Decoupled Query Optimization for Federated Database Systems

Amol Deshpande Joseph M. Hellerstein
Computer Science Division
University of California, Berkeley
{amol, jmh}@cs.berkeley.edu

Abstract

We study the problem of query optimization in federated relational database systems. The nature of federated databases explicitly decouples many aspects of the optimization process, often making it imperative for the optimizer to consult underlying data sources while doing cost-based optimization. This not only increases the cost of optimization, but also changes the trade-offs involved in the optimization process significantly. The dominant cost in the decoupled optimization process is the "cost of costing" that traditionally has been considered insignificant. The optimizer can only afford a few rounds of messages to the underlying data sources and hence the optimization techniques in this environment must be geared toward gathering all the required cost information with minimal communication.

In this paper, we explore the design space for a query optimizer in this environment and demonstrate the need for decoupling various aspects of the optimization process. We present minimum-communication decoupled variants of various query optimization techniques, and discuss trade-offs in their performance in this scenario. We have implemented these techniques in the Cohera federated database system and our experimental results, somewhat surprisingly, indicate that a simple two-phase optimization scheme performs fairly well as long as the physical database design is known to the optimizer, though more aggressive algorithms are required otherwise.

1. Introduction

The need for federated database services has increased dramatically in recent years. Within enterprises, IT infrastructures are often decentralized as a result of mergers, acquisitions, and specialized corporate applications, resulting in deployment of large federated databases. Perhaps more dramatically, the Internet has enabled new inter-enterprise ventures including *Business-to-Business Net Markets* (or *Hubs*) [1, 32], whose business hinges on federating thousands of decentralized catalogs and other databases.

Broadly considered, federated database technology [44] has been the subject of multiple research thrusts, including schema integration [6, 35], data transformation [2], as well as federated query processing and optimization. The query optimization work goes back as far as the early distributed database systems (R*, SDD-1, Distributed Ingres [22, 14, 7]), and most recently has been focused on linking data sources of various capabilities and cost models [23, 30, 46]. However, query optimization in the broad federated environment presents peculiarities that change the trade-offs in the optimization process quite significantly. By nature, federated systems *decouple* many aspects of the query optimization process that were tightly integrated in both centralized and distributed database systems. These decouplings are often forced by administrative constraints, since federations typically span organizational boundaries; decoupling is also motivated by the need to scale the administration and performance of a system across thousands of sites[1]. Federated query processors need to consider three basic decouplings:

- **Decoupling of Query Processing**: In a large-scale federated system, both data access and computation can be carried out at various sites. For global efficiency, it is beneficial to consider assigning portions of a query plan in arbitrary distributed ways. In fact, this has been one of the major motivations for development of both distributed and federated database systems.
- **Decoupling of Cost Factors**: In a centralized DBMS, query execution "cost" is a unidimensional construct measured in abstract units. In a federation, costs must be decoupled into multiple dimensions under the control of various administrators. One proposal for a universal cost metric is hard currency [45], but typically there are other costs that are valuable to expose orthogonally, including response time [17], data freshness [36], and accuracy of computations [5].
- **Decoupling of Cost Estimation**: This work is motivated by the necessity of decoupling the cost estimation aspect of the query optimizer from the optimiza-

[1] We will use the terms *site* and *data source* interchangeably in this paper.

Proceedings of the 18th International Conference on Data Engineering (ICDE'02)
1063-6382/02 $17.00 © 2002 IEEE

tion process. Regardless of the number of cost dimensions, a centralized optimizer cannot accurately estimate the costs of operations at many autonomous sites. Garlic [23, 40] and other middleware systems [24, 46] address this problem by involving site-specific wrappers in the optimization process, but they do not consider the cost of communicating with these wrappers. This cost is not significant in these systems because the wrappers typically reside in the same address space as the optimizer. But in general, the execution costs may also depend on transient system issues including current loads and temporal administrative policies [45], and hence the cost estimation process must be federated in a manner reflective of the query processing, with cost estimates being provided by the sites that would be doing the work.

Many of these decouplings have been studied before individually in the context of distributed, heterogeneous or federated database research [41, 15, 38]. However, to the best of our knowledge, complete decoupling of cost estimation, which requires the optimizer to communicate with the sites merely to find the cost of an operation, has not been studied before. In such a scenario, communication may become the dominant cost in the query optimization process. The high *cost of costing* raises a number of new design challenges, and adds additional factors to the complexity of federated query optimization.

1.1. Contributions of the Paper

In this paper, we consider a large space of federated query optimizer design alternatives and argue the need for taking into consideration the high "cost of costing" in this environment. Accordingly, we present minimum-communication decoupled variants of various well-known optimization techniques. We have implemented these algorithms in the Cohera federated database system [25] and we present experimental results on a set of modified TPC-H benchmark queries.

Our experimental results, somewhat surprisingly, suggest that the simple technique of breaking the optimization process into two phases [26] — first finding the best query plan for a single machine and then scheduling it across the federation based on run time conditions — works very well in the presence of fluctuations in the loads on the underlying data sources and the communication costs, as long as the physical database design is known to the optimizer. On the other hand, if the optimizer is unaware of the physical database design (such as indexes or materialized views present at the underlying data sources), then more aggressive optimization techniques are required and we propose using a hybrid technique for tuning a previously proposed heuristic in those circumstances.

We also present a preliminary analysis explaining this surprising success of the two-phase optimizer for our cost model and experimental settings later in the paper (Section 4.3). Our analysis suggests that this behavior may not merely be a peculiarity of our experimental settings, but may hold true in general.

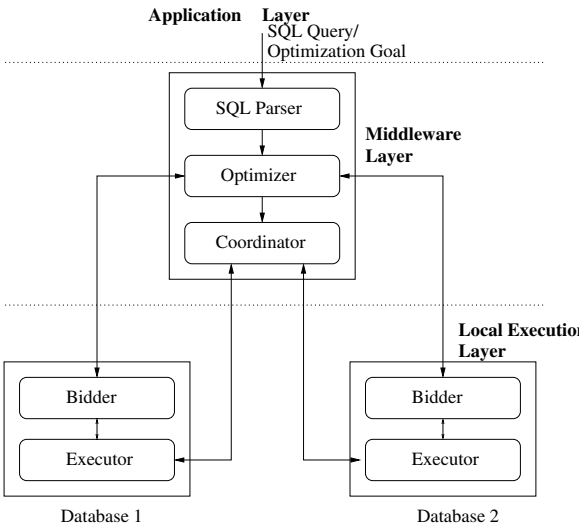

Figure 1. System Architecture

2. Architecture and Problem Definition

We base our system architecture on the Mariposa research system [45], which provides the decouplings discussed in the earlier section through the use of an *economic paradigm*. The main idea behind the economic paradigm is to integrate the underlying data sources into a *computational economy* that captures the autonomous nature of various sites in the federation. A significant and controversial goal of Mariposa was to demonstrate the global efficiency of this economic paradigm, e.g., in terms of distributed load balancing. For our purposes here, controversies over economic policy are not relevant; the long-term adaptivity problem that Mariposa tried to solve is beyond the scope of this paper. The main benefit of the economic model for us is that it provides a fully decoupled costing API among sources. As a result, each site has *local autonomy* to determine the cost to be reported for an operation, and can take into account factors such as resource consumption, response time, accuracy and staleness of data, administrative issues, and even supply and demand for specialized data processing.

For query optimization purposes, the most relevant parts of the system are the *query optimizer* in the middleware, and the *bidders* at the underlying sites (Figure 1). As in a centralized database system, the query optimizer could use a variety of different optimization algorithms, but the federated nature of the system requires that the cost estimates be made by the underlying data sources or in our case, by the bidders. The optimizer and the bidder communicate through use of two constructs : (1) *Request for Bid (RFB)* that the optimizer uses to request cost of an operation, and

(2) *Bid* through which a bidder makes cost estimates.

2.1. The Federated Query Optimization Problem

The federated query optimization problem is to find an execution plan for a user-specified query that satisfies an optimization goal provided by the user; this goal may be a function of many variables, including response time, total execution cost, accuracy and staleness of the data. For simplicity, we concentrate on two of these factors, response time and total execution cost (measured in abstract cost units), though it is fairly easy to extend these to include other factors, assuming they can be easily estimated. Since we assume that the only information we have about the costs of operations is through the interface to the bidders, the optimization problem has to be restated as optimizing over the cost information exported by the bidders. Before describing the adaptations of the known query optimization algorithms to take into account the high cost of costing, we will discuss two important issues that affect the optimization cost in this framework significantly.

2.1.1. Response Time Optimization vs. Total Cost Optimization : Traditionally the optimization goal has been minimization of the total cost of execution, but in many applications, other factors such as response time, staleness of the data used in answering the query [36], or accuracy of the data [5] may also be critical. As has been pointed out previously [17, 48], optimizing for such an optimization goal requires the use of *partial order dynamic programming* technique. This technique is a generalization of the classical dynamic programming algorithm where the cost of each plan is computed as a vector and two costs are considered incomparable if neither is less than or equal to the other in all the dimensions[2]. It can be shown that if the cost is an l-dimensional vector, then the time and space complexity of the optimization process increases by a factor of 2^l [17] over classical dynamic programming. Recently, a polynomial time approximation algorithm has also been proposed for this problem [39]. As [33] point out, even total cost optimization in a distributed setting requires partial order dynamic programming since two plans producing the same result on different sites are not comparable due to the subsequent communication costs which might differ.

2.1.2. Bidding Granularity and Intra-site Pipelining : The bidding granularity refers to the choice of the operations for which the optimizer requests costs. For maximum flexibility in scheduling the query plan, we would like this to be as fine-grained as possible. The natural choices for bidding granules to estimate the cost of a query plan are *scans* on the underlying base tables and *joins* in the query plan. This creates a problem if we want to use intra-site pipelining since the optimizer does not know whether a particular site will pipeline two consecutive joins. In the absence of any information from the sites, the optimizer could either assume that every pair of joins that appear one after another in the query plan *will be* pipelined at a site, or it could assume that there is no intra-site pipelining. Either assumption could result in incorrect estimation of the query execution cost. This problem can be solved by allowing *multi-join bid requests*, where the optimizer sends bid requests consisting of multiple relations and the bidder is asked to make a bid on the join involving all of these relations. The bidder can then use pipelining if there are enough resources.

[2]Partial order dynamic programming can also be thought of as a generalization of the *interesting orders* of System R [42], where two subplans are considered incomparable if they produce the same result in different sorted order, and the decision about the optimal subplan is only made at the end of the optimization process.

2.2. Simplifying Assumptions

To simplify the discussion in the rest of the paper, we will make the following assumptions :

- **Accurate Statistics :** We assume that statistics regarding the cardinalities and the selectivities are available. This information can be collected through standard protocols such as ODBC/JDBC that allow querying the host database about statistics, or by caching statistics from earlier query executions [3].
- **Communication Costs :** We assume that communication costs remain roughly constant for the duration of optimization and execution of the query, and that the optimizer can estimate the communication costs incurred in data transfer between any two sites involved in the query.
- **No Pipelining Across Sites :** We assume that there is no pipelining of data among query operators across sites. The main issue with pipelining across sites is that the pipelined operators tend to waste resources, especially *space shared* resources such as memory [19]. Even if the producer is not slow, the communication link between the two sites could be slow, especially for WANs, and the consumer will be holding resources while waiting for the network.

3. Adapting the Optimization Techniques

In this section, we discuss our adaptations of various well-known optimization techniques to take into account the high "cost of costing". Aside from minimizing the total communication cost, we also want to make sure that the plan space explored by the optimization algorithm remains the same as in the centralized version of the algorithm.

In general, we will break all optimization algorithms into three steps :

- **Step 1 :** Choose subplans that require cost estimates and prepare the requests for bids.
- **Step 2 :** Send messages to the bidders requesting costs.

- **Step 3 :** Calculate the costs for plans/subplans. If possible, decide on an execution plan for the query, otherwise, repeat steps 2 and 3.

Clearly we should try to minimize the number of repetitions of steps 2 and 3, since step 2 involves expensive communication.

3.1. Classical Dynamic Programming (Exhaustive)

This exhaustive algorithm searches through all possible plans for the query, using dynamic programming[3] and the principle of optimality to prune away bad subplans as early as possible [42]. Though the algorithm is exponential in nature, it finishes in reasonable time for joins involving a small number of relations, and it is guaranteed to find the optimal plan for executing the query.

Although traditionally this algorithm requires costing of sub-plans throughout the optimization process, we show here how the costing can be postponed until the end, thus requiring only one round of messages, without any significant impact on the optimization time :

1. Enumerate all feasible joins [37], and multi-joins (Section 2.1) if desired. A *feasible* relation is defined as either a base relation or an intermediate relation that can be generated without a cartesian product; a *feasible* join is defined to be a join of two or more feasible relations that does not involve a cartesian product.
2. Create bid requests for the joins (and multi-joins) computed above and also for scans on the base tables.
3. Request costs from the bidders for these join and scan operations. Note that for each join, we only request the cost of performing that individual join, assuming that the input relations have already been computed (in case the input relations are intermediate tables).
4. Calculate the costs for plans/subplans recursively using classical dynamic programming (partial order dynamic programming if multidimensional costs are desired) and find the optimal plan for the query.

Message Sizes : The size of the message sent while requesting the bids is directly proportional to the number of requests made. The first two columns of Table 1 show the number of bid requests required for different kinds of queries. The vertical axis lists different possible query graph shapes [37], with the *clique* shape denoting the worst possible case for any optimization algorithm. As we can see, the number of bid requests goes up exponentially when multi-join bids are also added.

Plan Space : The plan space explored by this algorithm is exactly the same as the plan space of a System R-style algorithm (modified to search through bushy plans as well). A System R-style optimizer also requires enumeration and costing of all the feasible joins though it does it on demand, and once the costing is done, the two algorithms perform exactly the same steps to find the optimal plan.

[3] Though the original System R algorithm only searched through left-deep plans, in our implementation, we search through bushy plans as well.

3.2. Exhaustive with Exact Pruning

An optimizer may be able to save a considerable amount of computation by pruning away subplans that it knows will not be part of any optimal plan. A top-down approach [21, 20] is more suitable for this kind of pruning than the bottom-up dynamic programming approach we described above, though it is possible to incorporate pruning in that algorithm as well. Typically, these algorithms first find some plan for the query and then use the cost of this plan to prune away those subplans whose cost exceeds the cost of this plan.

The main problem with using this kind of pruning to reduce the total number of bid requests made by the optimizer is that it requires multiple rounds of messages between the optimizer and the data sources. The effectiveness of pruning will depend heavily on the number of rounds of messages and as such, we believe that exact pruning is not very useful in our framework.

But a pruning-free top-down optimizer, that enumerates all the query plans before costing them, will be similar to the System R-style algorithm described above and can easily be used instead of the bottom-up optimizer.

3.3. Dynamic Programming with Heuristic Pruning

With dynamic programming, the number of bid requests required is exponential in most cases. As we will see later, the message time required to get these bids makes the optimization time prohibitive for all but smallest of queries. Since dynamic programming *requires* the costs for all the feasible joins, we can not reduce the number of bid requests without compromising the optimality of the technique.

Heuristic pruning techniques such as Iterative Dynamic Programming (IDP) [33] can be used instead to prune subplans earlier so that the total number of cost estimates required is much less. The main idea behind this algorithm is to heuristically choose and fix a subplan for a portion of the query before the optimization process is fully finished. We experiment with two variants of the iterative dynamic programming technique that are similar to the variants described in [33], except that the bid requests are batched together to minimize the number of rounds of messages :

- **IDP(k) :** We adapt this algorithm as follows :
  1. Enumerate all feasible k-way joins, *i.e.*, all feasible joins that contain less than or equal to k base tables. k ($\leq n$) is a parameter to the algorithm.
  2. Find costs for these by contacting the data sources using a single round of communication.
  3. Choose one subplan (and the corresponding k-way join) out of all the subplans for these k-

Shape of the query	DP w/o multi-joins	DP with multi-joins	IDP(k)[5] (for k $\ll$ n)	IDP-M(k,m)[5] (for k $\ll$ n)
Chain	$O(n^3)$	$O(n2^n)$	$O(n^2k)$	$O(mn^2k)$
Star	$O(n2^n)$	$O(3^{n+1})$	$O(n^k)$	$O(mn^k)$
Clique	$O(3^n)$	$O(2^{2^n})$[5]	$O((2n)^k)$	$O(m(2n)^k)$

Table 1. Number of Bid Requests

way joins using an *evaluation function* and throw away all the other subplans.

4. If not finished yet, repeat the optimization procedure using this intermediate relation and the rest of the relations that are not part of it.

In our experimental study, we use a simple evaluation function that chooses the subplan with lowest cost[4].

- **IDP-M(k,m) :** This is a natural generalization of the earlier variant [33]. It differs from IDP in that instead of choosing one k-way join out of all possible k-way joins, we keep m such joins and throw the rest away, where m is another parameter to the algorithm. The motivation behind this algorithm is that the first variant is too aggressive about pruning the plan space and may not find a very good plan in the end.

Aside from the possibility that these algorithms may not find a very good plan, they seem to require multiple rounds of messages while optimizing. But in fact, the first variant, IDP, can be designed so that the query execution starts immediately after the first subplan is chosen and the rest of the optimization can proceed in parallel with the execution of the subplan. IDP-M(k,m), unfortunately, does not admit any such parallelization.

Message Size : Table 1 shows the number of bid requests required for different query graph shapes. The total cost of costing here also depends on the number of rounds of communication required ($\lceil n/k \rceil$). Decreasing k decreases the total number of bid requests made; however since the startup costs are usually a significant factor in the message cost, the total communication cost may not necessarily decrease.

Plan Space : [33] discusses the plan space explored by this algorithm. It will be a subspace of the plan space explored by the exhaustive algorithm.

3.4. Two-phase Optimization

Two-phase optimization [26] has been used extensively [9, 19] in distributed and parallel query optimization mainly because of its simplicity and the ease of implementation. This algorithm works in two phases :

- **Phase 1 :** Find *the* optimal plan using a System R-style algorithm extended to search through the space of bushy plans as well. This phase assumes that all the relations are stored locally and uses a traditional cost model for estimating costs. If the physical database design is known (*e.g.*, existence of indexes or materialized views on the underlying data sources), then this information is used during the optimization process.

- **Phase 2 :** Schedule the optimal plan found in the first phase. This is done by first requesting the costs of executing the operators at the involved data sources from the bidders and then finding the optimal schedule using an exhaustive algorithm.

Note that the second phase usually requires the use of *partial order dynamic programming* (Section 2.1), even if the optimization goal is minimizing the total cost [33].

Message Size : Since only the joins in one query plan need to be costed, the size of the message is linear in the number of relations involved in the query, unless multi-join bid requests are also made, in which case, the size is exponential in the number of relations.

Plan Space : Though the plan space explored in the first phase is the same as a System R-style algorithm, only one plan is explored in the second phase (which is of more importance since it involves communication).

Considering that only one plan is fully explored by this algorithm, we expected this algorithm to produce much worse plans than the earlier algorithms that explore a much bigger plan space. We were quite surprised to find that it actually produced reasonably good plans. We will revisit this algorithm further in Section 4.3.

3.5. Randomized/Genetic Algorithms

Traditionally, randomized or genetic algorithms have been proposed to replace dynamic programming when dynamic programming is infeasible. The most successful of these algorithms, called 2PO, combines *iterative improvement* (a variant of *hill climbing*) with *simulated annealing* [27]. The problem with any of these randomized algorithms is that they must compute the costs of the plans under consideration (typically the current plan and its "neighbors" in some plan space) after each step, which means the optimizer will require multiple rounds of messages for costing. A natural way of extending these algorithms so as to require fewer rounds of messages, is to find all possible plans that the optimizer may consider in some amount of time, find costs for all of these and then run the optimizer on these costs. But unfortunately, the number of possible plans that the optimizer may consider in next l steps increases exponentially with l. Since typically the number of steps required is quite large, we believe this approach is not feasible in a federated environment.

[4] The techniques described in [33] based on minimum selectivity, etc., can be applied orthogonally. However, since they do nothing to reflect dynamic load and other cost considerations, we focus on cost-based pruning here.

[5] These denote upper bounds on the number of bid requests made by the optimizer.

4. Experimental Study

In this section, we present our initial experimental results comparing the performance of various optimization algorithms that we discussed above. The main goals of this experimental study are to motivate the need for dynamic costing as well as to understand the trade-offs involved in the optimization process. As we have already mentioned, the actual cost of execution is not relevant for evaluating an optimization algorithm, since the only information the optimizer has about the execution costs is through cost models exported by the bidders. As such, we use a simplistic cost model for the bidders as well as for the communication cost. The results we present here should not be significantly affected by the choice of these cost models.

4.1. Experimental Setup

We have implemented the algorithms described earlier in a modified version of the Cohera federated database system [25], a commercialization of the Mariposa research system [45]. The experiments were carried out on a stand-alone Windows NT machine running on a 233MHz Pentium with 96 MB of Memory. Both the optimizer and the underlying data sources connect to a Microsoft SQLServer running locally on the same machine. A set of bidders was started locally as required for the experiments. We simulate a network by using the following message cost model : A message of size N bytes takes $\alpha + \beta * N$ time to reach the other end, where α is the startup cost and β is the cost per byte. We experimented with two different communication settings corresponding to a local area network (LAN) with $\alpha = 10ms$, $\beta = 0.001ms$ and a wide area network (WAN) with $\alpha = 120ms$, $\beta = 0.005ms$.[6]

For the query workload, we use four queries from the TPC-H benchmark. Three of these queries involve a join of 6 relations each, whereas one (Query 8) requires joining 8 relations. We chose this data set since we wanted a realistic data set for performing our experiments. Since we want to concentrate only on the join order optimization, we modify the queries (*e.g.,* by removing aggregates on top of the query) as shown in Figure 2.

4.1.1. Data Distribution and Indexes For the TPC-H benchmark, we used a scaling factor of 1 which leads to the table sizes as shown in Table 2. The experiments were done using four data sources. The distribution of the tables and the indexes is as shown in Table 3.

4.1.2. Cost Model We use a simple cost model based on I/O that involves only Grace hash join and index nested loop join. We do not need to include *nested loop joins* since we assume that there is always sufficient memory to perform hash join in at most two passes. The cost formulas used for these two were as follows :

- **Index Join (Index on R) :** $cost = S_b + |S|(h_R - 1 + \lceil |R| \times s/B_R \rceil)$
- **Grace Hash Join :** $cost = R_b + S_b \; if \; M > min(R_b, S_b), \; R_b + S_b + 2(R'_b + S'_b) \; otherwise$

where M denotes the memory available for the join, B_R denotes the number of tuples of R in a block, $|R|$ denotes the number of tuples in the relation R, R_b denotes the number of blocks occupied by the relation R and R'_b denotes the number of blocks that will be occupied by R after performing any select/project operations that apply to it. h_R denotes the height of a B-Tree index on R and s denotes the selectivity of the join.

Table Name	Number of Tuples	Tuple Size	Table Name	Number of Tuples	Tuple Size
lineitem	6000000	120	orders	1500000	100
partsupp	800000	140	part	200000	160
customer	150000	180	supplier	10000	160
nation	25	120	region	5	120

Table 2. TPC-H Tables (Scaling Factor = 1)

Site	Tables (indexes shown in parantheses)
1	supplier(suppkey), part(partkey), lineitem(partkey), nation, region
2	orders(orderkey), lineitem(orderkey), nation, region
3	supplier(suppkey), part(partkey), partsupp, nation, region
4	orders(orderkey), customer(custkey), nation, region

Table 3. Data Distribution

4.2. Optimization Quality

In this section, we will see how the different optimization algorithms perform under various circumstances and further motivate the need for better costing in the optimization process. The algorithms that we compare are Exhaustive (E)(Section 3.1), Two-Phase (2PO)(Section 3.4) and four variants of Iterative Dynamic Programming(Section 3.3), IDP(4), IDP(3), IDP-M(4,5) and IDP-M(3,5).

4.2.1. Uncertain load conditions : Total cost optimization

For the first experiment, we artificially varied the load on the various data sources involved and also the amount of memory allocated for the joins. The variations in the loads were such that the cost of an operation varied by up to a factor of 20, whereas the memory at each site was chosen to be between $1MB$ to $10MB$. For each of the queries, the six algorithms were run for 40 randomly chosen settings of these parameters. The costs of the best plans found by each of the algorithms were scaled with the optimal plan for the query, found by the exhaustive algorithm. Figure 3(i) shows the mean for these scaled costs as well as the standard deviation for these 40 random runs for each of the algorithms. We show only the results for a wide-area network since the trends observed are similar in the local area network. As

[6] The startup times were obtained by finding the time taken by a ping request to http://www.mit.edu (for WAN) and to a local machine (for LAN) and the time per byte was obtained by finding the time taken to transfer data to and from these sites.

```
select *
from customer c, orders o, lineitem l, supplier
s, nation n, region r
where c.custkey = o.custkey and o.orderkey
= l.orderkey and l.suppkey = s.suppkey and
c.nationkey = s.nationkey and s.nationkey =
n.nationkey and n.regionkey = r.regionkey and
r.name = '[REGION]' and o.orderdate >= date
'[DATE]' and o.orderdate < date '[DATE]' + in-
terval '1' year
```

Query 5 (Q5)

```
select *
from customer c, orders o, lineitem l, supplier
s, nation n1, nation n2
where c.custkey = o.custkey and o.orderkey
= l.orderkey and l.suppkey = s.suppkey and
c.nationkey = n1.nationkey and s.nationkey =
n2.nationkey and ((n1.name = '[NATION1]' and
n2.name = '[NATION2]') or (n1.name = '[NATION2]'
and n2.name = '[NATION1])) and l.shipdate between
date '1995-01-01' and date '1996-12-31'"
```

Query 7 (Q7)

```
select *
from customer c, orders o, lineitem l, supplier
s, part p, nation n1, nation n2, region r
where c.custkey = o.custkey and o.orderkey =
l.orderkey and l.suppkey = s.suppkey and p.partkey
= l.partkey and c.nationkey = n1.nationkey and
n1.regionkey = r.regionkey and s.nationkey =
n2.nationkey and r.name = '[REGION]' and p.type =
'[TYPE]' and o.orderdate between date '1995-01-01'
and date '1996-12-31'"
```

Query 8 (Q8)

```
select *
from orders o, lineitem l, supplier s, part p,
partsupp ps, nation n
where o.orderkey = l.orderkey and l.suppkey
= s.suppkey and p.partkey = l.partkey and
s.nationkey = n.nationkey and ps.suppkey =
l.suppkey and ps.partkey = l.partkey and p.name
like '%[COLOR]%'"
```

Query 9 (Q9)

Figure 2. Modified TPC-H Queries

we can see, though the two-phase algorithm performs somewhat worse than the exhaustive algorithm, in many cases it does find the optimal plan. This counter-intuitive observation can be partially explained by noting that the two-phase optimizer only fixes the join order and not the placement of operators on the data sources involved. Observing the query plans that were chosen by the Exhaustive Algorithm, we found that under many circumstances the optimal query plan was the same as (or very similar to) the plan found by the static optimizer. This was observed for all four queries that we tried (though to a lesser extent for Query 9). We will discuss this phenomenon further in Section 4.3.

The performance of IDP variants as compared to IDP-M variants is as expected , with IDP-M never doing any worse than IDP, though in some cases both algorithms perform similarly. The performance of IDP(3) and IDP(4) shows another interesting phenomenon, *i.e.,* in some cases IDP(3) performed better that IDP(4) (*e.g.*, Query 7). This is mainly because of the artificial constraint imposed by the IDP algorithm of choosing a 3 (or 4) relation subplan in the first stage. This can be demonstrated by the query plan chosen by the two algorithms for Query 7. Figure 4 shows the plans chosen by the Exhaustive algorithm and these two variants for a setting of parameters. As we can see, because of the requirement of choosing the lowest cost subplan of size 4 in the beginning, IDP(4) does not produce the plan chosen by the other algorithms.

4.2.2. Uncertain load conditions : Response time optimization

This experiment is similar to the experiment above except that we changed the optimization goal to be minimizing the response time. Figure 3(ii) shows the relative performance of these optimization algorithms in this case.

As we can see, for two of the queries, Query 5 and Query 9, 2PO finds a much worse plan than the optimal plan. For queries 7 and 8, on the other hand, it performs almost as well as the optimal plan. The main reason for this is that since we have extended the first phase of the two-phase optimizer to search through bushy plans as well, it finds bushy plans for these queries and as such, they can be effectively parallelized. The IDP variants perform almost the same as in the earlier experiment, though with higher deviations in some cases.

4.2.3. Effect of Presence of Materialized Views As we mentioned earlier, the data sources may have materialized views that are not exposed to the optimizer, either because the data source is not exporting this information or it may be generating such views dynamically. For this experiment, we introduce one view in the system, join of the *customer*, *orders* and *lineitem* relations, at Site 2. Since both joins involved in this view are foreign-key joins, the number of tuples in the view is same as the number of tuples in the relation *lineitem*. The static optimizer is not made aware of this view, whereas the Site 2 has access to this information while generating bids. The selections on the relations in the view, if any, are pulled above the view. The rest of the setup, including the indexes, is kept as in the earlier experiment.

Figure 3(iii) shows the results from this experiment for queries Q5 and Q7. The results for Q8 were similar, whereas Q9 is not affected by this view. As we can see, 2PO consistently produces a plan much worse than the optimal plan. IDP variants once again show unpredictable results with IDP-4 performing very well, since it is able to take advantage of the view, whereas IDP-3 performs almost as bad as two-phase optimizer, since the restriction of choosing the lowest cost subplan of size 3 makes it choose the

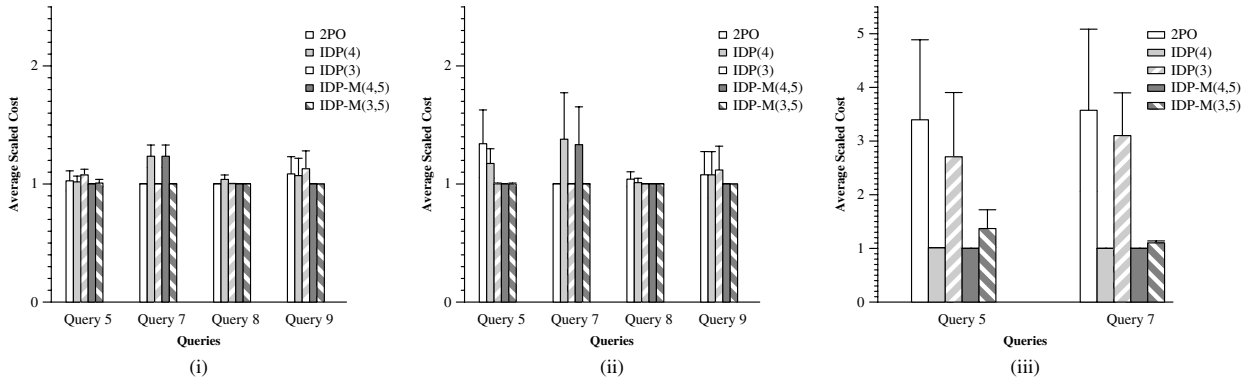

Figure 3. (i) Total cost optimization and (ii) Response time optimization under uncertain load conditions; (iii) Total cost optimization in presence of materialized views

wrong plan.

4.3. Discussion

4.3.1. Iterative Dynamic Programming As we can see, the IDP variants are quite sensitive to their parameters, but in almost all cases, at least one of IDP(3) and IDP(4) performed better than the two-phase optimizer. This suggests that a hybrid of two such algorithms might be the algorithm of choice in the federated environment, especially when the physical designs of the underlying data sources may be hidden from the optimizer. Such a hybrid algorithm will involve running IDP(k_1) and IDP(k_2), for different parameters k_1 and k_2 and then choosing better of the two plans. If k_2 is divisible by k_1, then the plan chosen by IDP(k_1) is clearly going to be worse than the plan chosen by IDP(k_2). Usually, since the number of joins in a query is reasonably small, choosing small values for k_1 and k_2 such that $k_2 = k_1 + 1$ should be ideal. We plan to address this issue in future work.

4.3.2. Two-Phase Optimization The most surprising fact that arises from our experiments is that the two-phase optimization algorithm does not perform much worse than the exhaustive algorithm for total cost optimization if the physical database design is known to the optimizer. In this section, we will try to analyze this phenomenon for our cost models and experimental settings. We will argue that the runtime cost of the plan chosen by the two-phase optimizer can not be more than a small multiple of the runtime cost of the optimal plan.

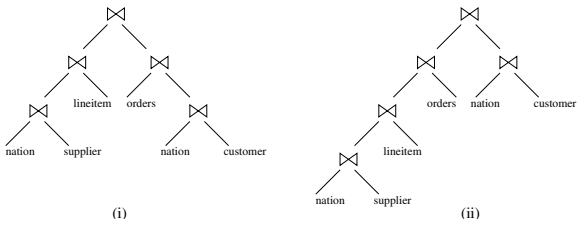

Figure 4. (i) Plan chosen by Exhaustive and IDP(3) for Query 7, (ii) Plan chosen by IDP(4)

Query	Ave % Comm Cost (Std. Dev.)
Query 5	2% ($\pm$ 1.7%)
Query 7	4.6% ($\pm$ 3.6%)
Query 8	1.5% ($\pm$ 1.15%)
Query 9	5.7% ($\pm$ 4.8%)

Table 4. Average % Communication Cost (WAN)

Some key observations about our experimental setup help us analyze this :

1. There was no intra-site or inter-site pipelining. As a result of this, the total execution cost of the plan can be separated out into the costs of execution of each join in the plan.
2. **(a)** The first phase of the two-phase optimizer was aware of the indexes present at the data sources.
   (b) There was always sufficient memory to execute a hash join in at most two passes over the data [43].
   These observations lead us to the following assertion :
   Assertion 1 [7] **:** For a query plan p, if $Cost_{st}(p)$ denotes the cost of the plan as computed by the first phase and $Cost^1_{dyn}(p)$ denotes the cost of the plan under runtime conditions with the loads on the sites being equal (*i.e.*, the cost mark-ups on all the sites are equal to 1 and communication costs being zero, then
   $$1/2 \times Cost^1_{dyn}(p) \leq Cost_{st}(p) \leq 2 \times Cost^1_{dyn}(p)$$
   Intuitively, the factor 2 arises because, based on memory allocated at runtime, a hash join might have to perform two passes instead of one pass over the data.
3. The communication cost during query execution is not a significant fraction of the total execution cost. Table 4 shows the average fraction of the total cost that is spent communicating for the plan found by the exhaustive optimizer. As we can see, communication cost forms a small fraction of the total cost for most of the queries. This holds true even for a wide-area network since the selections and projections in the query plan

[7] Please refer to the full paper [12] for the proofs

make the total data communicated much smaller than the total data read from the disk. Also, all the joins in the TPC-H queries are key foreign-key joins and as a result, the intermediate tables are never larger than the base tables and are usually much smaller.

Given these observations, we can speculate as to why two-phase may be performing as well as our experimental study shows. We will consider various factors that might have an impact on the runtime execution time of a query plan in turn and reason that the difference between the runtime execution cost of the plan found by the two-phase optimizer and the optimal runtime plan can not be large.

Let the optimal plan found by the first phase of the two-phase algorithm be $Plan_{st}$ and let the optimal plan under runtime conditions as found by the exhaustive algorithm be $Plan_{dyn}$. Since the two-phase algorithm finds the best static plan, we have that $Cost_{st}(Plan_{st}) \leq Cost_{st}(Plan_{dyn})$.

- If the loads on all the sites are equal and communication costs are zero, then using Assertion 1 and assuming worst case memory conditions ($Plan_{dyn}$ requires only one pass for every hash join, whereas $Plan_{st}$ requires two passes for every hash join),

  $Cost^1_{dyn}(Plan_{st}) \leq 4 \times Cost^1_{dyn}(Plan_{dyn})$

- If we let f be the fraction of total run-time cost of $Plan_{st}$ that is communication cost, then under the worst case assumption that $Plan_{dyn}$ does not incur any communication cost, we get that

  $Cost^2_{dyn}(Plan_{st}) \leq 4/(1-f) \times Cost^2_{dyn}(Plan_{dyn})$

  where $Cost^2_{dyn}()$ is the cost of the plan at runtime including the communication costs. If $f < 1/2$, then $4/(1-f) < 8$ and usually, f is going to be much smaller (cf. Table 4).

- Finally, the impact of dynamically changing load conditions is mitigated because our two-phase optimizer schedules the joins at run-time taking into account the load conditions. All the joins in the query will (probably) be scheduled on lightly loaded sites. It is possible that $Plan_{st}$ may incur more communication cost as a result of this, but as we have already argued, the communication cost does not form a major factor of the total query execution cost.

Inspite of worst case assumptions, runtime execution cost of the statically optimal plan is going to be less than a small multiple of the dynamically optimal plan in our experiments. We observe much smaller ratios than this, and the main reason behind that might be that these two plans $Plan_{st}$ and $Plan_{dyn}$ do not differ much in the joins they contain. We believe that this is an artifact of the shape of the query plan topology [28]. If every other local minimum in the query plan space has much higher cost that the absolute minimum ($Plan_{st}$) (*i.e.*, the plan space has a deep *well* in it [28]), then the optimal plan under different runtime conditions may all turn out to be very similar to the statically optimal plan.

Our experimental observations and the analysis above suggest that this phenomenon may not be limited to our experimental setup and the cost model. We believe that this applies to a much more general scenario, and we plan to address this issue in a more general scenario in future.

4.4. Optimization Overheads

In this subsection, we look at some of the trade-offs in optimization overheads of these algorithms.

The optimization overheads for IDP variants depend significantly on the parameter k of the algorithm. Figure 5(i) shows the optimization time for a star query of size 10 for various values of parameter k for WAN (similar trends are observed for LAN). We intentionally chose a large query for IDP, since for smaller queries, it is often more efficient to use exhaustive optimization. As we can see, the algorithm is significantly faster for smaller values of k, approaching the cost of the Exhaustive algorithm as k approaches the number of relations. Note that, even though increasing k decreases the number of rounds of messages (*e.g.*, $k = 3$ requires 4 rounds, whereas $k = 4$ requires 3 rounds), the total communication cost may not necessarily decrease, because the increase in the total number of RFBs made by the optimizer may offset the savings due to smaller number of rounds of messages.

Finally, we look at the optimization times of the algorithms that we compared in the earlier sections for the 4 TPC-H queries. We compare only the *total cost optimization* times, though we still need to use partial order dynamic programming. As we can see (Figure 5), 2PO takes much less time than most of the other algorithms, with IDP-M(3,5) taking the most time for both configurations. We can see the effect of high message costs under the WAN configuration with the cost of exhaustive algorithm (one message) dropping below the cost of other algorithms with more messages. Also, IDP(3,5) incurs much higher cost for Query 8 as it requires 3 rounds of messages.

4.5. Summary of the Experimental Results

Our experiments on the modified TPC-H benchmark demonstrate the need for aggressive optimization algorithms that take into account dynamic runtime conditions while optimizing, and of algorithms that require very few messages to accumulate the required cost information from the underlying data sources. The Exhaustive Dynamic Programming algorithm, modified to use a single message, works reasonably well for small queries, but for large queries, a heuristic algorithm such as IDP may have to be used. We found IDP to be very sensitive to its parameters (particularly, the parameter k), and running IDP in parallel for two different values of the parameter may work best in practice.

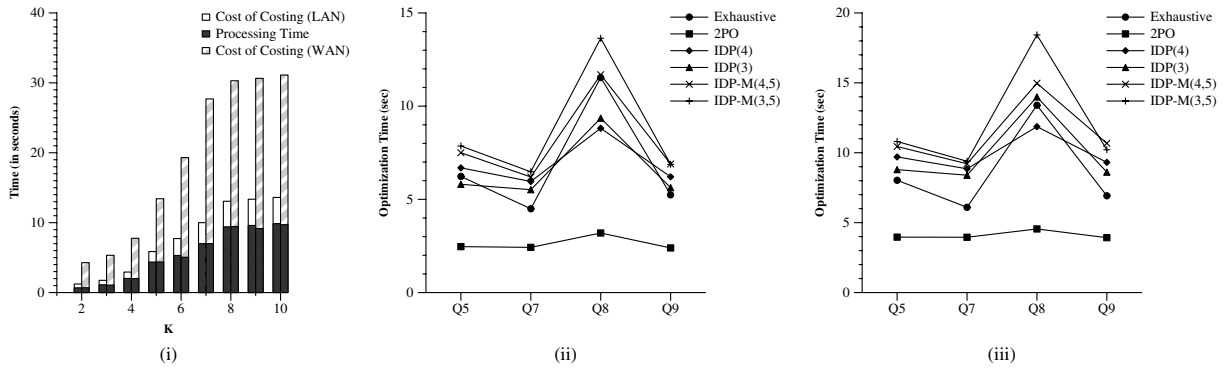

Figure 5. (i)IDP optimization time for a 10 relation star query (WAN); (ii) Optimization time for TPC-H Queries for LAN; (iii) for WAN

Another surprising observation from our experiments was that the two-phase optimization algorithm performed reasonably well in many cases, especially when the optimizer knew about the materialized views and the indexes at the data sources and the optimization goal was minimizing total cost. We have tried to analyze this observation for our experimental settings; this is clearly an interesting direction for future work.

5. Related Work

In this section, we will try to put the work presented in this paper in perspective. To explore the design space for a query optimizer for the federated environment, we will concentrate on the two factors that most significantly affect the complexity of the optimization process : *dynamic load conditions*, and *unknown cost models*. Using these two factors, the design space for a federated query optimizer can be divided into four categories, as shown in Table 5.

In this paper, we have focused on the uppermost category, where the cost models for the underlying data sources are not known to the optimizer and the optimizer has to consider load conditions while optimizing. Note that the top row makes the least assumptions about the environment, and hence any algorithms developed for this scenario can be applied to any of the other scenarios.

From the query optimization perspective, the simplest of these scenarios is the lowest row, where the cost models for the underlying data sources are known to the optimizer and the optimization goal is to minimize the total execution cost. Earliest distributed database systems such as R* [22] were built with such assumptions about the environment. The R* optimizer was an extended version of the System R optimizer with an extra term added in the execution cost formula for communication cost. The Iterative Dynamic Programming [33] (Section 3.3) can be used if the exhaustive algorithm turns out to be too complex in time or space; this is not unlikely in a distributed scenario even with small queries. Randomized algorithms have also been proposed for complex join queries [27, 34].

The second row from the bottom is relevant even in centralized database systems, where run-time conditions can significantly affect the execution cost of a query plan. [11, 29, 16, 10] discuss how *parametric optimization* can be used to compute a set of plans optimal for different values of the run-time parameters, instead of just one plan, and then to choose a plan at run-time when the values of parameters are known. [18] discuss how these techniques may be extended to distributed databases, where the loads on the underlying databases may not be known at optimization time. The *two-phase optimization* approach (Section 3.4) was also first proposed for this scenario [26] and was later also used in the Mariposa system [45].

The second row from the top focuses on the heterogeneous nature of federated databases, without considering any dynamic runtime issues. Heterogeneous database systems present the challenge of incorporating the cost models of the underlying data sources into the optimizer. One solution to this problem is to try to learn the cost models of the data sources using "calibration" or "learning" techniques [49, 13, 8]. Middleware systems such as Garlic [23] have chosen to solve this problem through the use of *wrappers*, which are programmed to encapsulate the cost model and capabilities of a site. The cost of costing is not significant in Garlic, since the wrappers execute in the same address space as the optimizer. Garlic uses exhaustive dynamic programming to find the optimal plan, and though we are not aware of any explicit work in this area that uses the other optimization techniques, it should not be difficult to extend those for this scenario.

[15, 38, 41] discuss query optimization issues in a loosely coupled federated system such as modelled by the top row. From the query optimization perspective, the main focus of [15, 38] is on incorporating the knowledge of statistics and system parameters gained while executing the query at run-time, into the query plan. They use a statistical decision mechanism that schedules the query plan a join at a time. [41] propose a mechanism for a loosely coupled multi-database system where all the underlying databases

	Exhaustive	Heuristic Pruning	Two-Phase	Randomized
Dynamic, Unknown Cost Models	Y	Y	Y	Y
Static, Unknown Cost Models	*e.g.* Garlic[23]	Y	Y	Y
Dynamic, Known Cost Models	Parametric Optimization[11, 29]	Y	Mariposa [45]	[34]
Static, Known Cost Models	R*[22]	IDP[33]	N/A	Simulated Annealing, 2PO[27]

Table 5. Design Space for a Federated Optimizer ("Y" denotes scenarios considered in this paper)

cooperate to find the optimal plan. This algorithm is not very suitable for a federated database system as the environment may not be completely cooperative and also, the number of messages that are exchanged between the underlying data sources increases exponentially with the size of the query.

Mid-execution re-optimization [31, 30], Query scrambling [47] and Eddies [4] have also been proposed for use in dynamic environments when the required statistics may not be accurately estimated at the optimization time or when the characteristics of the underyling data sources may change dramatically during the query execution. The kind of databases we considered in this paper are more static in that respect, as we assume the existence of relevant metadata, including statistics about the data sources; we do not target quickly fluctuating performances as suggested in Eddies. Also, all of these techniques work in a centralized fashion accessing data from various data sources, but executing the queries on a single machine, whereas in a federated database, we are more interested in distribution of work among the participating data sources.

6. Conclusion and Future Work

Uncertain load conditions and unknown cost models make decoupled query optimization a necessity for federated database systems. To decouple cost computations from the optimization process, the optimizer must consult the data sources involved in an operation to find the cost of that operation. This changes the trade-offs involved in the optimization process significantly, since the dominant cost in optimization becomes the cost of contacting the underlying data sources. Because of these new trade-offs, optimization techniques such as randomized/genetic algorithms, that by nature require multiple rounds of messages, are rendered impractical in this scenario.

In this paper, we presented minimum-communication adaptations of various well-known query optimization algorithm and discussed the trade-offs in their performance. Our experimental results on the TPC-H benchmark indicate that, in many cases, especially when the physical database design is known to the optimizer, two-phase optimization works very well. In absence of such information, more aggressive optimization techniques must be used. We also found that the Iterative Dynamic Programming technique is very sensitive to its parameters, though running IDP with multiple parameter choices may work best in practice. We plan to address both these issues, the surprising effectiveness of two-phase optimization algorithms and the best way to combine IDP variants, in future.

Acknowledgements

We would like to thank everyone at Cohera Corporation, and in particular, Dr. Wei Hong, for their invaluable help during the initial stages of this project. We would also like to thank Cohera Corporation for letting us use their codebase for this project. We would also like to thank Prof. Mike Franklin for his invaluable comments on an earlier draft of this paper. This work was supported by California Micro Grant, CONTROL 442427-21389 and Sloan Foundation Fellowship.

References

[1] Net market makers inc. http://www.netmarketmakers.com, 1999.

[2] S. Abiteboul, S. Cluet, T. Milo, P. Mogilevsky, J. Siméon, and S. Zohar. Tools for data translation and integration. *IEEE Data Engineering Bulletin*, 1999.

[3] S. Adali, K. S. Candan, Y. Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *SIGMOD*, 1996.

[4] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *SIGMOD*, 2000.

[5] R. Avnur, J. M. Hellerstein, B. Lo, C. Olston, B. Raman, V. Raman, T. Roth, and K. Wylie. Control: Continuous output and navigation technology with refinement on-line. In *SIGMOD*, 1998.

[6] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 1986.

[7] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. B. R. Jr. Query processing in a system for distributed databases (SDD-1). *TODS*, 1981.

[8] J. Boulos, Y. Viémont, and K. Ono. Analytical Models and Neural Networks for Query Cost Evaluation. In *Proc. 3rd International Workshop on Next Generation Information Technology Systems*, 1997.

[9] C. Chekuri, W. Hasan, and R. Motwani. Scheduling problems in parallel query optimization. In *PODS*, 1995.

[10] R. Cole. A decision theoretic cost model for dynamic plans. *IEEE Data Engineering Bulletin*, 2000.

[11] R. L. Cole and G. Graefe. Optimization of dynamic query evaluation plans. In *SIGMOD*, 1994.
[12] A. Deshpande and J. Hellerstein. Decoupled query optimization for federated database systems. Technical Report UCB//CSD-01-1140, University of California, Berkeley, 2001.
[13] W. Du, R. Krishnamurthy, and M.-C. Shan. Query optimization in a heterogeneous dbms. In *VLDB*, 1992.
[14] R. S. Epstein, M. Stonebraker, and E. Wong. Distributed query processing in a relational data base system. In *SIGMOD*, 1978.
[15] C. Evrendilek, A. Dogac, S. Nural, and F. Ozcan. Multidatabase query optimization. *Distributed and Parallel Databases*, 1997.
[16] S. Ganguly. Design and analysis of parametric query optimization algorithms. In *VLDB*, 1998.
[17] S. Ganguly, W. Hasan, and R. Krishnamurthy. Query optimization for parallel execution. In *SIGMOD*, 1992.
[18] S. Ganguly and R. Krishnamurthy. Parametric distributed query optimization based on load conditions. In *Sixth International Conference on Management of Data (COMAD)*, 1994.
[19] M. N. Garofalakis and Y. E. Ioannidis. Parallel query scheduling and optimization with time- and space-shared resources. In *VLDB*, 1997.
[20] G. Graefe. The cascades framework for query optimization. *IEEE Data Engineering Bulletin*, 1995.
[21] G. Graefe and W. J. McKenna. The volcano optimizer generator: Extensibility and efficient search. In *ICDE*, 1993.
[22] L. Haas, P. Selinger, E. Bertino, D. Daniels, B. Lindsay, G. Lohman, Y. Masunaga, C. Mohan, P. Ng, P. Wilms, and R. Yost. R*: A research project on distributed relational dbms. *IEEE Data Engineering Bulletin*, 1982.
[23] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *VLDB*, 1997.
[24] J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom. Information translation, mediation, and mosaic-based browsing in the tsimmis system. In *SIGMOD*, 1995.
[25] J. M. Hellerstein, M. Stonebraker, and R. Caccia. Independent, open enterprise data integration. *IEEE Data Engineering Bulletin*, 1999.
[26] W. Hong and M. Stonebraker. Optimization of parallel query execution plans in xprs. In *PDIS*, 1991.
[27] Y. E. Ioannidis and Y. C. Kang. Randomized algorithms for optimizing large join queries. In *SIGMOD*, 1990.
[28] Y. E. Ioannidis and Y. C. Kang. Left-deep vs. bushy trees: An analysis of strategy spaces and its implications for query optimization. In *SIGMOD*, 1991.
[29] Y. E. Ioannidis, R. T. Ng, K. Shim, and T. K. Sellis. Parametric query optimization. In *VLDB*, 1992.
[30] Z. G. Ives, D. Florescu, M. Friedman, A. Y. Levy, and D. S. Weld. An adaptive query execution system for data integration. In *SIGMOD*, 1999.
[31] N. Kabra and D. J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *SIGMOD*, 1998.
[32] L. Knight. "the e-market maker revolution", dataquest inc. http://www.netmarketmakers.com/documents/perspective1.pdf, 1999.
[33] D. Kossmann and K. Stocker. Iterative dynamic programming: a new class of query optimization algorithms. *ACM TODS*, 2000.
[34] R. S. G. Lanzelotte, P. Valduriez, and M. Zaït. On the effectiveness of optimization search strategies for parallel execution spaces. In *VLDB*, 1993.
[35] R. J. Miller, L. M. Haas, and M. A. Hernández. Schema mapping as query discovery. In *VLDB*, 2000.
[36] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *VLDB*, 2000.
[37] K. Ono and G. M. Lohman. Measuring the complexity of join enumeration in query optimization. In *VLDB*, 1990.
[38] F. Ozcan, S. Nural, P. Koksal, C. Evrendilek, and A. Dogac. Dynamic query optimization on a distributed object management platform. In *Proceedings of the fifth international conference on Information and knowledge management*, 1996.
[39] C. Papadimitriou and M. Yannakakis. Multiobjective query optimization. In *PODS*, 2001.
[40] M. T. Roth, F. Ozcan, and L. M. Haas. Cost models do matter: Providing cost information for diverse data sources in a federated system. In *VLDB*, 1999.
[41] S. Salza, G. Barone, and T. Morzy. Distributed query optimization in loosely coupled multidatabase systems. In *ICDT*, 1995.
[42] 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 *SIGMOD*, 1979.
[43] L. D. Shapiro. Join processing in database systems with large main memories. *TODS*, 1986.
[44] A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 1990.
[45] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An economic paradigm for query processing and data migration in mariposa. In *PDIS*, 1994.
[46] A. Tomasic, R. Amouroux, P. Bonnet, O. Kapitskaia, H. Naacke, and L. Raschid. The distributed information search component (DISCO) and the world wide web. In *SIGMOD*, 1997.
[47] T. Urhan, M. J. Franklin, and L. Amsaleg. Cost based query scrambling for initial delays. In *SIGMOD*, 1998.
[48] C. T. Yu, Z. M. Ozsoyoglu, and K. Lam. Optimization of distributed tree queries. *JCSS*, 1984.
[49] Q. Zhu and P.-Å. Larson. A query sampling method of estimating local cost parameters in a multidatabase system. In *ICDE*, 1994.