Adaptive Cost Estimation for Client-Server based Heterogeneous Database Systems*

Zhaohui Yao Chungmin M. Chen Nick Roussopoulos†
Department of Computer Science
University of Maryland, College Park
e-mail: zhaohui@cs.umd.edu, min@cs.umd.edu, nick@cs.umd.edu

Abstract

In this paper, we propose a new method for estimating query cost in client-server based *heterogeneous database management system.* The cost estimation parameters are adjusted by an *Adaptive Cost Estimation (ACE)* module which uses query execution feedback yielding more and more accurate cost estimates. The most important features of ACE are its detailed cost model which accounts for all costs incurred, its rapid convergence to the actual parameter values, and its low overhead which permits continuous adaptation during the run time of the system. ACE has been implemented and tested with Oracle 6, Oracle 7, Ingres, and ADMS. Extensive experiments performed on these systems show that the ACE's time estimates are within 20% of the real wall-clock time for more than 92% of the queries. This percentage surpasses 98% for queries over 20 seconds.

1 Introduction

Advances in relational database technologies have enabled large organizations and companies to store and manage unprecedented large volumes of data in relational databases. However, due to certain considerations, such as specialized applications requirements, legacy systems, or even strategic decisions, an organization may, instead of conforming to a database management system (DBMS) from a single vendor, elect to adopt DBMSs from different vendors at the same time. To best serve the organization, it is desirable that data stored in heterogeneous DBMS platforms can be retrieved and inter-operated in a convenient and efficient way. A *Heterogeneous Database Management System, (HDBMS)*, coordinates the inter-operation by accessing

*This work was supported in part by the National Science Foundation under grants IRI-9057573 and GDR-85-00108, by NASA/USRA under contracts 5555-09 and NAG 5-2926, by ARPA under contract 003195, by the Institute of Systems Research, and by the University of Maryland Institute for Advanced Computer Studies (UMIACS).

†Also with the University of Maryland Institute for Advanced Computer Studies (UMIACS)

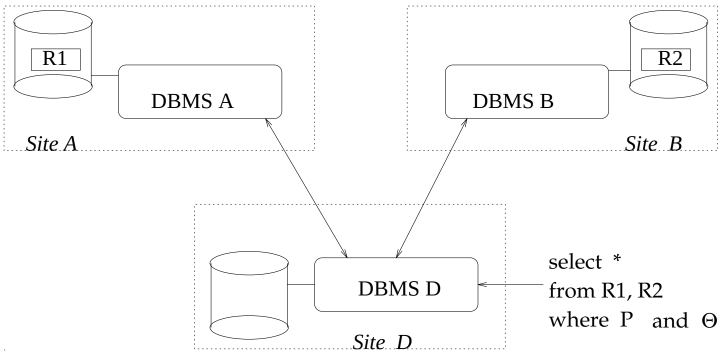
and manipulating data among the DBMSs (referred to as *foreign* DBMS). An HDBMS is usually configured in a distributed client-server architecture where each foreign DBMS is run on a possibly dedicated server machine, taking requests from the HDBMS through its API. An HDBMS treats each of the foreign DBMSs as a "closed-box" component which cannot be modified and which retains its *local autonomy*, i.e. continues to support local applications.

As in traditional distributed DBMSs, query optimization is important in HDBMS [Day85, SY+89, SL90, DKS92, LS92, ZL94, DSD95], particularly for *global queries* which are joins between tables from separate foreign DBMSs. A *global execution plan* for a global query constitutes of a sequence of sub-queries which specifies the join order, table/result shipping direction, and execution sites. Although the optimization techniques used in traditional distributed DBMSs [ML86] can be adapted to HDBMSs [DKS92], they induce some non-trivial problems. One of the problems is *cost estimation* for query plans, which has been a recent research issue in HDBMS [DKS92, ZL94]. Cost estimation is essential in selecting the best plan among various global query plans. The problem is harder in HDBMSs than in traditional distributed DBMSs because foreign DBMSs from different vendors have different access methods, optimization strategies and cost models, all of which may be hidden from the global optimizer of the HDBMS.

This paper presents a practical method for estimating the costs of global query plans for distributed HDBMSs based on experience acquired from previous query executions. The basic idea is to use *query feedback* to adapt a parametric cost function. The parameters of the cost function are gradually adjusted after each query execution, using *query and database dependent* feedback such as table size and predicate selectivities measured during the execution of the query, and *query execution time* measured after the query. Query and database feedback is independent of the performance characteristics of the underlying DBMS, network, and HDBMS client-server implementation, while query execution time is totally determined by them. An *Adaptive Cost Estimation (ACE)* module has been designed and implemented which adapts its parameters by distributing amongst them the estimation error which is the difference between estimated and actual values. The adapted parameters are then used for estimating follow-up queries. ACE is operational in $ADMS\pm$, an Enhanced Client-Server HDBMS prototype developed at the University of Maryland [RK86, RES93, DR94], and obtains accurate cost estimates with small CPU overhead but no I/O. The ACE module works together with another adaptive module of $ADMS\pm$ which estimates the selectivities from exactly the sizes of the returned results [CR94].

1.1 The Problem and Related Work

Consider the distributed query shown in Figure 1 where two global query plans are considered. In the query, P is a selection predicate on R_1, Θ a join predicate between R_1 and R_2. Symbols σ and $\bowtie$ denote selection and join respectively. The query is issued at site D, with the final



Plan 1:

step 1. perform $\sigma_P(R_1)$ at site A, send result to a temporary table T at site B.

step 2. perform $T \bowtie_\Theta R_2$ at site B, send result to site D.

Plan 2:

step 1. send R_2 to site A.

step 2. perform $\sigma(R_1) \bowtie_\Theta R_2$ at site A, send result to site D.

Figure 1: An Example of Distributed and Heterogeneous DBMSs

result to be sent back to the same site. Sites A and B are running different DBMSs that are foreign to the HDBMS at site D. Each of the two plans can be the less costly one depending on the size of the tables, the selectivity of the selection, and in particular, the efficiency of the access methods supported by the underlying foreign DBMSs. For example, if the result size of $\sigma_P(R_1)$ is much smaller than the size of R_2, then Plan 1 might be favored because it incurs less communication cost. However, if DBMS A has much faster access methods on the selection and join attributes of $R1$ which can be used to perform $\sigma(R_1) \bowtie_\Theta R_2$ than DBMS B can perform $T \bowtie_\Theta R_2$, then Plan 2 might be favored. In a homogeneous distributed DBMS, the costs of performing $\sigma(R_1) \bowtie_\Theta R_2$ (at site A) and $T \bowtie_\Theta R_2$ (at site B) are easier to estimate because all sites (A, B, D)are running the same DBMS. In a HDBMS environment, it is rather difficult because commercial DBMSs have no mechanism for exposing internal optimization parameters and statistics to be used by the global optimizer at the site where the query is issued (site D).

The importance of developing an appropriate cost estimation mechanism for query optimization in HDBMSs was first pointed out in [SY$^+$89]. Two methods, *database calibration* [DKS92] and *query sampling* [ZL94], were later proposed based on the philosophy that individual foreign DBMSs are viewed as "black-boxes" of which cost models can be "deduced" based on the cost information revealed by the execution of queries (namely, the actual query execution times).

In the calibration method, synthetic databases are created and imported into foreign DBMSs, then a set of queries are issued so as to deduce the parameters of the cost models. This method requires the overhead of creating and loading the "special purpose" synthetic database into each member foreign DBMS.

In the query sampling method, sample queries are performed against existing databases in the foreign DBMSs. The query execution times are then measured and used to determine the cost parameters using a multiple regression technique. Using existing databases instead of creating synthetic databases relieves the burden and overhead of defining and populating the synthetic database. Nevertheless, the method still incurs the overhead of initially running the sample queries and periodically repeating them to reflect the database evolution through updates.

1.2 Structure of the Paper

The rest of the paper is organized as follows: Section 2 describes a representation model for global query plans, a classification scheme for foreign queries, and the adaptive cost estimation technique. Section 3 describes the implementation of ACE in $ADMS\pm$. Section 4 analyzes the performance results of ACE based on the implementation of $ADMS\pm$. Conclusions and future work are discussed in Section 5.

2 Adaptive Cost Estimation for Distributed HDBMS

We first present a model for global query plans which is used to decompose each global query to a collection of *local subqueries* each of which is executable on a single site. Based on this model, the total cost of a global query plan can be easily obtained by summing up the composing subqueries. A scheme is then used to classify the subqueries into different classes according to their performance characteristics. We then propose a generic parametric cost function for all classes of subqueries and describe the technique for adapting the parametric cost function.

2.1 Global Query Plan and Classification

We decompose a general query to subqueries that can be computed locally on a single server site with a *selection-projection (sp)* or *selection-projection-join (spj)* operation. The rationale in this decomposition is that these subqueries have predictable cost behavior mostly because they are performed in one pass.

Definition 1 *A* Local-Compute-and-Forward (LCF) *(sub)query, denoted by*

$$\pi_\alpha(\sigma_P(R))_{A\rightarrow B} \quad \text{or} \quad \pi_\alpha(\sigma_P(R_i \bowtie R_j))_{A\rightarrow B},$$

Class	Query Type	Index on Selection/Join Attributes
QC_1	$\pi_\alpha(\sigma_P(R))_{A\rightarrow B}$	*sp* with a clustered index
QC_2	$\pi_\alpha(\sigma_P(R))_{A\rightarrow B}$	*sp* with a non-clustered index
QC_3	$\pi_\alpha(\sigma_P(R))_{A\rightarrow B}$	*sp* with no index
QC_4	$\pi_\alpha(\sigma_P(R_i \bowtie R_j))_{A\rightarrow B}$	*spj* with a clustered index
QC_5	$\pi_\alpha(\sigma_P(R_i \bowtie R_j))_{A\rightarrow B}$	*spj* with a non-clustered index
QC_6	$\pi_\alpha(\sigma_P(R_i \bowtie R_j))_{A\rightarrow B}$	*spj* with no index

Table 1: Query Classes

is a selection-projection (sp) or selection-projection-join (spj) query that is to be executed at a single site A of a DBMS with the result being exported from site A and imported to site B.

Note that for a *spj* LCF query, the two operand tables R_i and R_j must both reside in site A's DBMS. When A and B refer to the same site no result export/import is needed. Otherwise, the result is to be generated from A's DBMS, transmitted over the network, and imported into the DBMS of site B.

Every global query plan can be represented by a sequence of LCF-queries. For example, consider a global query $\pi_\alpha(\sigma_P(R_1) \bowtie R_2 \bowtie R_3)$ where R_1, R_2, R_3 reside in sites A, B, C respectively and each of which running a different DBMS, and the final result is to be imported into site D. Let α_{R_i} be the union of the projection of α on R_i and the selection/join attribute(s) of R_i. The following are two possible global query plans:

$$\pi_\alpha((\sigma_P(R_1)_{A\rightarrow B} \bowtie R_2)_{B\rightarrow C} \bowtie R_3)_{B\rightarrow D}$$

$$((\pi_{\alpha_{R_1}}(\sigma_P(R_1)) \bowtie \pi_{\alpha_{R_2}}(R_2)_{B\rightarrow A}) \bowtie \pi_{\alpha_{R_3}}(R_3)_{C\rightarrow A})_{A\rightarrow D}$$

The total cost of a global query plan can be obtained by summing up the costs of the composing LCF-queries. In this paper, we shall concentrate on estimating the costs for LCF-queries as they constitute the basics of a global query plan.

In Definition 1, we define two classes of LCF-queries, *sp* and *spj*. However, since each class represents a broad range of queries whose best access methods may greatly vary, we adopt the classification of [ZL94] for LCF-queries which is based on whether a clustered, or unclustered index is present in the selection/join attributes or no index. Table 1 shows the six LCF query classes $QC_1 \sim QC_6$ obtained by this classification. We associate a distinct cost function to each of these classes. The rationale behind such a classification is that queries in the same class have the same set of available access methods and are, therefore, most likely to be handled by the foreign optimizer in the same way.

Cost Factor	Meaning
f_1	constant initialization overhead cost of 1 unit
f_2	number of messages required to execute an LCF
f_3	cardinality of the first operand table
f_4	average tuple length of the first operand table in bytes
f_5	cardinality of the second operand table
f_6	average tuple length of the second operand table in bytes
f_7	cardinality of query result
f_8	average tuple length of query result in bytes
f_9	total size of query result in bytes

Table 2: Notations for Cost Factors

2.2 Cost Model

In traditional distributed database systems where all sites are running the same DBMS, the following formula is typically used in estimating the cost of a distributed query plan (with known local access methods) [ML86, OV91]:

$$\begin{aligned} Total\ cost \ = \ & W_{CPU} * (number\_of\_instructions) + W_{I/O} * (number\_of\_I/Os) + \\ & W_{MSG} * (number\_of\_messages) + W_{BYTE} * (number\_of\_bytes) \end{aligned} \qquad (1)$$

where $W_{CPU}, W_{I/O}, W_{MSG}$, and W_{BYTE}, are system-wide constants that denote the *weighted* (relative) cost per instruction execution, per I/O operation, per message transmitted and per byte of data transmitted over the network, respectively. Usually, these weights are empirical values obtained by running a large set of sample queries and are hard-coded into the DBMS kernel. Details about the local access methods of the query plan must be known a priori in order to estimate the parameters including number of instructions executed, number of I/Os performed, number of messages and total bytes of data transferred over the network. These parameters depend not only on the query characteristic and data profile (including table sizes and query selectivities), but also on the DBMS kernel's characteristics (including the size of the code that implements each access method, the buffer manager's strategies, and the network interface parameters). These parameters can only be available in proprietary solutions of homogeneous systems, and therefore are refereed to as *system-dependent* parameters.

The above formula, however, is of no use for the HDBMS case because the global query optimizer has no access to the system-dependent parameters and/or knowledge on how the optimizer of each foreign DBMS will perform. Since system-dependent parameters are unavailable, we must use a cost model that is solely based on *query/data-dependent* parameters such as query expression, data statistics, and estimated sizes of results. Like [S$^+$79, K$^+$85, ZL94], we assume that all four parameters of formula 1 are in proportion to a few basic quantitative query/data dependent parameters, called *cost factors*. ACE uses nine cost factors $f_1 \sim f_9$

whose meaning is shown in Table 2. For a *sp* query, the factors f_5 and f_6 are omitted and their values are zeros. The CPU and I/O costs on the server are captured in the $f_3 \sim f_7$ factors while the rest of them model the communication network and the client-server inter-operation cost. Exact or pretty accurate estimates of the above factor values can be obtained by the query feedback as these factors do not depend on the internals of the foreign DBMSs, and can readily be obtained.

For each query class $QC_j (1 \leq j \leq 6)$, ACE maintains and uses a cost estimation function:

$$\hat{c}(q) = \sum_i a_{i,j} \cdot f_i(q) \tag{2}$$

where $\hat{c}(q)$, the estimated cost of a LCF query $q \in QC_j$, is a linear combination of cost factors $f_i(q)$, with $a_{i,j}$ being the *cost coefficients* (of query class QC_j) that map the cost factors to the estimated cost. Note that unlike formula 1 where the CPU, I/O, and network costs are considered separately, the ACE cost formulae model the cumulative cost of all these costs regardless of the idiosyncrasies of the underlying DBMS and network.

Consider a *sp* query q of class QC_1 where the clustered index is maintained as a B-tree. The cost of shipping the query to the foreign DBMS will be subsumed by $a_{1,1}f_1(q)$ and $a_{2,1}f_2(q)$. The cost of navigating the B-tree, which includes the initialization overhead (a constant) and the number of B-tree nodes retrieved (depending on the height of the tree and the selectivity of the predicate), will be subsumed by $a_{1,1}f_1(q)$, $a_{3,1}f_3(q)$ and $a_{7,1}f_7(q)$. Similarly, the cost of retrieving and processing the qualified tuples from the relation will be subsumed by $a_{4,1}f_4(q)$, $a_{7,1}f_7(q)$, and $a_{8,1}f_8(q)$; the cost of transmitting the result over the network will be subsumed by $a_{9,1}f_9(q)$. Similarly, if a linear scan, rather than the B-tree, is chosen as the access method, the cost of the linear scan can still be properly subsumed by different products in the cost function. The purpose of the cost coefficients $a_{i,j}$ is to map a query to the cost of the access method that is most likely to be chosen, based on the characteristics of the query (which are quantified by the cost factors). The values of $a_{i,j}$ determine the accuracy of the cost estimation.

2.3 Adaptive Cost Estimation

Database calibration [DKS92] and query sampling [ZL94] techniques have been proposed to estimate the unknown parameters of a foreign cost function. As discussed in section 1.1, both methods require a non-trivial periodic tuning and incur substantial overhead. We propose a much simpler, dynamic, and cost-effective approach for estimating the cost coefficients, which uses neither synthetic tables nor sample queries. In our method, the coefficients adjust gradually after each query execution by using actual cost measurement. The coefficients are guaranteed to converge to the "optimal values" (which will be defined later) as queries proceed, thus yielding more and more accurate cost estimates.

Consider the cost function $\hat{c}(q) = \sum_{i=1}^{n} a_i \cdot f_i(q)$ that is associated with a query class QC of a foreign DBMS A. Let $q_1, q_2, \ldots, q_k$ ($k \geq n$) be a sequence of queries from class QC and $c_1, c_2, \ldots, c_k$ be the corresponding actual query execution times measured. The cost estimation error for q_i can be expressed as $\hat{c}(q_i) - c_i$. A reasonable way to assign values to a_i's is to find the values that will minimize the squared sum of the cost estimation errors:

$$\sum_{i=1}^{k}(\hat{c}(q_i) - c_i)^2 = \sum_{i=1}^{k}(\sum_{j=1}^{n} a_j f_j(q_i) - c_i)^2 \quad (3)$$

$$= \sum_{i=1}^{k}(X_i \cdot A - c_i)^2 \quad (4)$$

where $X_i = [f_1(q_i), f_2(q_i), \ldots, f_n(q_i)]$ and $A = [a_1, a_2, \ldots, a_n]^T$ (x^T denotes the transpose of x). This problem is known as the Least Square Problem and its solution $A_k^* = [a_1^{(k)}, a_2^{(k)}, \ldots, a_n^{(k)}]^T$ which minimizes expression 4 can be computed as [Mar87]:

$$A_k^* = (X^T X)^{-1}(X^T C) \quad (5)$$

where X is a $k \times n$ matrix with X_i being the i'th row, and $C = [c_1, c_2, \ldots, c_k]^T$. This "stage-wise" method requires recomputation every time a new estimate error is to be included. For example, if an additional query q_{k+1} and its observed cost c_{k+1} are to be included, then A_{k+1}^* needs to be computed form scratch, because it takes no advantage of previously computed values of A_k^*. A better method, called *recursive least-square estimation*, [Lee64, You84], eliminates the duplication by using a recursive expression. It expresses the solution A_k^*, when $k > n$, in a recursive form:

$$a_l^{(k)} = a_l^{(k-1)} - [\sum_{i=1}^{n} g_{l,i}^{k} \cdot f_i(q_k)][\sum_{i=1}^{n} a_i^{(k-1)} \cdot f_i(q_k) - c_k], \quad \text{for } 1 \leq l \leq n \quad (6)$$

$$g_{l,m}^{(k)} = g_{l,m}^{(k-1)} - \alpha[\sum_{i=1}^{n} g_{l,i}^{(k-1)} \cdot f_i(q_k)][\sum_{i=1}^{n} f_i(q_k) g_{i,m}^{(k-1)}], \quad \text{for } 1 \leq l, m \leq n \quad (7)$$

where $\alpha = 1/(X_k \cdot G \cdot X_k^T)$, G is an $n \times n$ matrix with $G(i,j)_{1 \leq i,j \leq n} = g_{i,j}^{(k-1)}$.

In the above recursion, the coefficient a_i's are adjusted after each query by subtracting a corrected term which is in proportion to the estimation error. The $g_{i,j}$'s can be viewed as a matrix of self-organized memory cells that take subsequent estimation errors as feedback to support the adaptation of a_i's. Note that the above recursive relation does not hold for $k \leq n$ because there does not exist a unique least square solution for a system of k equations which has more than k variables. However, if we start the recursion by assigning $a_i^{(0)} = 0$, $g_{i,j}^{(0)} = 0$ for $i \neq j$, and $g_{i,i}^{(0)}$ to some large number, then the solution computed using formulae 6 and 7 eventually converges to the one computed using formulae 5 as k increases [You84].

The adaptation mechanism starts as soon as the first query is executed. During the first few queries, the cost estimation errors may be relatively large because the cost function is still in its learning stage. Our experiments show that, however, after a few queries, the estimated costs are very close to the actual costs. The advantages of ACE over previous methods include:

- *Efficiency.* The coefficients of the cost function are determined and dynamically modified on-line without I/O access and with negligible CPU overhead.
- *Robustness.* It is practically difficult to isolate the costs of a query plan that are contributed by the underlying operating system and/or network softwares, because they are heterogeneous and proprietary systems. In our method, however, these modules are treated as black-boxes and their contributing costs are subsumed in the cost function implicitly by using a linear combination of query-dependent only parameters with adaptive coefficients.
- *Practicality.* Based only on a generic cost function with adaptable parameters and a simple recursion scheme, ACE is easy to customize for and incorporated into different database architectures/environments. We have implemented ACE in a client-server based HDBMS prototype with very little coding overhead.

3 Implementation of ACE

We implemented ACE inside $ADMS\pm$, an enhanced multi-site client/server (E-CS) HDBMS, in which the clients are fully-fledged DBMSs capable of caching and maintaining downloaded data subsets obtained as a result of running global queries on multiple DBMSs [RK86, RES93, DR94]. The database servers are commercial and other prototype DBMSs accessed through application level gateway software, called $ADMS+$, which capitalizes on incremental access methods [Rou91] for downloading and maintaining cached data results in the form of *materialized views*. The communication between servers and clients is based on TCP/IP Networking Protocol over LAN/WAN. Figure 2 shows the system architecture of $ADMS\pm$ with three commercial DBMSs and our own ADMS prototype. Each client runs a single-user version of ADMS, called $ADMS-$, which maintains on its own local storage materialized views, cached catalogs, and statistics.

ACE is built into the *Global-Query Optimizer* of $ADMS\pm$ to estimate the costs of different classes of LCF queries. The *Global-Query Optimizer* parses a global HDBMS query into a sequence of LCF-subqueries, obtains statistic information about the operand tables(cardinality, tuple length, indexes etc.) from the locally stored system catalogs, maps each LCF-subqueries into its corresponding query class based on the classification criteria defined in Table 1, then invokes the ACE module to produce the cost estimation for each LCF-subquery. The total cost of a global query plan can be obtained by summing up the costs of the composing LCF-queries. The Global-Query Optimizer then prunes off costly query plans and generates an execution plan with the minimum cost estimate.

One of the key factors related to the cost estimation is the *predicate selectivity*, which is the number of tuples satisfying a given predicate. The accuracy of selectivity estimation

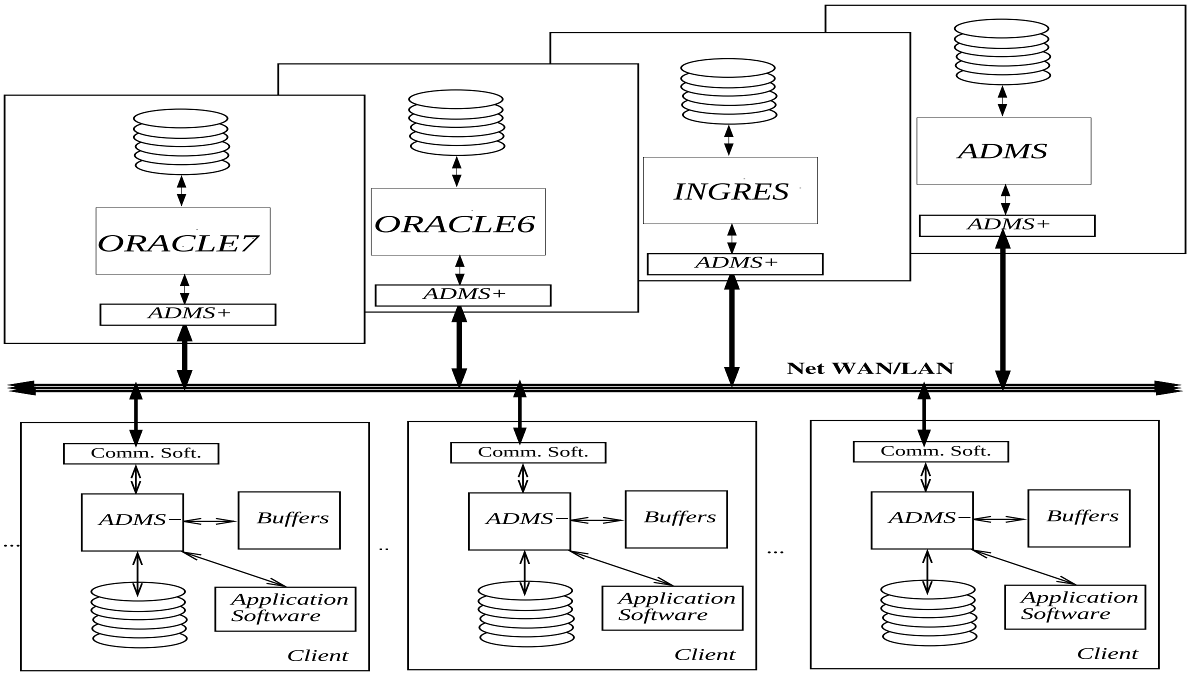


Figure 2: $ADMS\pm$ System Architecture

directly affects the accuracy of the query cost estimation. ADMS uses another adaptive module, called *Adaptive Selectivity Estimator (ASE)* [CR94], for interpolating the value distributions of attributes which are then used to estimate selectivities. ASE produces accurate estimates of record selectivities from real attribute value distributions which are adaptively approximated by a curve-fitting polynomial using the query feedback mechanism. Its accuracy and performance have been reported in the above paper.

Both ACE and ASE are modules of the global-query optimizer of $ADMS\pm$ using query feedback to adapt. ASE is invoked by ACE when a LCF-subquery with a selection predicate is generated by the global-query optimizer and its selectivity needs to be estimated. In $ADMS\pm$, the query feedback consists of (a) the actual selectivity obtained after running the query, (b) the actual real time cost of the query execution, (c) catalog statistics from the server(s). ASE uses (a) and (c) while ACE uses (b) and (c). Catalog statistics basically include table cardinalities and indexing information. These are piggy-backed with the query result from the server(s) and used to update the locally cached client catalogs.

ACE and ASE require some matrix manipulation and mathematical computation but only incur CPU cost. In our $ADMS\pm$ implementation of ACE and ASE, the overhead of the ACE and ASE module computation is only a small fraction of the optimization cost and negligible when compared to the real query execution cost.

As mentioned above, ACE uses real wall-clock time observed on the client to adapt. For each query, a start timestamp is obtained by the client just before it begins to transmit the query to the server(s) and an end timestamp is recorded after the last record of the result has been received. The elapsed time between the timestamps is our metric of cost and measures all other costs, inter-operation, server CPU, server I/O, communication, and server and network contention factors.

4 Experimental Results

We performed extensive experiments to estimate LCF query cost in $ADMS\pm$. The configuration of these experiments included three commercial DBMS servers Oracle (v7.0), Oracle (v6.0) and Ingres (v6.0), our own prototype ADMS (v3.3) server, and the $ADMS\pm$ (v2.0) Enhanced Client-Server HDBMS. Oracle 7 runs on a SparcStation 20, Oracle 6 on a SparcStation 2, Ingres on a DECstation 5000/200, and ADMS on a SparcStation 2. The clients were run on separate SparcStation 2s. All client and server machines are connected via a shared Ethernet network. All the experiments were conducted during the night under low network/server loads.

We used the Wisconsin Benchmark relations [BT94] in all our experiments. The eight tables used along with their statistics are shown in Table 3. These are pre-loaded into each of the server DBMSs before the experiments are run. A range-varying parametric query for each of the six LCF classes of queries was used to generate randomly distributed range queries. These are shown in Table 4 in their $ADMS\pm$ extended SQL syntax with C_1 and C_2 being the variable range parameters and, $REL1$ and $REL2$ being relation variables from the database. The result is to be downloaded to and stored in a materialized view on the client.

For each LCF class, one hundred different ranges were randomly generated but with controlled selectivity to generate results of varying sizes from 10 to 10000 records. Each of these groups of one hundred queries was regenerated for four different sets of varying size relations to obtain 400 queries for each class or 2400 queries for the whole experiment. These 2400 queries were randomly mixed to generate the final query stream used in all server runs of our experiment. The random ranges, the variations of relations and the random mix were employed to reduce side-effects of shared buffers by similar queries.

The query stream was run from *cold start* and a single log for each server was generated. From these logs we make our observations and draw conclusions. We compare the ACE estimated cost with the actual real-time costs and generate histograms and graphs showing the accuracy of the estimates, the relative errors, and the adaptive capability of ACE.

Figure 3 $\sim$ Figure 6 show the statistical and confidence analysis of ACE's runs on all four servers. The histograms show the percentage of queries for each 10% intervals of relative error. On Oracle7, 92% of the queries had relative error between 0 and 10%. The corresponding figure for Oracle6 is 90%, for Ingres 78% and for ADMS[1] 70%. The percentage of queries for

[1] ADMS is more susceptible to Unix mannerisms and its eager prefetching which are more difficult to estimate. Another reason for the lower figure for ADMS is that query execution times are much shorter than all the other

Relation	Tuple_Length	Cardinality	Clustered_Index	Non_Clustered_Index
onek1	182	1000	Y	Y
onek2	182	1000	N	N
twok1	182	2000	Y	Y
twok2	182	2000	N	N
fivek1	182	5000	Y	Y
fivek2	182	5000	N	N
tenk1	182	10000	Y	Y
tenk2	182	10000	N	N

Table 3: Experiment Relations

Query Type	$ADMS\pm$ Extended SQL Format
sp with a cluster-indexed attribute	select $a_1, \ldots, a_n$ from $DB.REL1$ where $un2 > C_1$ and $un2 < C_2$ into $VIEW1$;
sp with a non-cluster-indexed attribute	select $a_1, \ldots, a_n$ from $DB.REL1$ where $un1 > C_1$ and $un1 < C_2$ into $VIEW2$;
sp with no indexed attribute	select $a_1, \ldots, a_n$ from $DB.REL1$ where $k1 > C_1$ and $k1 < C_2$ into $VIEW3$;
spj with a cluster-indexed attribute	select $a_1, \ldots, a_m, b_1, \ldots, b_n$ from $DB.REL1$, $DB.REL2$ where $DB.REL1.un2 = DB.REL2.un2$ and $DB.REL1.un2 > C_1$ and $DB.REL1.un2 < C_2$ into $VIEW4$;
spj with a non-cluster-indexed attribute	select $a_1, \ldots, a_m, b_1, \ldots, b_n$ from $DB.REL1$, $DB.REL2$ where $DB.REL1.un1 = DB.REL2.un2$ and $DB.REL1.un1 > C_1$ and $DB.REL1.un1 < C_2$ into $VIEW5$;
spj with no indexed attribute	select $a_1, \ldots, a_m, b_1, \ldots, b_n$ from $DB.REL1$, $DB.REL2$ where $DB.REL1.k1 = DB.REL2.un2$ and $DB.REL1.k1 > C_1$ and $DB.REL1.k1 < C_2$ into $VIEW6$;

Table 4: Experiment Queries

which ACE had relative error of less than 20% range from an impressive 97% for Oracle7 down to 92% for Ingres.

The right hand side of Figure 3 $\sim$ Figure 6 illustrate the confidence analysis on the mean relative error for each 10-second query time interval ranging from 0 to the maximum query time on each server. The *confidence coefficient* was set to 95%. These graphs show the mean relative error and its standard deviation contained below the 20% value.

From the experiment logs, we found that more than 98.5% of the of long queries, i.e. with real-time $\geq$ 20 seconds, have relative error less than 20% for all four servers with ADMS being the winner this time with 99.2% of its long queries. This is important because HDBMS queries

server DBMS and thus the standard deviation of the relative error is much more sensitive.

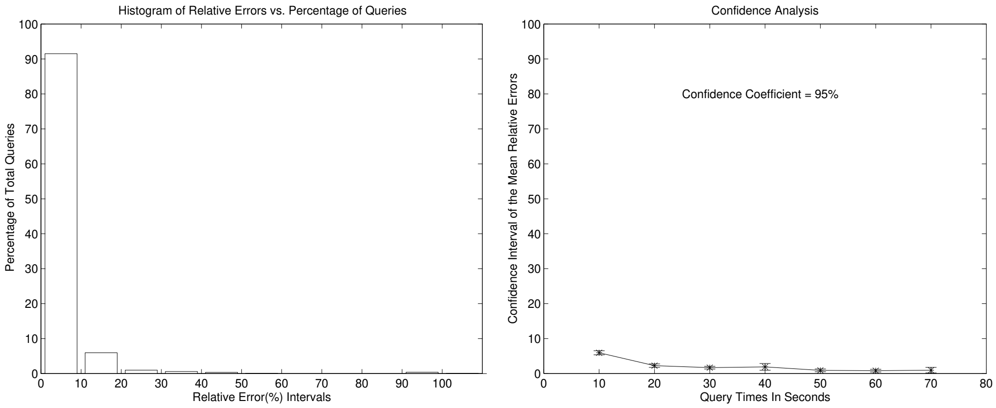


Figure 3: Complete Mixed Queries on Oracle7 Server

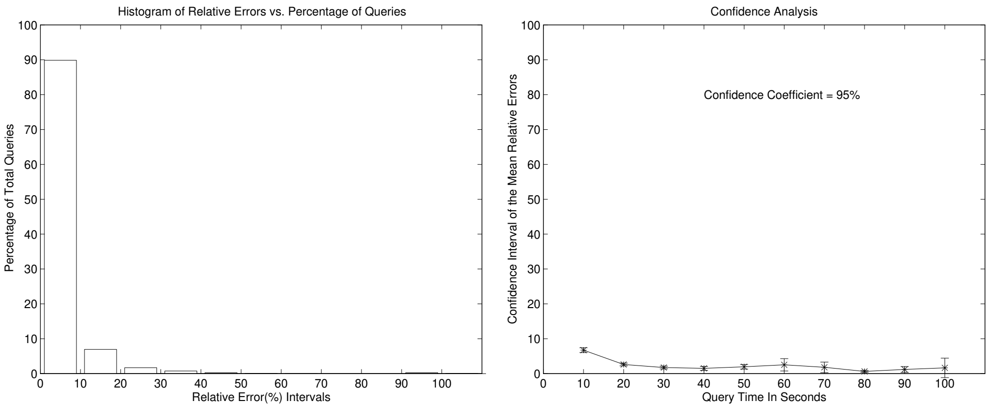


Figure 4: Complete Mixed Queries on Oracle6 Server

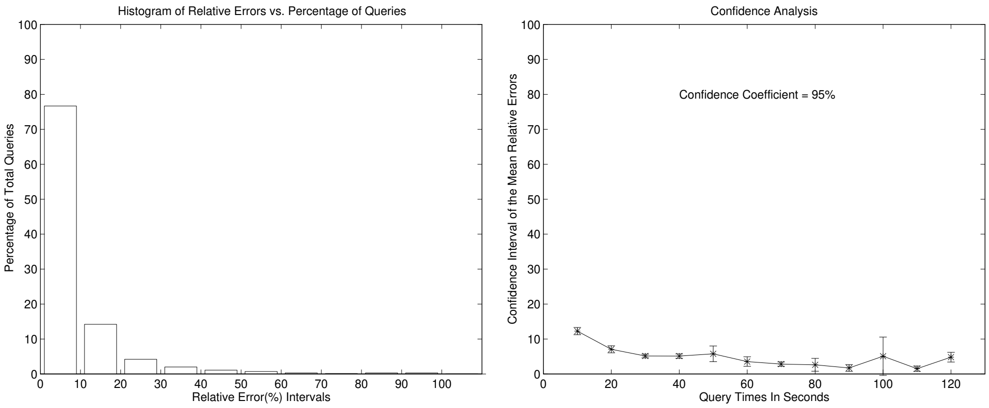


Figure 5: Complete Mixed Queries on Ingres Server

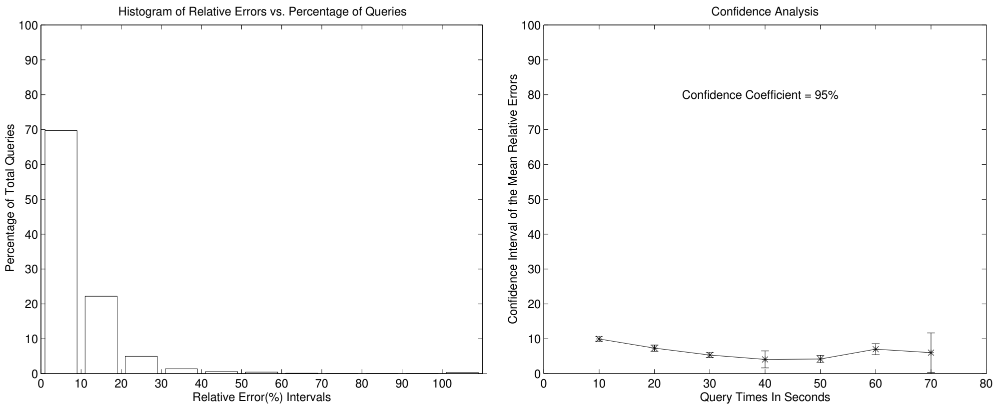


Figure 6: Complete Mixed Queries on ADMS Server

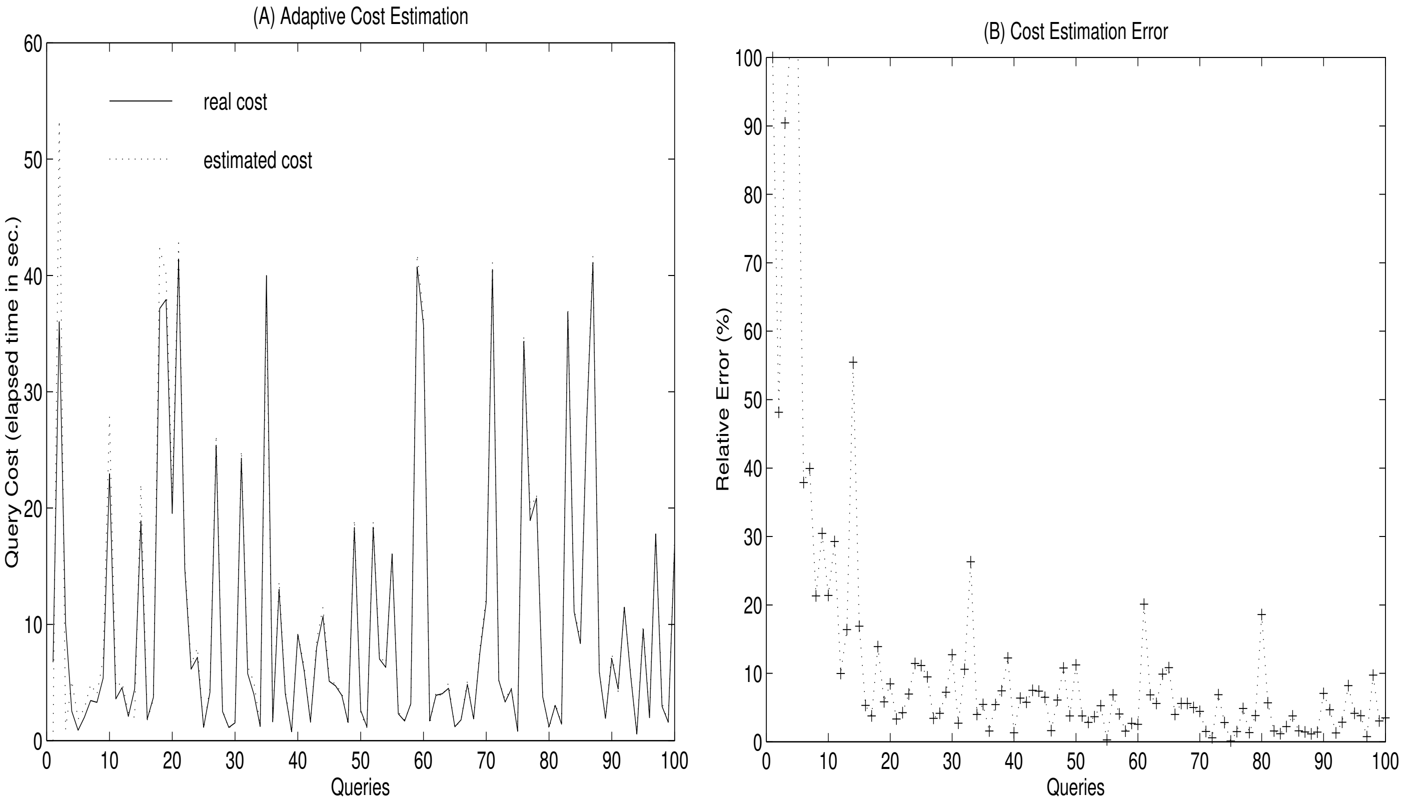


Figure 7: sp with a non-clustered index on Oracle7(Query Class:QC_2)

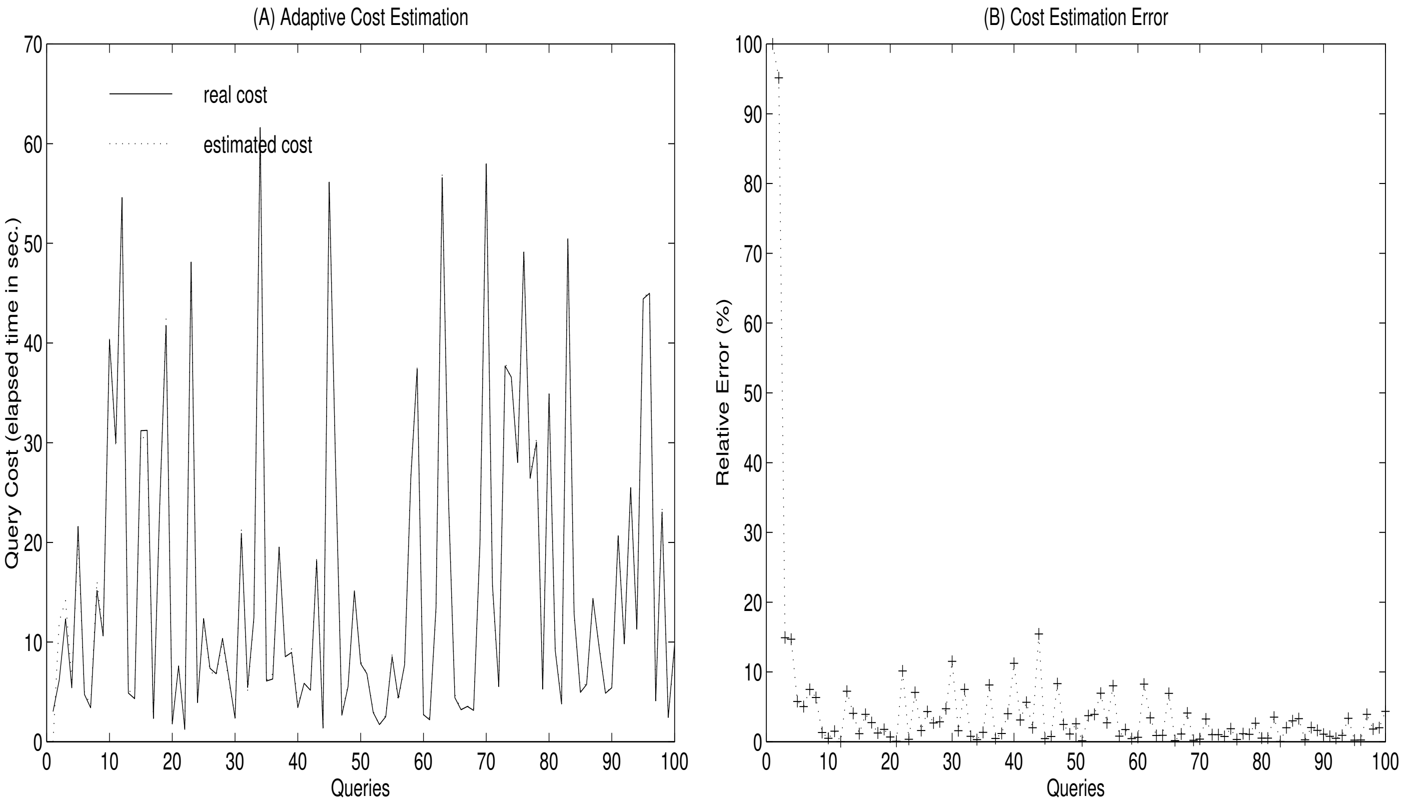


Figure 8: spj with a clustered index on Oracle7(Query Class:QC_4)

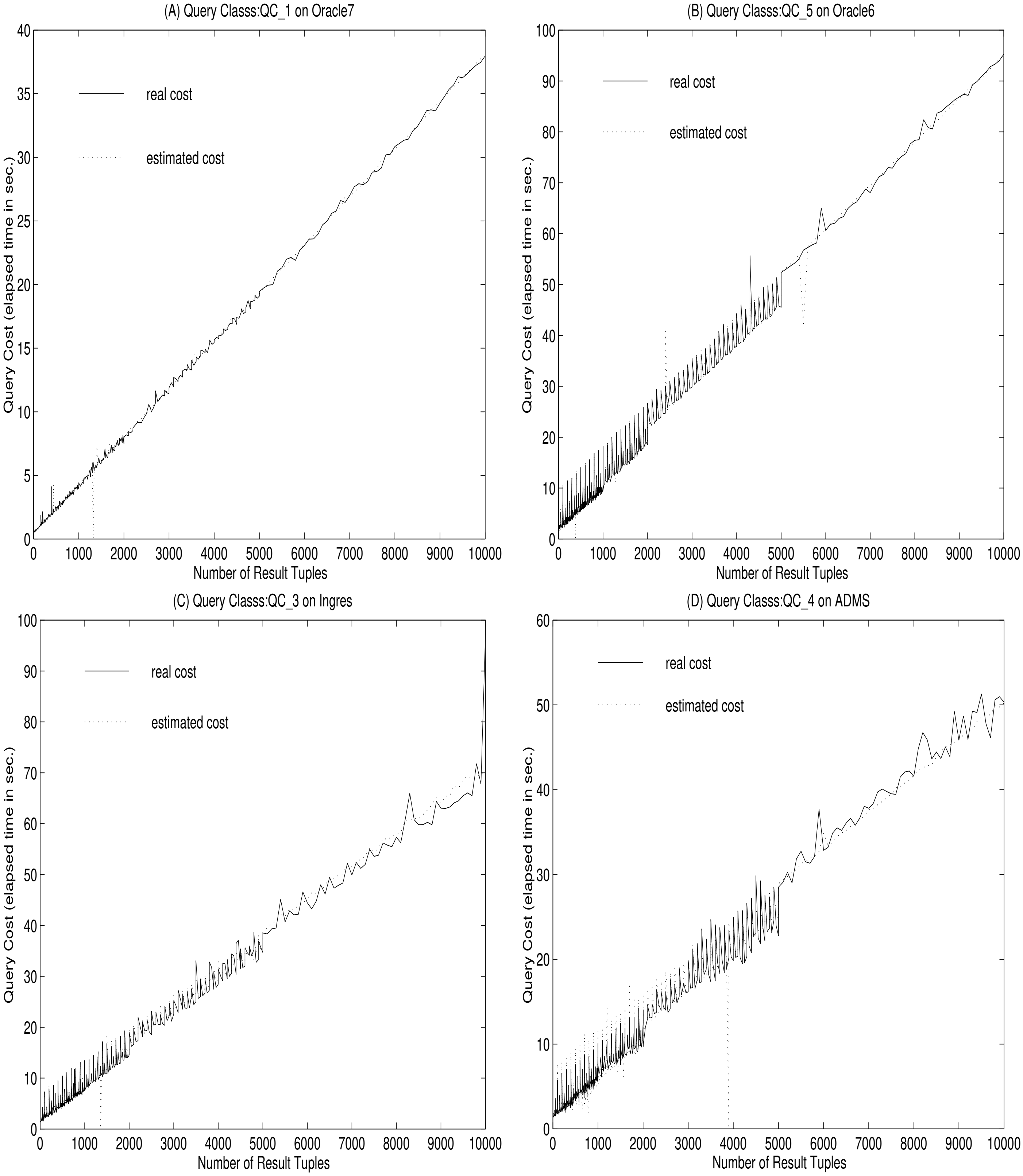


Figure 9: Adaptive Cost Estimation for different query classes in order of query result size

are by nature more time-consuming than local ones. In our experiments, the larger relative errors are contributed by the short queries in which the error is perhaps more tolerable. A 20% relative error on a 100 second query is itself intolerable but more acceptable for $1 \sim 5$ second queries.

We now show how ACE behaves on some of the LCF classes. It should be noted that the graphs for each of the classes were extracted from the same log of the 2400 queries but illustrate ACE's performance for each individual class. The left hand side of Figure 7 and Figure 8 depict ACE's estimated time and the real-time of each query. The queries shown on the x-axis are in the very same chronological order as they occur in the 2400 query stream and, therefore, have wide time variation. The graphs on the right hand side of these figures illustrate the relative errors corresponding to the same chronological order of the queries on the left hand side. They are included to show the adaptive capability of ACE for the corresponding classes. Note that ACE was started from "cold" in all experiments and that it only takes a few queries to reduce the relative error below 20%.

Finally, in Figure 9 we present yet another (and customary) view of some of the LCF-query classes but now with the x-axis showing the queries sorted on the number of records they generate. Since queries on all size relations are plotted and since query results of 10 to 1000 are generated from four pairs of relations, there are considerably more points in that range. Again, these figures show that ACE makes very accurate estimates of the real-time cost. Note also in these figures the spikes to zero of the estimated cost correspond to the cold start of the very first query of the class.

5 Conclusion and Future Work

In this paper, we proposed a new technique for estimating the query cost in a heterogeneous DBMS environment. Accurate cost estimation is very important in such systems because errors can have a huge impact on actual execution cost. We formulated and provided a solution to the problem by a detailed cost model which explicitly accounts for server CPU, server I/O, and network overhead, but implicitly captures system dependent factors such as hardware configuration, operating system and underlying DBMS.

The cost formula coefficients are adaptively adjusted by an adaptive cost estimation module using query feedback and statistics obtained during execution yielding more and more accurate cost estimates. The most important features of our approach are the rapid convergence capability of ACE and its low overhead which permits continuous adaptation during the run time of the system.

The implementation of ACE and its experimentation with commercial DBMSs, in conjunction with the accuracy of the results which have been repeatedly validated for more than two months, makes us optimistic that sooner, rather than later, there will be available efficient query optimizers for HDBMSs. We believe that the work presented in this paper contributes towards achieving this goal.

There are several directions that the ACE method can be extended. First, additional experiments with other commercial DBMSs would strengthen the presented results. Second, the classification of the query types can be refined to improve the accuracy of cost estimation. Third, the cost estimation formula can be extended to explicitly model system-dependent run-time parameters such as system and network load. Finally, the method can be extended to model more complex multi-server queries which require large data moves.

References

[BT94] D. Bitton and C. Turbyfill. A Retrospective on the Wisconsin Benchmark. In M. Stonebraker, editor, *Readings in Database Systems*, pages 422–441. Morgan kaufmann, 1994.

[CR94] C. Chen and N. Roussopoulos. Adaptive Selectivity Estimation Using Query Feedback. In *Proc. of ACM SIGMOD*, 1994.

[Day85] U. Dayal. Query Processing in Multidatabase System. In W. Kim, D. Reiner, and D. Batory, editors, *Query Processing in Database Systems*. Springer Verlag, 1985.

[DKS92] W. Du, R. Krishnamurthy, and M. Shan. Query Optimization in Heterogeneous DBMS. In *Proc. of the 18th International Conference on VLDB*, Vancouver, Canada, 1992.

[DR94] A. Delis and N. Roussopoulos. Management of Updates in the Enhanced Client–Server DBMS. In *Proccedings of the 14th IEEE Int. Conference on Distributed Computing Systems*, Poznan, Poland, June 1994.

[DSD95] W. Du, M.-C. Shan, and U. Dayal. Reducing Multidatabase Query Response Time by Tree Balancing. In *Procs. of the 1995 ACM-SIGMOD Int'l Conf. on Management of Data*, 1995.

[K+85] W. Kim et al., editors. *Distributed Database Query Processing*. Springer-Verlag, 1985.

[Lee64] R.C.K. Lee. *Optimal Estimation, Identification, and Control*. MIT Press, 1964.

[LS92] H. Lu and M. Shan. Global Query Optimization in Multidatabase Systems. *1992 NFS Workshop on Heterogeneous Databases and Semantic Interoperability*, 1992.

[Mar87] M.J. Maron. *Numerical Analysis — A practical approach*. Macmillan Publishing Co., New York, 1987.

[ML86] L.F. Mackert and G.M. Lohman. R* Optimizer Validation and Performance Evaluation for Distributed Queries. In *Procs. of the 12th Intl. Conf. on Very Large Data Bases*, 1986.

[OV91] M. Ozsu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[RES93] N. Roussopoulos, N. Economou, and A. Stamenas. ADMS: A Testbed for Incremental Access Methods. *IEEE Trans. on Knowledge and Data Engineering*, 5(5):762–774, 1993.

[RK86] N. Roussopoulos and H. Kang. Principles and Techniques in the Design of $ADMS\pm$. *Computer*, December 1986.

[Rou91] N. Roussopoulos. The Incremental Access Method of View Cache: Concept, Algorithms, and Cost Analysis. *ACM–Transactions on Database Systems*, 16(3):535–563, September 1991.

[S+79] P. Selinger et al. Access Path Selection in a Relational Database Management System. In *Proc. of ACM SIGMOD*, 1979.

[SL90] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3), 1990.

[SY+89] P. Scheuermann, C. Yu, et al. Report on the Workshop on Heterogeneous Database Systems held at Northwestern University, Evanston, Illinois, Dec. 1989.

[You84] P. Young. *Recursive Estimation and Time-Series Analysis.* Springer-Verlag, New York, 1984.

[ZL94] Q. Zhu and P. Larson. A Query Sampling Method for Estimating Local Cost Parameters in a Multidatabase System. In *Proc. of the 10th International Conference on Data Engineering*, 1994.