs capable of evaluating joins using the nested-loops and the merge-scan algorithms. 80% of the simulated RDBMs could also use the hash-join method. The local I/O speed of each node was not constant and varied between 5 and 20 Mbytes/s.

The data-set used in the experiment was synthetically constructed. We constructed the metadata of a large schema consisting of 10,000 relations. There was no need to actually build the data since our experiments measured the performance of query optimization, not that of query execution. Each relation had 200,000 tuples on average and was horizontally range-partitioned in 1-8 disjoint partitions that were



**Figure 5. The experiment network architecture.**

stored in possibly different nodes in such a way that each partition had 0-3 mirrors. The nodes were allowed to create 3 local indices per locally stored partition.

In order for the nodes to be aware of all currently active RFBs, we used a publish-subscribe mechanism, since these are widely used in existing agent-based e-commerce platforms [15] and have good scalability characteristics. Our framework worked as follows (see Figure 5). All nodes registered their local relations' partitions to a special subscription server. When a node asked for bids, it contacted this server which in turn informed seller nodes who had data related to the requested query to make bids. Thus, the overall cost of each bidding in terms of network resources was two messages per seller plus one message from the buyer to the subscription server. Each bidding procedure was set to last for a maximum of 400ms.

It should be noted that the subscription server in our experiments, hosted a directory service used by buyers to efficiently locate candidate sellers. This type of architecture is typical for small e-commerce negotiation frameworks [3]. There are several different architectures that could also be used for the construction of the bidding framework. For instance, on very large networks, such as the Internet, where nodes join and leave the network in an arbitrary way, the directory service could be based on the P2P paradigm (e.g., the P2P auction mechanism [15]). The choice of using a subscription server for locating candidate sellers does not reduce the generality of the presented results, as the affected parameter is the duration of each bidding round.

### Query optimization algorithms

In each experiment, we studied the execution plans produced by the following four algorithms: SystemR, IDP(2,5), Mariposa, and Query Trading. More specifically:

**SystemR** The SystemR algorithm [22] was examined as it produces optimal execution plans. However, it is not a viable solution in the large autonomous and distributed environments that we consider for two reasons. The first one is that it cannot cope with the complexity of the optimization search space. For instance, a relation with 8 partitions and 3 mirrors per partition may be spread in 24 different nodes, which in turn means that a join of two such relations can be split horizontally and be parallelized in thousands of different ways. The second one is that it requires cost estimations from remote nodes. Not only this makes the nodes not autonomous, but for a query with  $n$  joins, it will take  $n$  rounds of message exchanges to find the required information. In each round, the remote nodes will have to find the cost of every feasible  $k$ -way join ( $k=1..N$ ), which quickly leads to a network bottleneck for even very small numbers of  $n$ .

We implemented SystemR so that we have a solid base for comparing the quality of the plans produced by the rest algorithms. Our implementation assumed that nodes running the algorithm had exact knowledge of the state of the whole network. This made possible running the algorithm without the bottleneck of network throughput, which would normally dominate its execution time.

**IDP-M(2,5)** Recently, a heuristic extension of the SystemR algorithm, the IDP-M( $k,m$ ), was proposed for use in distributed environments [6]. Therefore, we choose to include an instance of this algorithm in our study. Given an  $n$ -way join query, it works like this [6]: First, it enumerates all

feasible  $k$ -way joins, i.e., all feasible joins that contain less than or equal to  $k$  base tables and finds their costs, just like SystemR does. Then, it chooses the best  $m$  subplans out of all the subplans for these  $k$ -way joins and purges all others. Finally, it continues the optimization procedure by examining the rest  $n - k$  joins in a similar to SystemR way. The IDP algorithm is not suitable for autonomous environment as it shares the problems of SystemR mentioned above.

For an  $n$ -way star join query ( $k \ll n$ ), where each relation has  $p$  horizontal partitions ( $p \geq 2$ ) each with  $M$  mirrors ( $M \geq 2$ ), at least  $O(M2^p mn^k)$  plans [6] would have to be transmitted through the network. Our implementation assume that nodes running the algorithm have exact knowledge of the state of the whole network and thus avoids the bottleneck of network throughput, which similar to SystemR, would normally dominate its execution time [6].

**Mariposa** The Mariposa query optimization algorithm [16, 24] is a two-step algorithm that considers conventional optimization factors (such as join orders) separately from distributed system factors (such as data layout and execution location). First, it uses information that it keeps locally about various aspects of the data to construct a locally optimal plan by running a local optimizer over the query, disregarding the physical distribution of the base relations and fixing such items as join order and the application of join and restriction operators. It then uses network yellow-pages information to parallelize the query operators, and a bidding protocol to select the execution sites, all in a single interaction with the remote nodes. The degree of parallelism is statistically determined by the system administrator before query execution and is independent of the available distributed resources.

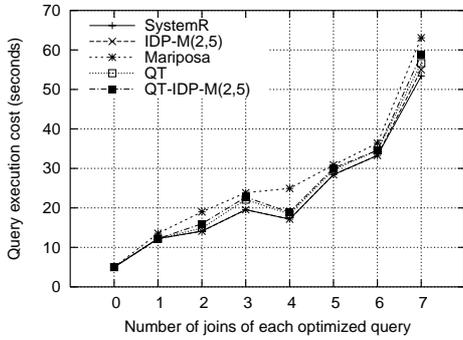
The Mariposa System was initially designed to cover a large range of different requirements. In this paper, we implemented the Mariposa algorithm as described in [1] with the difference that in the second phase of the algorithm, the optimization goal was set to minimize the total execution time of the query, which is the task of interest in our experiments. We included Mariposa in our study as it is one of the fastest-running known algorithm for distributed query optimization. Nevertheless, it is not suitable for true autonomous environments as it requires information from nodes that harm their autonomy, such as their cost functions, their join capabilities, and information on the data and indices they have.

**Query trading** We instantiated the Query Trading Algorithm using the following properties:

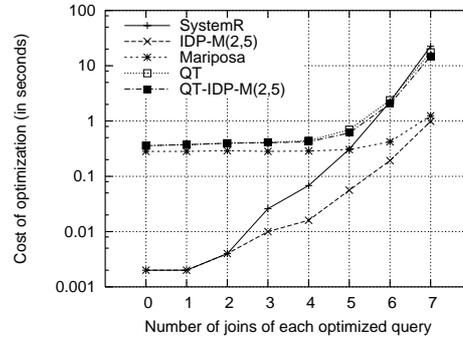
- For the Negotiation protocol, we choose the bidding protocol (see section 2.1) as the expected number of offered bids for each RFB was not large enough for an auction protocol to be beneficiary. We ranked each offer based only on the time required to return the complete query answer. In this way, our optimization algorithm produced plans that had the minimum execution time, reflecting the traditional optimization task.
- We used a plain cooperative strategy. Thus nodes replied to RFBs with offers matching exactly their available resources. The seller strategy module had a small buffer that held the last 1,000 accepted bids. These bids were collected from past biddings that the node had participated in. Using the contents of this buffer, nodes estimated the most probable value of the offer that will win a bidding and never made offers with value more than 20% of this estimation. This strategy helped to reduce the number of exchanged messages.
- In the Buyer query plan generator we used a traditional answering queries using views dynamic programming algorithm. However, to decrease its complexity we also tested the plan generator with the IDP-M(2,5) algorithm.

## Simulated scenarios

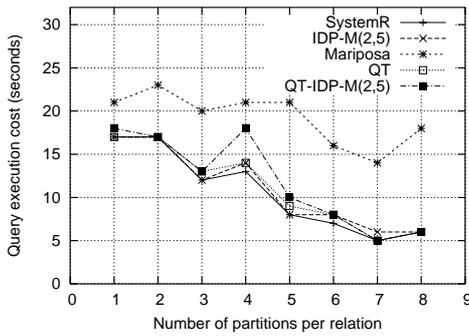
We initially run some experiments to assert the scalability of our algorithm in terms of network size. As was expected, the performance of our algorithm was not dependent on the total number of network nodes but on (a) the number of nodes which replied to RFBs and (b) the complexity of the queries. Thus we ended up running three sets of experiments: In the first set, the workload consisted of star-join



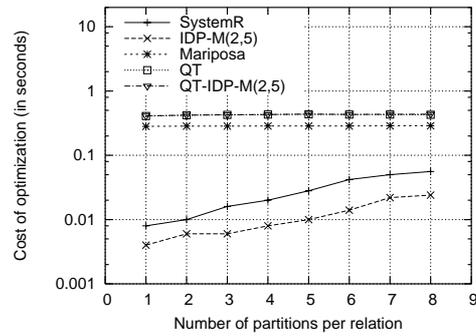
(a) Average execution cost of plans vs number of joins.



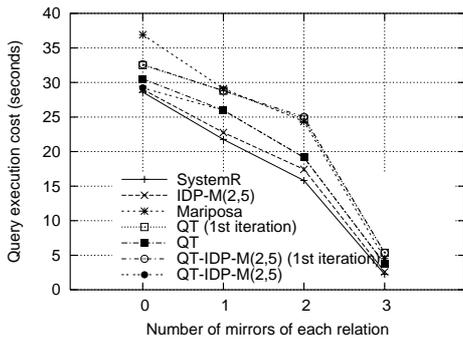
(b) Cost of optimization in seconds vs number of joins.



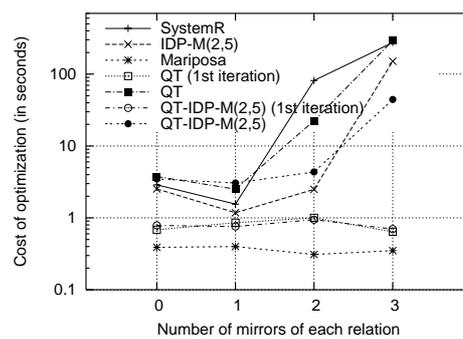
(c) Average execution cost of plans vs number of horizontal partitions.



(d) Cost of optimization in seconds vs number of horizontal partitions.



(e) Average execution cost of plans vs number of mirrors per partition.



(f) Cost of optimization in seconds vs number of mirrors per partition.

**Figure 6. Performance of various distributed query optimization algorithms.**

queries with a varying number of 0-7 joins. The relations referenced in the joins had no mirrors and only one partition. This workload was used to test the behavior of the optimization algorithms as the complexity of queries increased. In the second set of experiments, we considered 3-way-join star-queries that referenced relations without mirrors but with a varying number of 1-8 horizontal partitions. This test measured the behavior of the algorithms as relations' data were split and stored (data spreading) in different nodes. Finally, in the last set of experiments we considered 3-way-join star-queries that referenced relations with three partitions and a varying number of mirrors (1-3). This experiment measured the behavior of the algorithms in the presence of redundancy.

We run each experiment five times and measured the average execution time of the algorithms and the average plan cost of the queries, i.e., the time in seconds required to execute the produced queries as this was estimated by the algorithms. We should note that the execution times of SystemR and IDP are not directly comparable to those of other algorithms, as they do not include the cost of network message exchanges. The numbers presented for them are essentially for a non-autonomous environment with a centralized node for query optimization.

## 4.2 The results

### Number of joins

Figures 6(a) and 6(b) present the results of the first set of tests. The first figure presents the average execution cost of the plans produced by the SystemR, IDP(2,5), Mariposa and the two instances of the QT algorithm. It is surprising to see that even for such simple distributed data-sets (no mirrors, no partitions) all algorithms (except SystemR) fail to find the optimal plan. As expected, IDP(2,5) algorithm deviates from the optimal plan when the number of joins is more than two. The Mariposa algorithm makes the largest error producing plans that require on average 20% more time than those produced by SystemR. This is because in its first phase of the optimization procedure, Mariposa is not aware of network costs and delays and thus, fails to find the best joins order for the base tables. The rest of the algorithms usually selected the proper joins order, taking into account that the delay caused by slow data sources may be masked if the joins related with these sources are executed last in the subplans pipeline. The QT-IDP-M algorithm produces plans that are only marginally inferior to those of plain QT.

The execution time (Figure 6(b)) of all algorithms depends exponentially on the number of joins. The QT, QT-IDP and Mariposa algorithms have a start-up cost of a single round of message exchanges (approx. 400ms and 280ms respectively) caused by the bidding procedure. The QT and QT-IDP algorithms never had to run more than one round of bids.

It should be noted that the queries in this set of experiments represent the worst case scenario for the complexity of QT and QT-IDP. Each table was neither partitioned nor mirrored. The bidding procedure of these algorithms could not prune any of the received bids, as there were no bids for the same subquery. Thus, the optimization costs of QT and QT-IDP, apart from the initial penalty of 400ms approx. for their bidding procedure, were similar to those of the centralized SystemR and IDP-M, respectively. Nevertheless, recall that the SystemR and IDP-M optimization costs of figure 6(b) do not include the cost of network message exchanges.

### Data partitioning

Figures 6(c) and 6(d) summarize the results of the second set of experiments, which evaluate the algorithm as the partitioning and spreading of information varied. The first figure shows that the QT algorithm is little affected by the level of data spreading and partitioning. This was expected since the bidding procedure completely masks data spreading. The performance of the Mariposa algorithm is substantially affected by the level of data partitioning, producing plans that are up to three times slower than those of SystemR. This is because as the number of partitions increase, the chances that some remote partitions may have some useful remote indices (which are disregarded by Mariposa as each node

	Few joins	Many joins
No mirroring and no partitions	<b>QT</b>	<b>QT</b>
No mirroring and many partitions	<b>QT</b>	<b>QT</b>
Mirroring	<b>QT-IDP-M</b> if cost of query execution large else <b>Mariposa</b>	<b>QT-IDP-M</b> if cost of query execution large else <b>Mariposa</b>

**Table 2. Simulation summary results.**

has different indices ) are increased. Moreover, Mariposa does not properly splits the join operators among the different horizontal partitions and nodes, as operators splitting is statically determined by the database administrator and not by the Mariposa algorithm itself.

The execution times of all algorithms depend on the number of relations partitions, as they all try to find the optimal join and unions orders to minimize the execution cost. Figure 6(d) shows the execution time of the QT, QT-IDP-M and Mariposa algorithm constant due to the logarithmic scale and the fact that the cost of the single bid round was much larger than that of the rest costs. Similarly to the previous set of experiments, the QT and QT-IDP-M algorithms run on average a single round of bids.

### Data mirroring

Figures 6(e) and 6(f) summarize the results of the last set of experiments, where the level of data mirroring is varied. Data mirroring substantially increases the complexity of all algorithms as they have to horizontally split table-scan, join and union operators and select the nodes which allow for the best parallelization of these operators. The QT algorithm is additionally affected by data mirroring, as it has to run multiple rounds of bid procedures. On average, three rounds per query were run. For comparison reasons we captured the execution time and plan produced both at the end of the first iteration and at the end of the final one.

Figures 6(e) and 6(f) shows that although the first iteration of the QT algorithm completed in less than a second, the plans produced were on average the worst ones. Mariposa completed the optimization in the shortest possible time (less that 0.5s) and produced plans that were usually better than those produced in the first iteration of QT and QT-IDP, but worst than those produced in their last iteration . Finally, the best plans were produced by SystemR and IDP-M(2,5).

As far as the optimization cost is concerned, the SystemR, QT, and IDP-M(2,5) algorithms were the slowest ones. The QT-IDP-M algorithm was substantially faster than QT and yet, produced plans that were close to those produced by the later. Finally, Mariposa was the fastest of all algorithms but produced plans that were on average up to 60% slower than those of SystemR.

### 4.3 Summary

Table 2 summarizes our experimental findings when the autonomy or size of network nodes forbids the use of the SystemR or IDP-M algorithm. If the data are not mirrored, then the QT algorithms produces close to optimal execution plans within a few hundreds of ms. If data are mirrored then the QT-IDP-M algorithm can be used with a relatively moderate "cost of costing". A multi-step optimization algorithm, like Mariposa, should be considered only when the expected total execution cost of the query is small compared to the execution time of the QT-IDP-M algorithm.

## 5 Discussion

The SystemR and IDP-M centralized algorithms attempt to find the best execution plan for a query by examining several different solutions. However, the search space is too large and the volume of information exchanged with the remote nodes creates a network flooding problem. Thus, SystemR and IDP-M are inappropriate for optimizing queries in large networks of autonomous nodes.

In such networks, the optimization procedure must be distributed, i.e., all (remote) nodes must contribute to the optimization process. This ensures that the search space is efficiently divided to many

nodes and at the same time, it minimizes the number of messages exchanged during query optimization, as remote nodes need not inform a node running some centralized algorithm of their status. The QT algorithm cleverly distributes the optimization process in many remote nodes. This is done by asking candidate sellers to calculate and find the cost of any possible sub-queries that might be useful for the construction of the global plan. Network flooding is avoided using standard e-commerce techniques. For instance, in our experiments, the strategy module disallowed bids that had few chances of being won.

Obviously, a good distributed query optimization algorithm must construct the query plan and select the nodes that will execute the parts of this plan simultaneously. Distributed query optimization requires good knowledge of the current state of the nodes of the network. This includes the location of table partitions, the available indices and materialized views, the network interconnections throughput, the sizing and capabilities of remote nodes, their cost functions and their current load. In the environments that we consider, nodes have to ask other ones for this information, as no node has exact knowledge of all these parameters. Consequently, any good algorithm in an autonomous environment must initially contact remote nodes to retrieve this necessary information. This is what the QT algorithm does.

The Mariposa, two-step algorithm is completely different to QT. It first builds the execution plan, disregarding the physical distribution of base relations and then selects the nodes where the plan will be executed using a greedy approach. Working this way, Mariposa and more generally any other two-step algorithm that treats network interconnection delays and data location as second-class citizens produce plans that exhibit unnecessarily high communication costs [11] and are arbitrarily far from the desired optimum [16]. Furthermore, they violate the autonomy of remote nodes as they require all nodes to follow a common cost model and ask remote nodes to expose information on their internal state. Finally they cannot take advantage of any materialized views or advanced access methods that may exist in remote nodes.

The QT algorithm is the only one of the four algorithms examined that truly respects the autonomy of remote nodes. It treats them as true black boxes and runs without any information on them, apart from that implied by their bids. It does not need or make any assumptions on the state of remote nodes (e.g., their CPU resources, etc.), not even on the data model that they actually internally use. QT is the only of the algorithms examined that protects the privacy of remote nodes, which is very important for keeping them truly autonomous.

## 6 Conclusions and future work

We discussed the parameters affecting a framework for trading queries and their answers, showed how we can use such a framework to build a WAN distributed query optimizer, and thoroughly examined its performance characteristics.

Our algorithm can be used in any large federation of cooperating or competitive DBMS, hosted in a large intranet or the Internet. As far as the authors are aware of, this is the only algorithm that runs in an environment consisting of true autonomous nodes. The experimental results show that the algorithm is scalable and produces plans that are comparable in terms of quality to those produced by other existing distributed optimization algorithms.

The possibilities for further improving the QT algorithm are many, including the use of subcontracting for better parallelization of the execution plans, the use of *contracting* to model the notion of partial/adaptive optimization used by many other algorithms, the use of other cost functions (e.g. network bytes, for minimizing network resources) and the use of advanced competitive strategies.

## References

- [1] Mariposa distributed database management systems, User's Manual. Available at <http://s2k-ftp.cs.berkeley.edu:8000/mariposa/src/alpha-1/mariposa-manual.pdf>, 2002.
- [2] M. Bichler, M. Kaukal, and A. Segev. Multi-attribute auctions for electronic procurement. In *Proc. of the 1st IBM IAC Workshop on Internet Based Negotiation Technologies, Yorktown Heights, NY, March 18-19, 1999*.

- [3] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. In *Proc. of Commercial Applications for High-Performance Computing Conference, SPIE International Symposium on The Convergence of Information Technologies and Communications (ITCom 2001), August 20-24, 2001, Denver, Colorado*, 2001.
- [4] J. Collins, M. Tsvetovat, R. Sundareswara, J. van Tonder, M. L. Gini, and B. Mobasher. Evaluating risk: Flexibility and feasibility in multi-agent contracting. In *Proc. of the 3rd Annual Conf. on Autonomous Agents*, Seattle, WA, USA., May 1999.
- [5] V. Conitzer and T. Sandholm. Complexity results about nash equilibria. *Technical report CMU-CS-02-135*, [http://www-2.cs.cmu.edu/~sandholm/Nash\\_complexity.pdf](http://www-2.cs.cmu.edu/~sandholm/Nash_complexity.pdf), 2002.
- [6] A. Deshpande and J. M. Hellerstein. Decoupled query optimization for federated database systems. In *Proc. of 18th. ICDE, San Jose, CA*, pages 716–727, 2002.
- [7] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.
- [8] Y. E. Ioannidis and Y. C. Kang. Randomized algorithms for optimizing large join queries. In H. Garcia-Molina and H. V. Jagadish, editors, *Proc. of the 1990 ACM SIGMOD Conf., Atlantic City, NJ*, pages 312–321. ACM Press, 1990.
- [9] Y. E. Ioannidis, R. T. Ng, K. Shim, and T. K. Sellis. Parametric query optimization. *VLDB Journal*, 6(2):132–151, 1997.
- [10] J. H. Kagel. *Auctions: A Survey of Experimental Research*. The Handbook of Experimental Economics, edited by John E. Kagel and Alvin E. Roth, Princeton: Princeton University Press, 1995.
- [11] D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, September 2000.
- [12] S. Kraus. *Strategic Negotiation in Multiagent Environments (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2001.
- [13] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of 22th Int. Conf. on VLDB*, pages 251–262, 1996.
- [14] J. C. Navas and M. Wynblatt. The Network is the Database: Data Management for Highly Distributed Systems. In *Proceedings of ACM SIGMOD’01 Conference*, 2001.
- [15] E. Ogston and S. Vassiliadis. A Peer-to-Peer Agent Auction. In *Proc. of AAMAS02, Bologna, Italy*, July 15–19 2002.
- [16] C. H. Papadimitriou and M. Yannakakis. Multiobjective query optimization. In *Proc. of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on PODS, May 21-23, 2001, Santa Barbara, CA, USA*. ACM, ACM, 2001.
- [17] H. V. D. Parunak. *Manufacturing experience with the contract net*. Distributed Artificial Intelligence, Michael N. Huhns (editor), Research Notes in Artificial Intelligence, chapter 10, pages 285-310. Pitman, 1987.
- [18] R. Pottinger and A. Levy. A scalable algorithm for answering queries using views. In *Proc. of the 26th VLDB Conference, Cairo, Egypt*, 2000.
- [19] X. Qian. Query folding. In *Proc. of ICDE, New Orleans, LA*, pages 48–55, 1996.
- [20] J. S. Rosenchein and G. Zlotkin. *Rules of Encounter : designing conventions for automated negotiation among computers*. The MIT Press series in artificial intelligence, 1994.
- [21] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, 2002.
- [22] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proc. of 1979 ACM SIGMOD*, pages 22–34. ACM, 1979.
- [23] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12):1104–1113, December 1980.
- [24] M. Stonebraker, P. M. Aoki, R. Devine, W. Litwin, and M. A. Olson. Mariposa: A new architecture for distributed data. In *ICDE*, pages 54–65, 1994.
- [25] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeller, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A wide-area distributed database system. *VLDB Journal*, 5(1):48–63, 1996.
- [26] S. Y. Su, C. Huang, J. Hammer, Y. Huang, H. Li, L. Wang, Y. Liu, C. Pluempitwiriyawej, M. Lee, and H. Lam. An internet-based negotiation server for e-commerce. *VLDB Journal*, 10:72–90, 2001.
- [27] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, 1961.
- [28] P. Winoto, G. McCalla, and J. Vassileva. An extended alternating-offers bargaining protocol for automated negotiation in multi-agent systems. In *Proc. of 10th Int. Conf. on CoopIS’ 2002*, Irvine, U.S.A., November 2002.
- [29] M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh, and M. Urata. Answering complex sql queries using automatic summary tables. In *Proceedings of ACM SIGMOD’00 Conference*, pages 105-116, 2000.