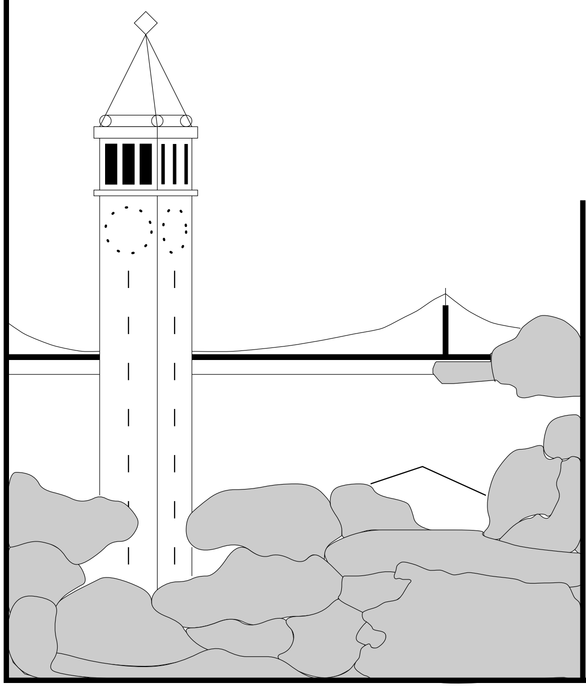
Decoupled Query Optimization for Federated Database Systems

Amol V. Deshpande and Joseph M. Hellerstein
CS Division, EECS Department
University of California, Berkeley.
{amol,jmh}@cs.berkeley.edu



Report No. UCB/CSD-01-1140

April 2001

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Decoupled Query Optimization for Federated Database Systems

Amol V. Deshpande and Joseph M. Hellerstein
CS Division, EECS Department
University of California, Berkeley.
{amol,jmh}@cs.berkeley.edu

April 2001

Abstract

We study the problem of query optimization in federated 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*) [net99, Kni99], whose business hinges on federating thousands of decentralized catalogs and other databases.

Broadly considered, federated database technology has been the subject of multiple research thrusts, including schema integration, data transformation, 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 [HSB$^+$82, ESW78, BGW$^+$81]), and most recently has been focused on linking data sources of various capabilities and cost models. 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 [SDK+94]. 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. This flexibility is often constrained by various issues, including administrative constraints [SDK+94] and limited site capabilities [FLMS99]. Nonetheless, a federated system should not be restricted to centralized processing – in order to perform and scale, it must consider all data and computation resources available at any time.
- **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 [SDK+94], but typically there are other costs that are valuable to expose orthogonally, including response time [GHK92], data freshness [OW00], and accuracy of computations [AHL+98].
- **Decoupling of Cost Estimation**: Regardless of the number of cost dimensions, a centralized optimizer cannot accurately estimate the costs of operations at many autonomous sites, which may depend on transient system issues including current loads and temporal administrative policies [SDK+94]. 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. In addition, the time to transfer data across a network among the various components must also be modeled.

Many of these decouplings have been studied before individually in the context of distributed, heterogeneous or federated database research [SBM95, EDNO97, ONK+96]. However, to the best of our knowledge, the decoupling of cost estimation, which requires the optimizer to communicate with 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 Cost of Costing

To get an idea of how much cost of costing might affect the optimization time, we plot the time required to obtain the costs as a function of the number of operations for which costs are required, on a wide-area network. This experiment was run with the optimizer communicating with a single data source over a simulated wide-area network for a set of artificially constructed queries[1]. As we can see in Figure 1, the naive method of sending a message for each operation turns out to be infeasible for even the smallest of the

[1] The message cost model used is described in detail in Section 6.1

queries (*e.g.,* the number of operations that need to be costed for the TPC-H Query 5 is about 62.) Even when all the requests are batched together in a single message, the time required to find the costs is quite significant and hence, the optimizer has to be careful in choosing the operations for which it requests costs.

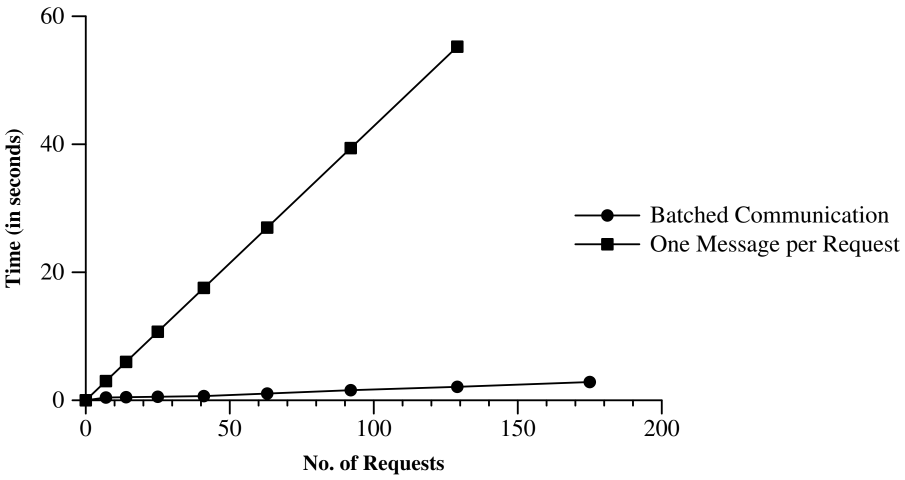


Figure 1: Cost of Costing

1.2 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 and we present experimental results on modified TPC-H benchmark queries.

Our experimental results, somewhat surprisingly, suggest that the simple technique of breaking the optimization process into two phases [HS91] — first finding the best query plan under static conditions and then scheduling it across the federation based on the run time conditions — works very well in 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 previous 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 6.2.4). Our analysis suggests that this behavior may not merely be a peculiarity of our experimental settings, but may hold true in general.

1.3 Overview of the paper

We begin by describing the design space for a federated query optimizer and place prior work in context (Section 2). We then briefly describe the architecture of the federated system that we use for our experiments (Section 3) and discuss additional factors that a federated optimizer has to consider (Section 4). In Section 5, we present minimum communication decoupled variants of various well-known optimization techniques and discuss the trade-offs in their performance. Finally, we present the results of an empirical study of these algorithms on the TPC-H benchmark (Section 6).

2 Design Space and Related Work

To explore the design space for a federated optimizer, we concentrate on the two factors that most significantly affect the complexity of the optimization process :

- **Dynamic Load Conditions :** Run-time conditions including loads on the underlying data sources[2] and communication costs can have a significant impact on the execution cost of a query.
- **Unknown Cost Models :** As we have already discussed, the heterogeneous nature of the system makes it impossible for the optimizer to know the cost models of all the underlying data sources, which may depend on implementation issues as well as database design issues (e.g. the presence of indexes or materialized views).

These two factors divide the design space for a federated query optimizer into four categories, as shown in the vertical axis of Table 1. On the horizontal axis, we list various well-known optimization techniques that have been developed over the years.

	Exhaustive	**Heuristic Pruning**	**Two-Phase**	**Randomized**
Dynamic, Unknown Cost Models	Y	Y	Y	Y
Static, Unknown Cost Models	Garlic[RAH+96]	Y	Y	Y
Dynamic, Known Cost Models	Parametric Optimization[CG94, INSS92]	Y	Mariposa [SDK+94]	[LVZ93]
Static, Known Cost Models	R*[HSB+82]	IDP[KS]	N/A	Simulated Annealing, 2PO[IK90]

Table 1: Design Space for a Federated Optimizer. The vertical axis contains various cost scenarios, while the horizontal axis contains various optimization algorithms. "Y" denotes combinations that we consider in this paper; "N/A" denotes a scenario where the optimization algorithm is not applicable.

As we can see, there are various possibilities that have not been studied before. In this paper, we will focus 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

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

makes the least assumptions about the environment, and hence any algorithms developed for this scenario can be applied to any of the other scenarios. It may be possible in some cases to develop better algorithms for the more constrained scenarios however – e.g., by pruning more aggressively given the knowledge of the constraints. We proceed to consider each of the rows in the table in turn.

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. The earliest distributed database systems such as R* [HSB$^+$82] 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. Recently, Kossmann *et al.*, [KS] proposed an alternative to exhaustive dynamic programming, called Iterative Dynamic Programming (IDP), which 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. We discuss this algorithm in more detail in Section 5. Various randomized algorithms have also been proposed for complex join queries containing a large number of joins [IK90]. To combat large plan space in a parallel environment, [LVZ93] introduce a new randomized algorithm called *Toured Simulated Annealing*.

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. [CG94, INSS92, Gan98, Col00] 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. [GK94] 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 begins to make sense in this context.[3] Initially proposed for the XPRS parallel database system [HS91], the two-phase approach divides optimization into two parts: finding the best single-site plan, and scheduling it at run time based on the current conditions. Mariposa [SDK$^+$94] used a similar approach for federated optimization, first finding the best single-site plan, and then using a *bidding mechanism* to gather information about loads on the underlying systems, and to schedule the plan found in the first phase.

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 [ZL94, DKS92, BVO97] . Middleware systems such as Garlic [RAH$^+$96] 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 belong to the middleware, and in fact 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.

A fully decoupled federated database system is modeled by the top row – no assumptions can be made about either underlying cost models or transient costs. In a fully decoupled environment, the federated

[3] [IK90] use the term *two-phase optimization* for a hybrid randomized algorithm they propose; we follow this usage of the term from [HS91].

DBMS cannot know about the implementation or policies of the autonomous sites in the federation, which are free to change their implementations, physical database design, and cost estimation policies at will, perhaps even via automated mechanisms [SDK$^+$94]. Moreover, the uncertain load conditions and communication costs can have a very significant impact on the execution cost of a query, especially when optimizing for response time. [EDNO97, ONK$^+$96, SBM95] discuss query optimization issues in such loosely coupled federated systems. From the query optimization perspective, the main focus of [EDNO97, ONK$^+$96] 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. [SBM95] propose a mechanism for a loosely coupled multi-database system where all the underlying databases 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.

Apart from traditional query optimization techniques, [AH00] propose a new adaptive architecture called *eddies* for query processing. Though eddies are also designed to function well in highly dynamic environments, they are geared toward adapting to the idiosyncrasies of the data sources during the execution of query, rather than before the execution of the query. The kind of databases we consider are more static in that respect, as we assume the existence of relevant metadata, including statistics about the data sources. Also, as described in [AH00], eddies 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.

3 Architecture and Terminology

For our experimental study, we use the Cohera federated database system that is based on the Mariposa research system [SDK$^+$94], extended to handle heterogeneous data sources including various relational databases, legacy systems, and web data sources (HTML or XML). As part of its support for Mariposa's *economic paradigm* [SDK$^+$94], this system cleanly decouples various aspects of the optimization process as described previously. 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 economic computing, 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 how much an operation costs, 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 or processing. Local autonomy is critical in any federated system that spans organizational boundaries, as one might expect to find in both the Internet context and even within large enterprises.

In this section, we briefly discuss the architecture of this system and introduce the terminology we use

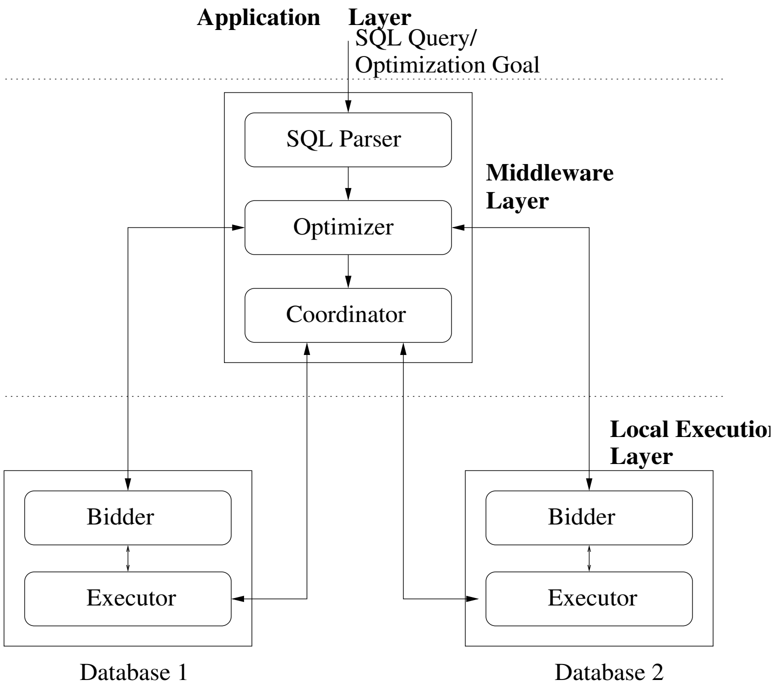


Figure 2: Architecture of Cohera Federated System

for the rest of the paper.

3.1 Architecture

Our system consists of three layers :

- **Application Layer :** This consists of the client program that interacts with the user. It passes the user query along with an optimization goal to the middleware for execution.
- **Middleware Layer :** The middleware layer consists of a *query parser*, an *optimizer* and a *coordinator*. A *metadata manager* (not shown in the figure) is used to get the relevant information about the tables such as their locations, statistics regarding cardinalities and the distribution of join attributes etc. The optimizer uses this information to formulate a set of operations to be performed, communicates with the data sources to compute the costs of the operations (using the bidding constructs as described below) and formulates a query plan. If a plan matching the optimization constraints is not found, then the client application is so notified, otherwise the query plan is passed to the coordinator that coordinates the execution of the query.
- **Local Execution Layer :** Each underlying site exposes two modules to the middleware layer, the *bidder* and the *executor*. The bidder participates in the query optimization process with the optimizer by making cost estimates on parts of queries. In Mariposa [SDK$^+$94], it was argued that the bidder has an incentive to be honest and accurate about the costs. The problem of dishonest/inaccurate bidders is similar in nature to the cost estimation problem that a single-site optimizer faces in that, if the bidders

don't make accurate bids, then the plan found may not turn out to be optimal during query execution. The issue of accurate cost estimation is orthogonal to the discussion in this paper. For the remainder of this paper, we assume that the bidders are "perfect", *i.e.*, the bidders export the cost of an operation accurately.

3.2 Terminology

We will use the following terminology (extended from Mariposa [SDK+94]) in the rest of the paper.

- **Request for Bid (RFB) :** An RFB is a construct used by the optimizer to find the execution cost of an operation on a remote data source. An RFB consists of the operation(s) to be performed and may contain relevant statistics if necessary (*e.g.,* if the input to a join operation is an intermediate table, then the optimizer will have to include the relevant statistics about the table in the corresponding RFB).
- **Bid :** The data source (more specifically, the bidder at the source) uses this construct to respond to an RFB. In most cases, the response will contain the estimated execution cost (possibly multi-dimensional) of the operation, but it can also be used to tell the optimizer about staleness/accuracy of the data.
- **Cost Mark-up :** This construct abstracts the load conditions on the site to a number that is used to scale all the bids that are made by the site. This number will usually depend on the load conditions at the data sources, but may sometimes be dictated by the administrative policies. We use this construct to control the load on a data source as seen by the optimizer in our experiments.

4 Federated Query Optimization

The query optimization problem is to find an execution plan 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 terms of *money*), 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.

4.1 What an Optimizer Needs to Consider

The plan space that a query optimizer has to search through depends on a large number of factors, some of which affect the search space quite significantly. We have already discussed two of the most important factors: dependence of execution costs on runtime conditions and unknown cost models. Here we enumerate additional factors that an optimizer needs to consider, and we analyze their effect on the optimization process.

4.1.1 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 [OW00], accuracy of the data [AHL$^+$98] *etc.* may also be critical. As [GHK92] point out, minimizing for such an optimization goal requires the use of *partial order dynamic programming* techniques.

Partial-order dynamic programming is a generalization of the classical dynamic programming algorithm where the cost of each plan is computed as an l-dimensional vector (where l depends on the particular scenario) and two costs are considered incomparable if neither is less than or equal to the other in all l dimensions. With each feasible subset of tables, a partial order DP scheme keeps a set of incomparable subplans, instead of just a single subplan that would be kept in a classical dynamic programming approach. In the end, when this set of incomparable plans is computed for the query, a plan is chosen according to the optimization goal[4]. It can be shown that if the cost is an l-dimensional vector, then the space and time complexity of the optimization process increases by an expected factor of 2^l [GHK92]. Recently, a polynomial time approximation algorithm has also been proposed for this problem [PY01].

As [KS] 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.

4.1.2 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 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 the site, or it could assume that there is no intra-site pipelining. Either assumption could result in incorrect estimation of the cost.

To solve this problem, we can allow for *multi-join bid requests* where the optimizer sends a bid request consisting of multiple relations (some of which may be intermediate relations) and the bidder is asked to make a bid on the join involving all these relations. Now, if the site has enough resources, the bidder can make use of them by pipelining the joins.

As one can imagine, this results in further explosion in the plan space, making optimization time exponential even for *chain queries* that can traditionally be optimized in polynomial time [OL90]. We will discuss the increase in plan space further in Section 5.

[4] Partial-order dynamic programming can also be thought of as a generalization of the *interesting orders* of System R [SAC$^+$79], 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.

4.2 Trade-offs in the Optimization Process

The time spent in optimization can be roughly divided into two parts :

- **CPU Time :** This includes the time taken by the optimizer to plan which RFBs to make (called pre-processing cost henceforth), time taken by the sites to decide what bids to make and the time taken by the optimizer for processing the resulting bids (post-processing cost). The time taken for planning bids is directly proportional to the number of bid requests made to the site, whereas the pre-processing and post-processing costs depend heavily on the optimization algorithm chosen and also on the connectivity of the query graph.
- **Message Time :** This includes the actual communication time during optimization, as well as the time taken for processing the message at both the ends. It depends on the number of rounds of messages that are performed and also on the total number of bid requests and responses from the sites. The size of each bid request plays an important role in this. Each bid request must contain the relevant information about the relations involved in the operation such as the name of the relation (if it is a base table), the statistics of the relations (since some of the relations may be intermediate relations) and also the details of the operation such as join/select predicates.

Considering that optimization times are traditionally about several seconds and the message startup times observed on Internet for WAN are of the order of few hundreds of milliseconds (Figure 1), the message time will be a significant part of the optimization time, even if the optimization process involves very few messages. We will study these trade-offs in more detail in Section 6.

4.3 Simplifying Assumptions

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

- **Relational Sources :** We assume that all the underlying databases are relational, or at least export a relational interface via some kind of a gateway tool [coh].
- **Accurate Statistics :** We assume that statistics regarding the cardinalities and the selectivities are available. There are two reasons why we feel justified in making this assumption :
  - Standard protocols such as ODBC/JDBC allow querying the host database about statistics.
  - As has been suggested elsewhere [ACPS96], it is possible to cache the statistics from earlier query executions.
- **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. Such information may be maintained by periodically probing the communication links and getting the latest statistics. It is also possible to get these costs at execution time by probing the communication links directly for the current statistics. These probes can be piggybacked on other messages sent to the bidders.
- **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 may waste

resources while waiting for each other. This is especially a problem with *space shared* resources like memory that can not be time shared between different processes easily [GI97]. Even if the producer is not slow, the communication link between the two sites can be slow and the consumer will be waiting for the network while holding the resources[5]. This could result in significant waste of resources under WAN where the communication times are usually high.

5 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.

5.1 Classical Dynamic Programming (Exhaustive):

Dynamic Programming, first proposed for System R [SAC$^+$79], is still the most prevalent optimization technique used in database systems today. The algorithm exhaustively searches through all plans for the query, using dynamic programming[6] and the principle of optimality to prune away bad subplans as early as possible. 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 without any significant impact on the optimization time. We adapt this algorithm to the federated scenario as follows, requiring only one round of messages to do costing :

1. Enumerate all feasible joins [OL90] and multi-joins (Section 4.1.2) 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 itself does not

[5] Although the ability to fetch records from foreign data streams was proposed as early as the original INGRES system [ESW78], commercial database systems have only recently started providing the functionality to fetch records from live data streams , e.g. in IBM DB2 UDB [RPK$^+$99], or Informix Dynamic Server UDO [SBH98]

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

involve a cartesian product.[7]

2. Create the bid requests for the joins (and multi-joins) computed above and also for scans on the base relations.
3. Request costs from the bidders for all feasible joins/multijoins. Note that for each join, we only request the cost of the 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 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 2 show the number of bid requests required for different kinds of queries. The vertical axis lists different possible query graph shapes [OL90], 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). System-R 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.

5.2 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 [GM93, Gra95] 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.

5.3 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.

[7]For simplicity of discussion, we ignore the case where a cartesian product is *required* for executing the query. This is a minor extension that is easy to add.

Heuristic pruning techniques such as Iterative Dynamic Programming [KS] can be used instead to prune subplans earlier so that the total number of costs required is much less. But to do pruning, we need to find the costs for the subplans under consideration, which requires contacting the bidders.[8] As we will see later, this introduces a trade-off between the number of rounds of messages and the total number of bid requests that are sent out. Also, the plan found at the end may not be the optimal plan, as these greedy strategies may throw away the optimal plan. We experiment with two variants of the iterative dynamic programming technique that are similar to the variants described in [KS], except that the bid requests are batched together to minimize the number of rounds of messages :

- **IDP(k) :** We adapt this algorirhtm as follows :

  1. Enumerate all feasible k-way joins, *i.e.*, all feasible joins that contain less than or equal to k base relations. $k(\leq n)$ 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-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.

- **IDP-M(k,m) :** This is a natural generalization of the earlier variant [KS]. 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 2 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 : [KS] discusses the plan space explored by this algorithm. It will be a subspace of the plan space explored by the exhaustive algorithm above.

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

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

Shape of the query	DP w/o multi-joins	DP with multi-joins	IDP(k)[9] (for k $\ll$ n)	IDP-M(k,m)[9] (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 2: Number of Bid Requests

5.4 Two-phase Optimization :

Two-phase optimization [HS91] has been used extensively [CHM95, GI97] in distributed and parallel query optimization mainly because of its simplicity and the ease of implementation. No changes need to be made to a single-site optimizer, which produces the best plan according to its local cost models; at execution time, only scheduling decisions need to made, which include selecting sites on which to run the operators [CL86] as well as selecting degrees of parallelism for the operators [GI97].

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 the System R algorithm, only one plan is explored in the second phase (which is of more importance in this scenario).

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 6.2.4.

5.5 Randomized/Genetic Algorithms :

Traditionally, randomized or genetic algorithms have been proposed to replace dynamic programming when it is infeasible. The most successful of these algorithms, called 2PO, combines *iterative improvement* (a variant of *hill climbing*) with *simulated annealing* [IK90]. 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.

A different kind of randomized algorithm was proposed in [GLPK94], based on the observation that many of the plans in the plan space have costs very close to that of the optimal plan and hence we should be able to find a reasonably good plan by generating a few plans randomly, costing them and taking the best among them. Though this approach is attractive since it requires only one round of messages, the problem with it is, if it randomly samples N plans, then for small N, the total number of bid requests generated is approximately $O(N * n)$, where n is the number of relations in the query. This very seriously limits the total number of plans evaluated by the optimizer and hence we do not think this approach is feasible in the federated scenario either.

6 Experiments

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.

6.1 Experimental Setup

We have implemented the algorithms described earlier in a modified version of the Cohera federated database system [HSC99]. The experiments were carried on a stand-alone Windows NT machine running on a 166MHz Pentium with 96 MB of Memory. Both the optimizer and the underlying data sources connect to an 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 local area and wide area networks. We used the following configurations for our experiments[10] :

- **LAN :** $\alpha = 10ms,\ \beta = 0.001ms$
- **WAN :** $\alpha = 120ms,\ \beta = 0.005ms$

For the query workload, we use four queries from the TPC-H benchmark. Three of these queries involve 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 below :

- **Query 5 (Q5) :**

```
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]' + interval '1' year
```

[10]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 from these sites.

- **Query 7 (Q7) :**
  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 data '1995-01-01' *and* date '1996-12-31'"
- **Query 8 (Q8) :**
  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 9 (Q9) :**
  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]%'"

6.1.1 Data Distribution and Indexes

For the TPC-H benchmark, we used a scaling factor of 1 which leads to the table sizes as shows in Table 3. The experiments were done using four data sources. The distribution of the tables and the indexes we used were as follows :

- **Site 1 :** supplier (index on suppkey), part (index on partkey), lineitem (index on partkey), nation, region
- **Site 2 :** orders (index on orderkey), lineitem (index on orderkey), nation, region
- **Site 3 :** supplier (index on suppkey), part(index on partkey), partsupp, nation, region
- **Site 4 :** orders (index on orderkey), customer (index on custkey), nation, region

We believe this is a realistic data distribution scheme for this data set since the tables which will be joined more frequently appear together.

Table Name	No. of Tuples	Tuple Size	Table Name	No. 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 3: TPC-H Table Sizes (Scaling Factor = 1)

6.1.2 Cost Model

We use a simple cost model based on I/O that involves only Grace hash join and the 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 :** $cost \ = \ R_b + S_b \ if \ M > min(R_b, \ S_b), \ R_b + S_b + 2(R'_b + S'_b) \ otherwise$

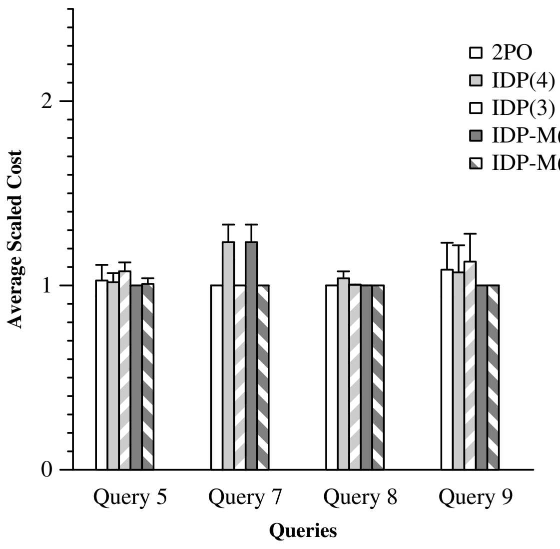
where M denotes the memory available for the join, B_R denotes the number of tuples of R in a block, $|R|$ denotes the number of tuples in the relation R, R_b denotes the number of blocks occupied by the relation R and R'_b denotes the number of blocks that will be occupied by R after performing any 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.

6.2 Optimizer Performance

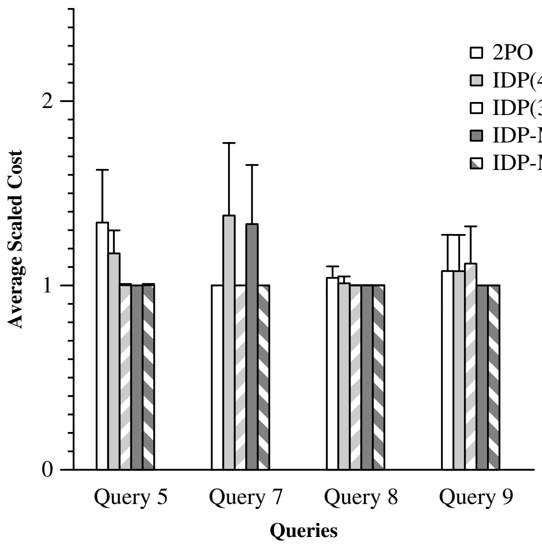
In this section, we will see how the different optimization algorithms perform under various circumstances and further motivate the need for better costing in the optimization process. The algorithms that we compare are Exhaustive (E)(Section 5.1), Two-Phase (2PO)(Section 5.4) and four variants of Iterative Dynamic Programming(Section 5.3), IDP(4), IDP(3), IDP-M(4,5) and IDP-M(3,5).

6.2.1 Effect of uncertain load conditions

Total Cost Optimization



Total Cost Optimization



Response Time Optimization

Figure 3: Optimization in presence of uncertain load conditions (The error bars show standard deviation over 40 runs)

For the first experiment, we varied the cost mark-up on the various data sources involved and also the amount of memory allocated for the joins. The cost mark-ups were chosen between 1 and 20, whereas the memory at each site were chosen between $1M$ to $10M$. 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 was scaled with the optimal plan for the query, found by the exhaustive algorithm. Figure 3 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 we can see, though the two-phase algorithm performs somewhat worse than 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 in Query 9). We will discuss this phenomenon further in the Section 6.2.4.

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 the 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). 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.

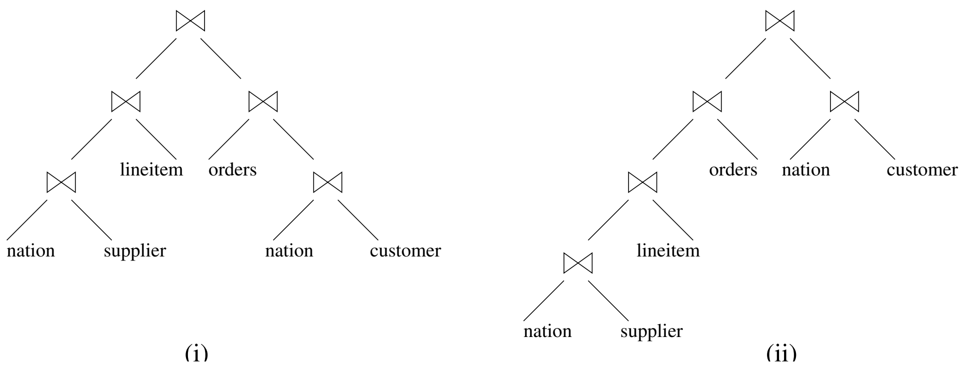


Figure 4: (i) Plan chosen by Exhaustive and IDP(3) for Query 7, (ii) Plan chosen by IDP(4)

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 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

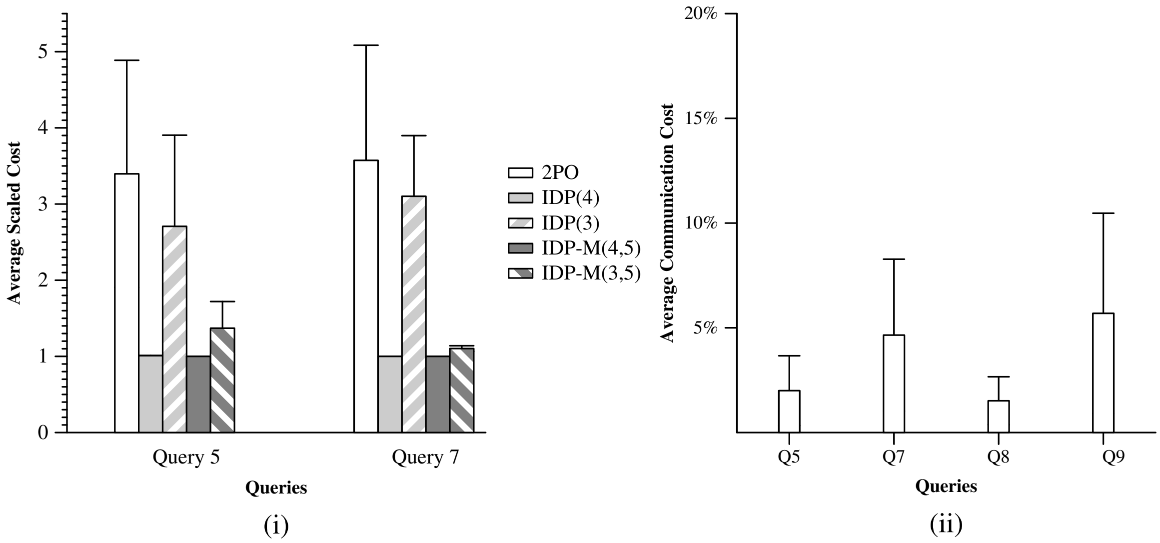


Figure 5: (i) Total cost optimization in presence of materialized views; (ii) Percentage of total cost that is communication cost for WAN (The error bars denote the standard deviation over 40 random runs)

main reason for this is that since we have extended the first 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 similarly as in the earlier experiment, though with higher deviations in some cases.

6.2.2 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 a black box or it may be generating such views dynamically. In this experiment, we will see how such views may affect the performance of different optimization algorithms.

For this experiment, we introduce one view in the system, join of *customer*, *orders* and *lineitem* relations, at Site 2. Since both the 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 data source 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 are kept as in the earlier experiment.

Figure 5 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 wrong plan.

6.2.3 Discussion : 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 our future work on federated query optimization.

6.2.4 Discussion : 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.

Some key observations about our experimental setup that 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 [Sha86].
   These observations leads us to the following assertion :
   Assertion 1 : For a query plan p, if $StaticCost(p)$ denotes the cost of the plan as computed by the first phase and $DynamicCost_1(p)$ 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[11]) and communication costs being zero, then

   $$1/2 \times DynamicCost_1(p) \leq StaticCost(p) \leq 2 \times DynamicCost_1(p)$$

   In other words, the runtime cost of a plan differs by a factor of at most 2 from the static cost of the plan, if communication costs are all zero and the loads on all the sites are equal.

   Proof : Consider any join in the plan. The first phase of the two-phase optimizer might have chosen either the index join method or the hash join method for executing this join. If it had chosen the index join method, then the cost of the join will remain the same under runtime conditions as described above. If it chooses the hash join method, then because of the memory allocated at runtime, it might have to perform two passes instead of one or vice versa. In either case, the cost of the join changes by a factor of at most 2; usually much less since the projections and selections in the query plan result

[11] Without loss of generality, we can set the cost mark-up for one of the sites to be 1.

in very small sizes of the intermediate relations (in comparison to the base tables) and as a result, the second pass is made over a much smaller relation.

3. The communication cost during query execution is not a significant fraction of the total execution cost. Figure 5 shows the average fraction of the total cost that is spend communicating for the plan found by the exhaustive optimizer. As we can see, communication cost forms a very 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 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 $Plan_{static}$ and let the optimal plan under runtime conditions as found by the exhaustive algorithm be $Plan_{dynamic}$. Since the two-phase algorithm finds the best static plan, we have that $StaticCost(Plan_{static}) <= StaticCost(Plan_{dynamic})$.

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

$$DynamicCost_1(Plan_{static}) < 4 \times DynamicCost_1(Plan_{dynamic})$$

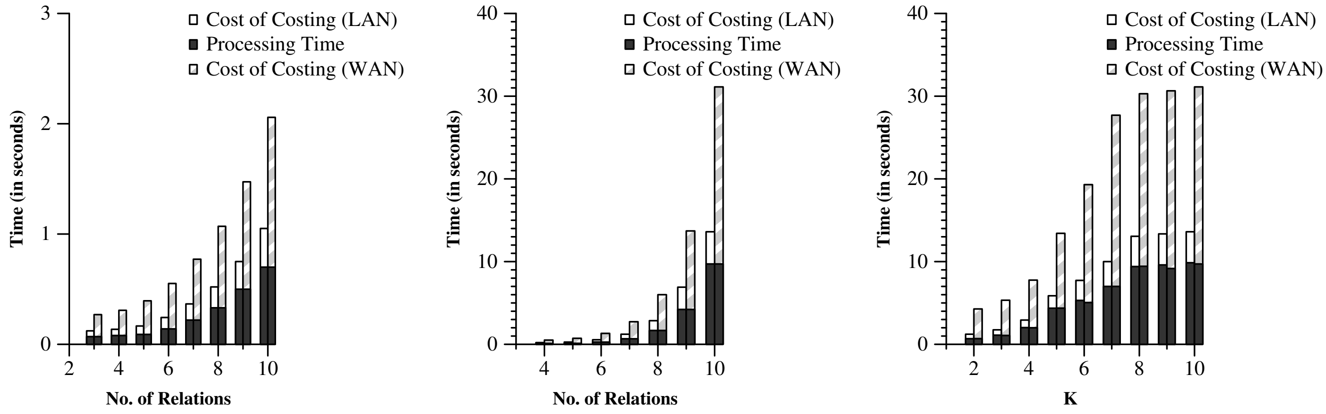
- If we let f be the fraction of total run-time cost of $Plan_{static}$ that is communication cost, then under the worst case assumption that $Plan_{dynamic}$ does not incur any communication cost,

$$(1-f) \times DynamicCost_2(Plan_{static}) < 4 \times DynamicCost_2(Plan_{dynamic})$$
$$\Rightarrow DynamicCost_2(Plan_{static}) < 4/(1-f) \times DynamicCost_2(Plan_{dynamic})$$

  where $DynamicCost_2()$ 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. Figure 5).

- Finally, let us consider the effect of load conditions. 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 be scheduled on lightly loaded sites. It is possible that $Plan_{static}$ 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.

In spite 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 $Plan_{static}$ and $Plan_{dynamic}$ do not differ much in the joins they contain. We believe that this is an artifact of the shape of the query plan topology [IK91]. If any other local minimum in the query plan space has much higher cost that the absolute minimum ($Plan_{static}$), then we expect to see precisely such a behavior.



(i) (ii) (iii)

Figure 6: (i) Optimization time for the exhaustive optimizer for a chain query, (ii) for a star query; (iii) Optimization time for IDP

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.

6.3 Optimization Overheads

In this subsection, we look at the optimization overheads for the various algorithms that we discussed in Section 5, and discuss further trade-offs in the optimization process.

Figure 6 shows the cost of optimization for the exhaustive dynamic programming algorithm as a function of the number of relations in the query, broken up into the message cost and the processing cost. This gives us an idea of how message cost relates to the processing cost in the optimization process. We show the results for two kinds of query graphs, *chain-shaped* and *star-shaped*. Given a particular query graph shape, the actual query has little effect on the number of bids or the CPU cost of optimization. Hence, in these experiments, the queries were artificially constructed by joining a relation with itself as many times as required. As we can see, even after batching all the RFB's in a single message, the cost of costing is still a significant fraction of the total optimization cost even in the LAN configuration.

The optimization overheads for IDP variants depend significantly on the parameter k of the algorithm. Figure 6 shows the optimization time for a star query of size 10 for various values of parameter k. We intentionally chose a large query for IDP, since for smaller queries, it is often more efficient to use exhaustive optimization as shown in Figure 6. We only show the optimization times for the WAN configuration as similar trends were observed for the other configuration. 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

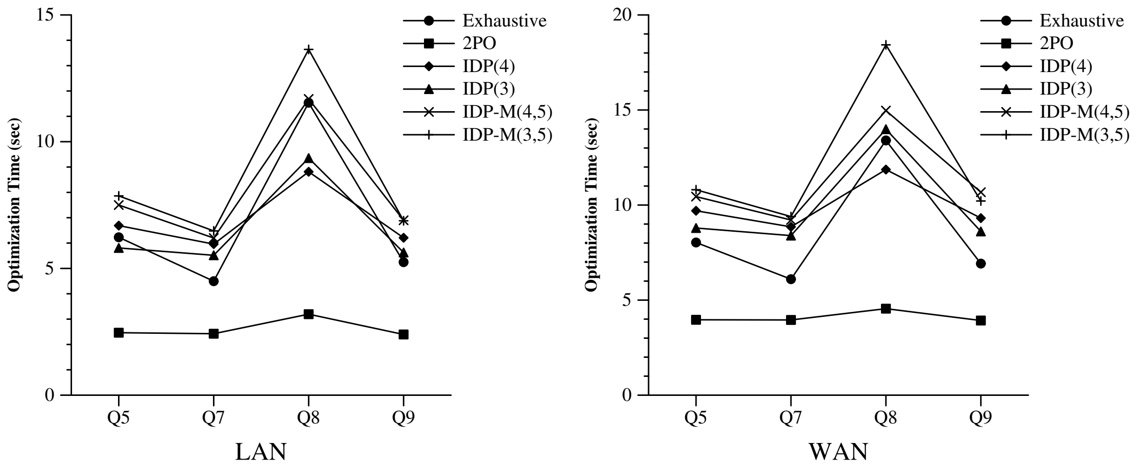


Figure 7: Optimization time for TPC-H Queries (The points are joined only for the purpose of clarity; there is no natural order on the x-axis).

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 the times for subplans that produce results at different sites are not comparable due to the subsequent communication costs. As we can see (Figure 7), 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.

6.4 Summary of the Experimental Results

Our experiments on 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 number of 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.

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 a very promising direction for future work.

7 Conclusion and Future Work

Uncertain load conditions and unknown cost models make decoupled query optimization a necessity for federated database systems. Decoupling of cost computations from the optimization process requires that the optimizer 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.

References

[ACPS96] Sibel Adali, K. Selçuk Candan, Yannis Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, 1996.

[AH00] Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, volume 29, 2000.

[AHL+98] Ron Avnur, Joseph M. Hellerstein, Bruce Lo, Chris Olston, Bhaskaran Raman, Vijayshankar Raman, Tali Roth, and Kirk Wylie. Control: Continuous output and navigation technology with refinement on-line. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data*, 1998.

[BGW+81] Philip A. Bernstein, Nathan Goodman, Eugene Wong, Christopher L. Reeve, and James B. Rothnie Jr. Query processing in a system for distributed databases (sdd-1). *TODS*, 6(4), 1981.

[BVO97] 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.

[CG94] Richard L. Cole and Goetz Graefe. Optimization of dynamic query evaluation plans. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, 1994.

[CHM95] Chandra Chekuri, Waqar Hasan, and Rajeev Motwani. Scheduling problems in parallel query optimization. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1995.

[CL86] Michael J. Carey and Hongjun Lu. Load balancing in a locally distributed database system. In *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*, 1986.

[coh] Cohera net query. http://www.cohera.com/press/presskit/CoheraNetQuery_Data_Sheet.pdf.

[Col00] Richard Cole. A decision theoretic cost model for dynamic plans. *IEEE Database Engineering Bulletin*, 23(2), 2000.

[DKS92] Weimin Du, Ravi Krishnamurthy, and Ming-Chien Shan. Query optimization in a heterogeneous dbms. In *18th International Conference on Very Large Data Bases*, 1992.

[EDNO97] Cem Evrendilek, Asuman Dogac, Sena Nural, and Fatma Ozcan. Multidatabase query optimization. *Distributed and Parallel Databases*, 1997.

[ESW78] Robert S. Epstein, Michael Stonebraker, and Eugene Wong. Distributed query processing in a relational data base system. In *Proceedings of the 1978 ACM SIGMOD International Conference on Management of Data*, 1978.

[FLMS99] Daniela Florescu, Alon Y. Levy, Ioana Manolescu, and Dan Suciu. Query optimization in the presence of limited access patterns. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data*, 1999.

[Gan98] Sumit Ganguly. Design and analysis of parametric query optimization algorithms. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases*, 1998.

[GHK92] Sumit Ganguly, Waqar Hasan, and Ravi Krishnamurthy. Query optimization for parallel execution. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, 1992.

[GI97] Minos N. Garofalakis and Yannis E. Ioannidis. Parallel query scheduling and optimization with time- and space-shared resources. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases*, 1997.

[GK94] Sumit Ganguly and Ravi Krishnamurthy. Parametric distributed query optimization based on load conditions. In *Sixth International Conference on Management of Data (COMAD)*, 1994.

[GLPK94] César A. Galindo-Legaria, Arjan Pellenkoft, and Martin L. Kersten. Fast, randomized join-order selection - why use transformations? In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases*, 1994.

[GM93] Goetz Graefe and William J. McKenna. The volcano optimizer generator: Extensibility and efficient search. In *Proceedings of the Ninth International Conference on Data Engineering*, pages 209–218, 1993.

[Gra95] Goetz Graefe. The cascades framework for query optimization. *Data Engineering Bulletin*, 1995.

[HS91] Wei Hong and Mike Stonebraker. Optimization of parallel query execution plans in xprs. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, December 1991.

[HSB+82] Laura Haas, Patricia Selinger, Elisa Bertino, Dean Daniels, Bruce Lindsay, Guy Lohman, Yoshifumi Masunaga, C. Mohan, Pui Ng, Paul Wilms, and Robert Yost. R*: A research project on distributed relational dbms. *IEEE Database Engineering Bulletin*, 5(4), December 1982.

[HSC99] Joseph M. Hellerstein, Michael Stonebraker, and Rick Caccia. Independent, open enterprise data integration. *IEEE Database Engineering Bulletin*, 22(1), March 1999.

[IK90] Yannis E. Ioannidis and Younkyung Cha Kang. Randomized algorithms for optimizing large join queries. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, 1990.

[IK91] Yannis E. Ioannidis and Younkyung Cha Kang. Left-deep vs. bushy trees: An analysis of strategy spaces and its implications for query optimization. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, 1991.

[INSS92] Yannis E. Ioannidis, Raymond T. Ng, Kyuseok Shim, and Timos K. Sellis. Parametric query optimization. In *18th International Conference on Very Large Data Bases*, 1992.

[Kni99] Leah Knight. "the e-market maker revolution", dataquest inc. http://www.netmarketmakers.com/documents/perspective1.pdf, September 1999.

[KS] Donald Kossmann and Konrad Stocker. Iterative dynamic programming : A new class of query optimization algorithms. see http://www.db.fmi.uni-passau.de/ kossmann/Papers/idp.pdf.

[LVZ93] Rosana S. G. Lanzelotte, Patrick Valduriez, and Mohamed Zaït. On the effectiveness of optimization search strategies for parallel execution spaces. In *19th International Conference on Very Large Data Bases*, 1993.

[net99] Net market makers inc. http://www.netmarketmakers.com/nm101, 1999.

[OL90] Kiyoshi Ono and Guy M. Lohman. Measuring the complexity of join enumeration in query optimization. In *16th International Conference on Very Large Data Bases*, 1990.

[ONK+96] Fatma Ozcan, Sema Nural, Pinar Koksal, Cem Evrendilek, and Asuman Dogac. Dynamic query optimization on a distributed object management platform. In *Proceedings of the fifth international conference on Information and knowledge management*, 1996.

[OW00] Chris Olston and Jennifer Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *26th International Conference on Very Large Data Bases*, 2000.

[PY01] Christos Papadimitriou and Mihalis Yannakakis. Multiobjective query optimization. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2001.

[RAH+96] Mary Tork Roth, Manish Arya, Laura M. Haas, Michael J. Carey, William F. Cody, Ronald Fagin, Peter M. Schwarz, Joachim Thomas II, and Edward L. Wimmers. The garlic project. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, 1996.

[RPK+99] B. Reinwald, H. Pirahesh, G. Krishnamoorthy, G. Lapis, B. Tran, and S. Vora. Heterogeneous Query Processing Through SQL Table Functions. In *15th International Conference on Data Engineering*, March 1999.

[SAC+79] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, 1979.

[SBH98] M. Stonebraker, P. Brown, and M. Herbach. Interoperability, Distributed Applications, and Distributed Databases: The Virtual Table Interface. *IEEE Data Engineering Bulletin*, 21(3), September 1998.

[SBM95] S. Salza, G. Barone, and T. Morzy. Distributed query optimization in loosely coupled multidatabase systems. In *Proceedings of the International Conference on Database Theory*, 1995.

[SDK+94] Michael Stonebraker, Robert Devine, Marcel Kornacker, Witold Litwin, Avi Pfeffer, Adam Sah, and Carl Staelin. An economic paradigm for query processing and data migration in mariposa. In *Proceedings of the Third International Conference on Parallel and Distributed Information Systems (PDIS 94)*, 1994.

[Sha86] Leonard D. Shapiro. Join processing in database systems with large main memories. *TODS*, 11(3), 1986.

[ZL94] Qiang Zhu and Per-Åke Larson. A query sampling method of estimating local cost parameters in a multidatabase system. In *Proceedings of the Tenth International Conference on Data Engineering*, 1994.