A Characterization of the Sensitivity of Query Optimization to Storage Access Cost Parameters[*] [†]

Frederick R. Reiss
Computer Science Division
University of California
Berkeley, CA 94720
phred@cs.berkeley.edu

Tapas Kanungo
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120
kanungo@almaden.ibm.com

ABSTRACT

Most relational query optimizers make use of information about the costs of accessing tuples and data structures on various storage devices. This information can at times be off by several orders of magnitude due to human error in configuration setup, sudden changes in load, or hardware failure. In this paper, we attempt to answer the following questions:

- Are inaccurate access cost estimates likely to cause a typical query optimizer to choose a suboptimal query plan?
- If an optimizer chooses a suboptimal plan as a result of inaccurate access cost estimates, how far from optimal is this plan likely to be?

To address these issues, we provide a theoretical, vector-based framework for analyzing the costs of query plans under various storage parameter costs. We then use this geometric framework to characterize experimentally a commercial query optimizer. We develop algorithms for extracting detailed information about query plans through narrow optimizer interfaces, and we perform the characterization using database statistics from a published run of the TPC-H benchmark and a wide range of storage parameters.

We show that, when data structures such as tables, indexes, and sorted runs reside on different storage devices, the optimizer can derive significant benefits from having accurate and timely information regarding the cost of accessing storage devices.

Keywords

Autonomic computing, databases, storage systems, parametric query optimization, computational geometry

[*]This work was done while Reiss was visiting IBM Almaden Research Center.

[†]Reiss was partially supported by a National Defense Science and Engineering Graduate Fellowship.

ACM SIGMOD 2003, June 9-12, 2003, San Diego, CA.

1. INTRODUCTION

The complexity of database and storage systems has been increasing steadily, and with it the total cost of ownership. While the newer systems can accomplish more, installing and managing them has become increasingly difficult, requiring skilled and expensive database and storage administrators. Many storage and database vendors have started addressing this issue by directing their scientists and engineers to design systems that can automatically configure, monitor, and tune themselves. Since such systems would minimize human intervention, they have the potential to reduce the total cost for the end user.

There are many scenarios in which automatic control of database and storage systems is desirable. A database product like IBM DB2 has nearly 150 parameters that have to be configured during setup. It is not uncommon for database administrators, for example, to use default (and obsolete) values for the storage throughput and latency parameters, or to set incorrectly parameters dealing with a RAID device. Furthermore, while some configuration parameters do not change over time, other parameters like the effective CPU speed and storage parameters can change over time due to load changes on the CPU or on the disk, network delays, device failures, RAID rebuilds, or maintenance tasks like data backups. Monitoring and updating dynamic system parameters in real time is not a pleasant job for any human administrator and some say the job is best done by autonomic machines. Some of these issues have been raised in [36].

However, before we can build algorithms that will allow database systems to adapt themselves to changing levels of storage, network, and CPU loads, we need to understand the sensitivity of these systems to such changes. In this paper, we quantitatively characterize the impact of changes in storage parameters on the database optimizer. We introduce a vector-space framework for analyzing the costs of executing query plans and then use it to analyze the effectiveness of a commercial query optimizer on the TPC-H [12] queries on a commercial database. In particular, we strive to answer the following questions:

- Are inaccurate access cost estimates likely to cause a typical query optimizer to choose a suboptimal query plan?
- If an optimizer chooses a suboptimal plan as a result of inaccurate access cost estimates, how far from optimal is this plan likely to be?

The organization of the rest of this paper is as follows: The next

section describes related literature. In Section 3, we describe our theoretical framework for characterizing the sensitivity of the query optimizer. The conditions under which an optimizer might generate suboptimal plans due to errors in estimating storage access costs are described in Section 4. The impact of these sub-optimal plans is discussed in Section 5. Algorithms used for finding optimal plans and experiments is discussed in Section 6. In Section 7, we give our experimental design, and in Section 8 we provide the results of characterizing the IBM DB2 query optimizer.

2. RELATED WORK

Research in autonomic computing is gaining interest. IBM has started an aggressive program to address this issue [8], which is similar work being done at Stanford and U.C. Berkeley [24]. The importance and issues from the database perspective have been outlined by Weikum and his colleagues [36].

Much of the recent autonomic computing work in databases has centered on improving the accuracy of query optimizer statistics. Chen and Roussopoulos [3] proposed a method of adaptively adjusting selectivity estimates by using the access patterns of queries to collect biased samples of tables. Stillger *et al.* [34] developed an incrementally adaptive query optimization algorithm that updates the database statistics on-line. Lu *et al.* [23] proposed a similar strategy for updating statistics. The LEO project [34] at IBM applies adaptive selectivity estimation to a real commercial system. Chaudhuri and Narasayya [2] proposed a method for automatically selecting a subset of statistical measurements. Deshpande and Hellerstein [13] studied methods of reducing the bandwidth needed to pass optimizer statistics between nodes of a federated database system.

Parametric query optimization has been studied extensively [15, 21, 6, 18, 14, 25, 28, 5, 4, 19]. For situations in which the selectivities of predicates in relational queries depend on parameters passed in by the user during runtime, this approach to optimization creates a set of optimized plans instead of just one. Depending on the user-specified runtime parameter, an appropriate query plan is selected. Much of the work in this field is based on a geometric interpretation of the query optimization problem that is similar to the vector-space approach we use in this paper. Some researchers have conducted experimental analyses to determine the number of potentially-optimal plans for queries with various types of join graphs, but we do not know any such analyses to determine the relative optimality of different plans under varying circumstances.

Researchers have created several different models of the performance of storage systems. Worthington *et al.* [37] and Ruemmler and Wilkes [29] describe realistic simulation models for storage devices. The complexity of tuning database systems is described in various product manuals [31, 7]. Evaluation of database systems is addressed by the Transaction Processing Council (TPC) which produces the TPC-H benchmark used in this paper [12, 10]. Tools for empirical evaluation of query optimizers were described in [33]. Impact of RAID rebuilds on storage access was addressed by Brown and Patterson [1].

For more details on query optimization the reader is referred to the original article by Selinger *et al.* [30], and various surveys [20, 22].

3. THEORETICAL FRAMEWORK

In this section, we describe a vector-based framework for reasoning about the decisions a query optimizer will make in the face of inaccurate access cost estimates. These constructs are similar to concepts used in some recent work on parametric query optimization [14, 19].

3.1 Cost Model

We assume that the database has access to n time-shared resources for performing operations on tuples. Each resource r_i has a *true resource cost* c_i. The true cost of using u_i units of resource r_i is $u_i \cdot c_i$.

For the purposes of this paper, we assume that query execution adheres to an *additive cost model* similar to that used by most query optimizers today [30, 20, 22]. The *true total cost* of a query plan is given by:

$$T = \sum_{i=1}^{n} u_i \cdot c_i. \tag{1}$$

The plan that minimizes T given in Equation 1 for a particular query is the *true optimal plan* for that query.

The time required to access data stored on a hard drive is generally not directly proportional to the amount of data accessed. To model hard drives more accurately, we treat a hard disk d as two resources: d_s to model queueing time, rotational delays, track-to-track seeks, etc.; and d_t to model sequential reads and writes. For example, a disk operation that involved 2 seeks and read a total of 3 blocks of data would incur a cost of $2 \cdot c_{d_s} + 3 \cdot c_{d_t}$. This model of disk drive access time, though not entirely accurate, is a good first approximation of drive behavior [29, 37].

3.2 Vector Notation

We can convert the scalar notation of the previous section to vector notation as follows:

Recall the cost equation

$$T = \sum_{i=1}^{n} u_i \cdot c_i. \tag{2}$$

where c_i is cost of using a unit of resource i and u_i is the number of units of resource i that a query plan uses.

We can rewrite this equation as

$$T = U \cdot C \tag{3}$$

Where $U = (u_1, u_2, \ldots u_n)$ and $C = (c_1, c_2, \ldots c_n)$. We call C the *resource cost vector* and U the *resource usage vector* for the query plan.

3.3 Modeling Inaccurate Access Costs

Since we focus in this paper on storage access cost parameters, we assume that the query optimizer makes accurate estimates of the amount of each resource that a given query plan uses. In particular, we assume that the optimizer's estimates of the selectivities of predicates and the sizes of intermediate results are accurate.

For resource usage costs, however, we assume that the optimizer may not know the exact value of each cost c_i. Rather, the optimizer has a set of *estimated resource costs* $\hat{c}_i$. The optimizer uses these

estimated resource costs to compute an *estimated total cost* for each candidate query plan:

$$\hat{T} = \sum_{i=1}^{n} u_i \cdot \hat{c}_i. \quad (4)$$

or, in vector notation,

$$\hat{T} = U \cdot \hat{C} \quad (5)$$

Using any of a variety of optimization algorithms [30, 22, 18, 32], the optimizer chooses a query plan that minimizes the estimated total cost $\hat{T}$. We call this plan the *estimated optimal plan*.

Differences between the estimated resource costs and the true resource costs can arise from sources such as:

- The storage or network hardware may undergo a period of heavy load or a partial failure.
- The optimizer may use costs for hardware that is older or newer than the actual hardware.
- The optimizer may use costs for a different number of disks, CPU's, network cards, etc., than the system actually uses.

We assume that the optimizer's costs may be off by at most a certain finite amount. That is, the true resource cost vector must always fall within a large but finite region of the cost vector space that contains the estimated resource cost vector. We call this region the *feasible cost region*, and we refer to any resource cost vector that falls within this region as a *feasible cost vector*.

4. CIRCUMSTANCES LEADING TO SUBOPTIMAL PLAN CHOICES

An important element of our sensitivity analysis is the answer to the question: Under what circumstances will a query optimizer choose a suboptimal query plan due to inaccurate access cost estimates?

In this section, we introduce *switchover planes* and *regions of influence*, constructs which formalize which regions of the resource cost vector space cause a particular candidate optimal query plan to become the optimal plan.

4.1 Equicost Lines

The dot-product of U and C as defined in Equation 3 is equal to the component of U in the direction of C. Thus, all resource utilization vectors along a line perpendicular to C will have the same total cost under C (See Figure 1.). We call such a line an *equicost line*.

More precisely, let A and B be the resource utilization vectors of query plans a and b, respectively. We notice that, if the resource cost vector C is the same for both plans, then the two plans have the same total cost if and only if $A \cdot C = B \cdot C$. It follows that, if both plans have the same cost, then the dot product $(A-B)\cdot C = 0$. Thus, any two utilization vectors A and B that lie on an equicost line result in the same total cost.

4.2 Switchover Planes

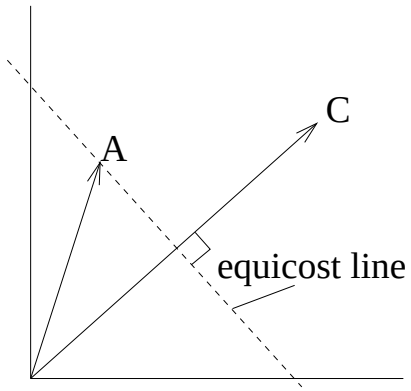

Figure 1: An equicost line. Any resource utilization vector along this line will have the same cost as query a under the resource costs given by C.

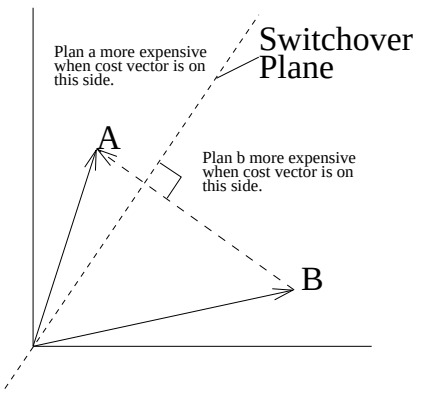

Figure 2: The switchover plane for two queries. A is the resource usage vector for query a, and B is the resource usage vector for query b. If the resource cost vector is on the B-dominated side of the switchover plane, then plan b is more expensive than plan a. Likewise, if the resource cost vector is on the A-dominated side of the plane, plan a is more expensive.

Consider two vectors A and B, representing the resource utilizations of query plans a and b, respectively. The *switchover plane* of A and B is a plane such that, if the resource cost vector C is in the switchover plane, then the total cost of query plan a is the same as the total cost of query plan b.

More precisely,

$$\begin{aligned} \text{Switchover}_{A,B} &= \{C \in \Re^n | A \cdot C = B \cdot C\} \\ &= \{C \in \Re^n | (A - B) \cdot C = 0\} \end{aligned}$$

$\text{Switchover}_{A,B}$ is a plane through the origin and orthogonal to the vector $A - B$ (See Figure 2).

4.3 Half-Spaces

$\text{Switchover}_{A,B}$ divides the vector space into two half-spaces. If the resource cost vector is on one side of the switchover plane, a always has a higher cost than b. On the other side, b is always more expensive than a.

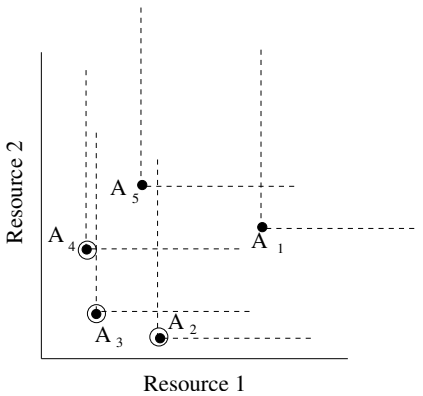

Figure 3: Dominated plans. Each plan a dominates over any plan that is in Q_a, the positive first quadrant relative plan a (denoted by dotted lines). Thus, plans A_1, and A_5 are not candidate optimal plans because they are dominated by other plans.

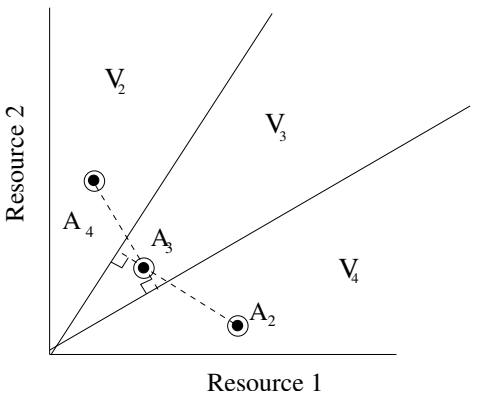

Figure 4: Regions of influence. Each candidate optimal plan has a regions of influence in which it is the optimal plan. These regions of influence are cone shaped with the apex at the origin. The regions of influence V_2, V_3, and V_4 corresponding to the candidate optimal plans A_2, A_3, and A_4 are shown here.

More precisely, we define the *A-dominated half-space* as $\{C \in \Re^n | \, A \cdot C > B \cdot C\}$, and the *B-dominated half-space* as $\{C \in \Re^n | \, A \cdot C < B \cdot C\}$.

4.4 Candidate Optimal Plans

A relational query optimizer typically considers a very large set of potential query plans when optimizing a query. However, only a subset of these plans can ever become the optimal plan as a result of changes in access method costs. We call this set of plans the *candidate optimal* plans.

Let Q denote the positive first quadrant and be defined as $Q = \Re^n_+ \cup \{0\}$. The positive first quadrant relative to a plan a is denoted by Q_a and is defined as $Q_a = \{x \in \Re^n | x = a + q \text{ where } q \in Q\}$.

A query plan a with resource usage vector A is *candidate optimal* if there exists a feasible resource cost vector C such that, for *any* query plan b with resource usage vector $B \neq A$, $A \cdot C \leq B \cdot C$. In fact, one can easily show that a plan cannot be a candidate optimal plan if it lies in the positive first quadrant relative to any other plan (See Figure 3). That is, a plan a *dominates* over all plans whose resource usage vectors line in Q_a, the positive first quadrant relative to a.

4.5 Regions of Influence

We are interested in determining the resource cost scenarios under which different query plans become optimal. Let A denote the resource usage vector for query plan a. The *region of influence* of A is the set of cost vectors under which plan a is the optimal query plan.

More precisely, let $R = \{A_1, A_2, \ldots, A_m\}$ denote the set of resource usage vectors for all plans for query q, and let $U \subset \Re^n$ denote the feasible cost region. We define the region of influence V_i for plan $A_i \in R$ with respect to R and U as:

$$V_i = \{v \in U | A_i \cdot v \leq A_j \cdot v \text{ for all } j \neq i\}. \quad (6)$$

Regions of influence are convex polytopes whose borders are switchover planes (See Section 4.2). They are similar to Voronoi regions [26] except that there is a Voronoi region corresponding to every "site" whereas in our case there is a Voronoi "cone" corresponding to every candidate optimal plan and *none* for plans that are not candidate optimal. This is illustrated in Figure 4.

5. POTENTIAL SEVERITY OF SUBOPTIMAL PLAN CHOICES

In addition to knowing whether a query optimizer may choose a suboptimal plan as a result of inaccurate access method cost parameters, it is also important to know how far from optimal such a choice would be.

In this section, we introduce *relative total cost*, a measure of how far from optimal a query plan is under a set of resource costs. We use relative total cost to formalize how far from optimal the query optimizer's choice of plan will be.

5.1 Relative Total Cost

We define the *relative total cost* of query plan a with respect to query plan b under resource costs C as

$$T_{rel}(a, b, C) = \frac{\text{Total cost of } a}{\text{Total cost of } b}. \quad (7)$$

Relative total cost is useful for two particular purposes:

- The set of cost vectors for which the relative cost of plans A and B is exactly 1 is the switchover plane (See Section 4.2.) between the two plans.
- Relative total cost is a way of *normalizing* the total cost of a query plan. In particular, relative total cost is *unitless*. It is generally more useful to know that a particular query plan is k times as expensive as the optimal plan than it is to know that a query plan has a total cost that is j units greater than the total cost of the optimal plan.

5.2 Global Relative Total Cost

If the query optimizer chooses a suboptimal plan, we would like to know how far from optimal the plan is likely to be. To gauge the

relative "badness" of a query plan, we use a measure called *global relative cost*.

The global relative cost of query plan a under resource costs C, written $GTC_{rel}(a, C)$, is defined as the relative total cost of a with respect to the *optimal* query plan under C. Intuitively, global relative cost measures how many times as fast the query would have run had the optimizer chosen the correct plan.

We observe that query plan a is a candidate optimal plan (See Section 4.4.) if and only if there exists a cost vector C such that $GTC_{rel}(a, C) = 1$.

5.3 Invariance of Relative Total Cost

Multiplying the resource cost vector by any constant scaling factor will not change the relative total cost of any pair of resource utilization vectors.

Consider a cost vector C and an arbitrary scaling factor k. Let A and B be the resource usage vectors for plans a and b, respectively. Let α and β denote the angles between A and C, and B and C, respectively. Then we have:

OBSERVATION 1. $T_{rel}(a, b, C) = T_{rel}(a, b, kC)$

Proof:

$$T_{rel}(a, b, kC) = \frac{A \cdot kC}{B \cdot kC} = T_{rel}(a, b, C). \quad \square$$

Intuitively, Observation 1 means that the relative total costs of different query plans do not change at all unless the costs of accessing resources change *relative to each other*. A change in the system that makes all resources faster or slower by the same multiplicative factor will have no effect on the relative costs of different query plans.

Observation 1 also implies that regions of influence (See Section 4.5.) are cone-shaped and radiate out from the origin of the cost vector space.

5.4 An Upper Bound

If the optimizer knows the cost of accessing storage devices to within a factor of δ, then the query plan it chooses will be within a factor of δ^2 of optimal. The following theorem and corollary state this bound more rigorously:

THEOREM 1. *Let a and b be two query plans. Let $C = (c_1, c_2, ..., c_n)$ be a resource cost vector, and let $\hat{C} = (\hat{c}_1, \hat{c}_2, ..., \hat{c}_n)$ be another resource cost vector such that $\hat{c}_i$ is between c_i/δ and $c_i \cdot \delta$ for all i from 1 to n. If the relative total cost of a and b under C, $T_{rel}(a, b, C)$, is γ, then the relative total cost of a and b under $\hat{C}$ must be between γ/δ^2 and $\gamma \cdot \delta^2$.*

Proof:
Let A and B be the resource usage vectors for plans a and b, respectively. Let $C_{max} = (c'_1, c'_2, ..., c'_n)$ denote the choice of $\hat{C}$ that maximizes $T_{rel}(a, b, \hat{C})$. Then we have:

$$\begin{aligned} T_{rel}(a, b, C_{max}) &= \frac{A \cdot C_{max}}{B \cdot C_{max}} \\ &\leq \frac{A \cdot (\delta \times C)}{B \cdot C_{max}}, \end{aligned}$$

since each element of C_{max} is within a factor of δ of the corresponding element of C.

$$\begin{aligned} &\leq \frac{A \cdot (\delta \times C)}{B \cdot (\frac{1}{\delta} \times C)} \\ &= \delta^2 \frac{A \cdot C}{B \cdot C} \\ &= \delta^2 \times T_{rel}(a, b, C). \end{aligned}$$

The other half of the bound follows from the same reasoning. $\square$

Corollary: If the optimizer's cost estimates are within a multiplicative factor of $1/\delta$ to δ of the true costs, then the cost of the plan the optimizer chooses will be within δ^2 of the cost of the optimal plan.

The above bound is tight, as demonstrated by the following example:

EXAMPLE 1. *Consider two plans a and b with two-element resource usage vectors, $A = (1, 0)$ and $B = (0, 1)$, respectively. Under the cost vector $C_1 = (1, 1)$, the relative total cost of the plans is:*

$$T_{rel}(a, b, C_1) = \frac{A \cdot C_1}{B \cdot C_1} = \frac{(1, 0) \cdot (1, 1)}{(0, 1) \cdot (1, 1)} = 1.$$

Under the cost vector $C_2 = (\delta, 1/\delta)$, the relative total cost of the plans is:

$$T_{rel}(a, b, C_2) = \frac{A \cdot C_2}{B \cdot C_2} = \frac{(1, 0) \cdot (\delta, 1/\delta)}{(0, 1) \cdot (\delta, 1/\delta)} = \delta^2.$$

Thus we see that the bound is tight. $\square$

5.5 A Tighter Bound for a Special Case

If the relationship between the resource usage vectors of candidate optimal plans meets certain criteria, then a significantly tighter bound on induced optimizer error can apply.

Let A and B denote the resource vectors for query plans a and b, respectively. We call a and b *complementary query plans* if there exists an i such that the ith element of A is nonzero and the ith element of B is zero, or vice versa.

If there are *no* complementary plans for a query, then arbitrary changes in resource costs can only induce a constant change in the relative cost of any two query plans. Regardless of how inaccurate the query optimizer's estimated resource costs are, the optimizer is guaranteed to choose a plan that is within a (potentially large) constant factor of optimal.

We will use the following result regarding ratios of dot products in the proof of the theorem in this section:

LEMMA 1. *Let a_1 and b_1 be real numbers greater than zero. Then, for any real numbers a_2 and b_2 greater than zero, such that*

$\frac{a_2}{b_2} \leq \frac{a_1}{b_1}$, *and any real numbers* $c_1, c_2 \geq 0$*:*

$$\frac{a_1c_1 + a_2c_2}{b_1c_1 + b_2c_2} \leq \frac{a_1}{b_1} \tag{8}$$

Proof:
If $c_1 = 0$, then the lemma is trivially true. Otherwise:

$$\begin{aligned}
\frac{a_2}{b_2} &\leq \frac{a_1}{b_1} \\
a_2b_1 &\leq a_1b_2 \\
a_2b_1c_2 &\leq a_1b_2c_2 \\
a_1b_1c_1 + a_2b_1c_2 &\leq a_1b_1c_1 + a_1b_2c_2 \\
b_1(a_1c_1 + a_2c_2) &\leq a_1(b_1c_1 + b_2c_2) \\
\frac{a_1c_1 + a_2c_2}{b_1c_1 + b_2c_2} &\leq \frac{a_1}{b_1}
\end{aligned}$$

That completes our proof. □

We now prove a bound on the relative speedup of non-complementary plans.

THEOREM 2. *Let* a *and* b *be query plans that are not complementary, and let* $A = (a_1, a_2, ..., a_n)$ *and* $B = (b_1, b_2, ..., b_n)$ *be the resource usage vectors for* a *and* b*, respectively. Let* $r^{a,b}_{max} = max_{1 \leq i \leq n} \frac{a_i}{b_i}$*, and let* $r^{a,b}_{min} = min_{1 \leq i \leq n} \frac{a_i}{b_i}$*. Then the relative total cost of* a *and* b *under any cost vector* C *must be between* $r^{a,b}_{min}$ *and* $r^{a,b}_{max}$*.*

Proof:
Without loss of generality, assume that all elements of A and B are greater than zero and that $\frac{a_1}{b_1} = r^{a,b}_{max}$. Consider an arbitrary resource cost vector $C = (c_1, c_2, ..., c_n)$.

From Lemma 1, we have:

$$\frac{a_1c_1 + a_2c_2}{b_1c_1 + b_2c_2} \leq \frac{a_1}{b_1}$$

Applying the lemma again gives us:

$$\frac{(a_1c_1 + a_2c_2) + a_3c_3}{(b_1c_1 + b_2c_2) + b_3c_3} \leq \frac{a_1c_1 + a_2c_2}{b_1c_1 + b_2c_2} \leq \frac{a_1}{b_1}$$

Applying the lemma $n - 3$ more times gives:

$$\begin{aligned}
\frac{a_1c_1 + a_2c_2 + ... + a_nc_n}{b_1c_1 + b_2c_2 + ... + b_nc_n} &\leq \frac{a_1}{b_1} \\
\frac{A \cdot C}{B \cdot C} &\leq \frac{a_1}{b_1}
\end{aligned}$$

So, by definition:

$$T_{rel}(a, b, C) \leq \frac{a_1}{b_1} = r^{a,b}_{max}$$

The other half of the bound follows from the same reasoning. □

Corollary: If there are no complementary plans, then the cost of the plan the optimizer chooses will be within a multiplicative factor of

$$max_{a,b}(max(r^{a,b}_{min}, r^{a,b}_{max})) \tag{9}$$

of the cost of the optimal plan, where a and b range over all pairs of candidate optimal plans.

Of course, the ratios between corresponding elements of two resource usage vectors could be quite large, as the following example illustrates:

EXAMPLE 2. *Consider a query with the following join graph:*

$$T_1 - T_2 - T_3 \tag{10}$$

We assume that join predicates are independent and that each table has one million tuples and the joins have selectivities of 1×10^{-8}*. We also assume that table* T_1 *is located on storage resource 1, and all other tables and indexes are on storage resource 2.*

Consider the following two alternative plans for the query:

- *Plan* A *scans table* T_1 *and probes indexes on tables* T_2 *and* T_3*, in that order.*
- *Plan* B *scans table* T_3 *and probes indexes on tables* T_2 *and* T_1*, in that order.*

Since it scans table T_1*, plan* A *will read all one million tuples from the table and does not use any indexes on* T_1*. Plan* B*, on the other hand, executes ten thousand probes against the index on table* T_1 *and fetches one hundred tuples from the underlying table.*

Thus, plan A *will use ten thousand times more of resource 1 (the disk storing table* T_1*) than plan* B*. So the maximum ratio between the cost vectors of plans* A *and* B *is 10000.* □

As the ratios between corresponding elements of resource usage vectors increase, the bound in Theorem 2 becomes less and less meaningful. In particular, if there exist complementary plans for a given query, then the bound does not hold at all. In this case, changes in resource costs by a multiplicative factor between $1/\delta$ and δ can cause the query optimizer to choose a plan that is suboptimal by as much as δ^2, following the bound in Theorem 1.

5.6 Circumstances Leading to Complementary Plans

In our analyses of resource usage vectors (See Section 8.2), we discovered three main causes for a given pair of plans being complementary or having a large ratio between corresponding elements of their resource usage vectors.

- If the plans access different numbers of tuples from a particular table, we call them *table complementary* plans.
- If the plans access the same numbers of tuples from a table, but they use different access paths to retrieve the tuples, we call them *access path complementary* plans.
- If of the plans makes extensive use of temporary out-of-core data structures such as sorted runs or hash buckets, while the other plan does not, we call the plans *temp complementary* plans.

6. ALGORITHMS

In this section we provide the algorithms used for finding the worst-case performance degradation due to incorrect query plan. We also provide an algorithm for computing the resource usage vector corresponding to a particular query plan. These algorithms were used in the experiments described in Section 8.

6.1 Worst-Case Analysis

Our first experiment aimed to determine the maximum possible effect of access method costs on the query optimizer's choice of query plan. The experiment proceeded as follows:

1. Choose a set of typical resource costs $C_0 = (c_1, c_2, ..., c_n)$, the *initial resource cost vector*. We call the query plan p_0 that is optimal under these resource costs the *initial query plan*.
2. Allow all resource costs c_i to vary independently by a multiplicative factor between $1/\delta$ and δ. For each possible set of costs, compute the global relative total cost (See Section 5.2.) of the initial query plan.
3. Report the maximum global relative cost as δ, the size of the cost ranges, increases.

In designing this experiment, we took advantage of the following observation:

OBSERVATION 2. *If the feasible cost region is a convex polytope with vertices V, then, for any query plan a, there must exist a vertex $v \in V$ such that, for any $u \in V$, $GTC_{rel}(a, v) \geq GTC_{rel}(a, u)$.*

Briefly, the proof of Observation 2 goes as follows: Let a and b be two arbitrary candidate optimal plans with resource usage vectors A and B. Recall from Section 5.1 that the relative total cost of plans a and b is $T_{rel}(a, b, C) = \frac{A \cdot C}{B \cdot C}$. The value of this expression is determined by the angles between A and B, respectively, and the resource cost vector C. Because of this relationship, $T_{rel}(a, b, C)$ is monotonic as C varies along any straight line. It follows that there must exist a vertex of the feasible cost region that maximizes $T_{rel}(a, b, C)$. Since a and b are arbitrary plans, there must be a vertex $v \in V$ that maximizes global relative cost.

Observation 2 allowed us to calculate the worst-case global relative cost for a query plan by measuring the global relative cost at each of the vertices of the feasible cost region. Intuitively, a vertex of the feasible cost region for this experiment is a point at which the cost of each resource i is either $c_i \cdot \delta$ or c_i/δ.

We used the DB2 query optimizer to calculate the cost of the optimal plan at each vertex of the feasible cost region. To determine the cost of the initial query plan under arbitrary resource costs, we used the optimizer to find the total cost of the plan at several different resource cost vectors. We then estimated the resource usage vector of the initial query plan using a least squares estimation technique described in the next section.

6.1.1 Least Squares Estimation of Resource Usage Vectors

An important aspect of our experimental design was to use the query optimizer from a well-established DBMS for computing the costs of query plans. In order to take advantage of the capabilities of an industrial-strength query optimizer, however, we needed to work around its limitations. In particular, commercial optimizers do not provide access to resource usage vectors for the query plans they consider. To obtain these vectors for different query plans, we needed to estimate them from information that the optimizer *does* provide. We made this estimation as follows:

Let n be the number of resources. Consider a candidate optimal plan p, and m ($m \geq n$) resource cost vectors $C_1, C_2, \ldots, C_m$ under which plan p is the optimal plan. Let $t_1, t_2, \ldots, t_m$ denote the total cost of plan p under the resource costs given by $C_1, C_2, \ldots, C_m$, respectively. Finally, let U_p denote the resource usage vector for plan p. Since the cost model is linear, we have:

$$\begin{aligned} C_1 \cdot U_p &= t_1 \\ C_1 \cdot U_p &= t_2 \\ &\vdots \\ C_m \cdot U_p &= t_m \,. \end{aligned}$$

We can write the above equations in a compact form as $CU_p = T$ where C is a $m \times n$ matrix with m rows of cost vectors C_i, and T is a $m \times 1$ vector of the observed costs reported by the database optimizer. The least squares estimate of U_p is given by [35, 17, 27]

$$\hat{U}_p = \left[C^t C\right]^{-1} C^t T$$

Solving for U_p gives the resource usage vector for query plan p. The matrix inverse is computed using Gaussian elimination [35, 17, 27].

In our experiments, we obtained each sample point (C_i, t_i) by feeding the costs in C_i into the DB2 optimizer and asking it for the optimal query plan and its estimated total cost t_i. In the ideal no-error case, $m = n$ points are sufficient to find the n unknowns c_i. However, to compensate for quantization error within the query optimizer, we always used at least $m = 2n$ samples when estimating resource usage vectors using least squares.

We validated this method by using the estimated resource usage vectors to compute the total cost of the query plan under additional cost vectors and comparing this cost with the cost that the query optimizer returns. We found the discrepancy between these computed total costs to be less than one percent.

6.2 Analysis of Resource Usage Vectors

Our second set of experiments involved computing the resource usage vectors for every candidate optimal plan (See Section 4.4.) for a given query and feasible cost region. The purpose of these experiments was to provide greater insight into the results of the first set of experiments.

6.2.1 Finding Candidate Optimal Plans

The analysis of each query began by identifying the candidate optimal plans for the query and determining their resource usage vectors.

We located the candidate optimal plans through a five-step process:

1. Choose a set of resource cost vectors S, all of which fall within the feasible cost region (See Section 3.3.).
2. Use the query optimizer to determine the optimal plan and its total cost under each cost vector in S.
3. Sample additional resource cost vectors until there are enough sample points to estimate the resource usage vector of each query plan found.
4. Use least squares to estimate (See Section 6.1.1.) the resource usage vectors of the plans found thus far.

5. Determine whether the plans found so far comprise all the candidate optimal plans. If there are additional candidate optimal plans, go back to step (3).

To determine whether a set of candidate optimal plans is complete, we took advantage of the following observation:

OBSERVATION 3. *Let C_1 and C_2 be two arbitrary resource cost vectors. If query plan a is the optimal plan under the costs given by both C_1 and C_2, then a is the optimal plan under any convex combination of C_1 and C_2.*

Briefly, the proof of Observation 3 goes as follows: Let $b \neq a$ be an arbitrary query plan, and let A and B denote the resource usage vectors for plans a and b, respectively. Let β be a constant such that $0 \leq \beta \leq 1$, and let the convex combination of the cost be $\bar{C} = \beta C_1 + (1-\beta)C_2$. Then we have:

$$\begin{aligned}
\text{Cost of plan } a \text{ under } \bar{C} &= A \cdot \bar{C} \\
&= A \cdot (\beta C_1 + (1-\beta)C_2) \\
&= \beta A \cdot C_1 + (1-\beta) A \cdot C_2 \\
&\leq \beta B \cdot C_1 + (1-\beta) B \cdot C_2 \\
&= B \cdot \bar{C} \\
&= \text{Cost of plan } b \text{ under } \bar{C}
\end{aligned}$$

Corollary to Observation 3: If query plan a is optimal at every vertex of a convex polytope in the cost vector space, then it is optimal at every point within the polytope.

Observation 3 provided us with a method to determine whether our set of candidate optimal plans was complete. The method involved dividing the feasible cost region into polytopes such that, for each polytope P, a single plan a was optimal at every vertex of P. To generate such polytopes, we calculated the region of influence (See Section 4.5.) of each plan under the assumption that our set of candidate optimal plans was complete and contracted each region of influence by a very small amount.

7. EXPERIMENT DESIGN

Using the theoretical framework from Section 3, we designed a set of experiments that measure the sensitivity of a query optimizer to changes in access method costs on a set of queries. In choosing our experimental setup, we attempted to choose a setup similar to real-world databases. Toward this end, we used the query optimizer from a leading commercial database and the database schema and queries from a well-known database benchmark.

7.1 Optimizer

In order to obtain results that apply to real-world database systems, we designed our experiments to work with any query optimizer that meets the following criteria:

- The optimizer uses a linear cost model, as described in Section 3.1.
- The optimizer allows the user to set the values of all resource costs.
- The optimizer reports the estimated total cost (See Section 3.3.) of the estimated optimal plan, as well as enough information to identify each plan uniquely.

Most commercial query optimizers in use today support the above functionality. For the results in this paper, we used a well-known commercial query optimizer, the IBM DB2 Version 8.1 for Linux optimizer [9, 16]. This optimizer considers a robust set of alternative plans, including plans with bushy join trees and plans with such exotic operations as index ORing and star joins.

7.2 Database Design and Statistics

We chose the database schema from the TPC-H benchmark for our test database. In particular, we used the indexes that IBM used in an actual published run of the benchmark [10]. Information on the specific indexes used can be found in the Full Disclosure Report on the benchmark run [10]. We obtained optimizer statistics from the 100 GB database described in the report. These statistics were the result of running the DB2 SQL command `RUNSTATS ON TABLE <table name> WITH DISTRIBUTION AND DETAILED INDEXES ALL` on each table in the schema, after using the Transaction Processing Performance Council's `dbgen` software [11] to load the database. The benchmark group at IBM used the `db2look` [7] utility to create a dump of the statistics, and we loaded this dump into the catalog of our empty test database. With the transplanted statistics in place, the query optimizer on our test database estimated operator selectivities for the database used in the benchmark run.

7.3 System Parameters

We ran our experiments using the DB2 Enterprise Edition for Linux Version 8.1. The DB2 query optimizer was configured to generate query plans for a 32-way symmetric multiprocessor. To speed our experiments, we loaded an identical copy of the database statistics into each node of a 4-node cluster of 1.39 GHz Pentium III computers. The computers in the cluster each ran a separate instance of DB2 Enterprise Edition for Linux Version 8.1 [9] on top of Red Hat Linux version 7.3.

To ensure that the query optimizer on our test databases "saw" the same system parameters it would use while running the TPC-H benchmark, we duplicated all of the DB2 environment variable settings and database parameters listed in the "Tunable System Parameters" section of the Full Disclosure Report that affect the DB2 query optimizer. We used the `db2fopt` utility to "fool" the optimizer into thinking that our test database had a 2.5 GB buffer pool and a 512 MB sorting heap, as on the benchmark system. We configured the optimizer to use the same optimization level that was used in the benchmark run.

The following table lists some of the more important system parameters we set.

Parameter Name	Value
DB2_EXTENDED_OPTIMIZATION	YES
DB2_ANTIJOIN	Y
DB2_CORRELATED_PREDICATES	Y
DB2_NEW_CORR_SQ_FF	Y
DB2_VECTOR	Y
DB2_HASH_JOIN	Y
DB2_BINSORT	Y
INTRA_PARALLEL	YES
FEDERATED	NO
DFT_DEGREE	32
AVG_APPLS	1
LOCKLIST	16384
DFT_QUERYOPT	7
OPT_BUFFPAGE	640000
OPT_SORTHEAP	128000

To the extent that the DB2 query optimizer can accurately predict the cost of executing query plans, the results of our experiments should apply directly to a 100 GB instance of the TPC-H benchmark.

7.4 Queries

We analyzed the 22 queries in the TPC-H benchmark suite in our experiments. The text of these queries can be found in the benchmark specification document [12].

8. EXPERIMENTAL RESULTS

In this section, we present the results of our experiments. Using the experimental setup in Section 7 and the procedures in Section 6, we performed two separate analyses of the 22 TPC-H queries.

8.1 Results of Worst-Case Analysis

Our worst-case analysis experiment used the algorithm in Section 6.1 to determine the maximum possible effect of various multiplicative changes in resource costs. To model a system administrator who leaves DB2's resource cost estimates at their default values instead of computing the actual values for her storage setup, we used DB2's default values of 24.1 time units and 9.0 time units for d_s and d_t (See Section 3.1.), respectively. We used a starting CPU cost of 1.0×10^{-6} time units per instruction. We then allowed these costs to vary independently by multiplicative factors ranging from $\frac{1}{\delta}$ to δ. For each level of variation, we recorded the maximum global relative total cost (See Section 5.2.) that could occur if each of the query optimizer's resource cost estimates was off by a multiplicative factor of up to δ.

8.1.1 All Tables and Indexes on Same Device

In our first worst-case analysis experiment, we simulated a database system that places all of its tables and indexes on the same storage device. That is, we constrained the cost to access a page of tuples from a given table to be the same as for any other table in the database. Thus, there was effectively a total of three resources in this analysis: The two disk parameters d_s and d_t (See Section3.1.) and the CPU.

As shown in Figure 5, the DB2 query optimizer proved relatively insensitive to changes in resource costs in this experiment. Even when we allowed costs to vary arbitrarily by a factor of 10000, the initial query plan was never more than a five times more expensive than the optimal query plan. The shape of the curves indicates that

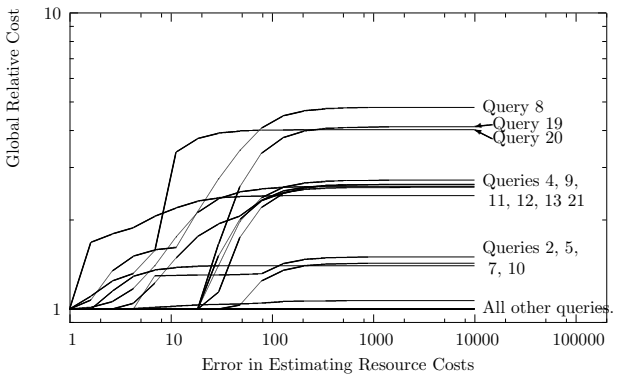

Figure 5: Worst-case global relative cost of the TPC-H queries with all tables on the same storage device. The x-axis measures the error in the optimizer's estimates of resource costs. At point δ on the x-axis, the true cost corresponding to each estimated resource cost c_i is allowed to range from $\frac{c_i}{\delta}$ to $\delta \times c_i$. Each line represents the worst-case relative performance of one of the 22 TPC-H query plans that the optimizer would choose with increasingly inaccurate resource cost estimates. Even under extreme errors in estimating the costs of accessing resources, the cost of the chosen plan was within a factor of 5 of optimal. *Note that the y-axis of this plot is on a logarithmic scale.*

all the queries had only non-complementary plans and therefore followed the constant bound in Section 5.5. For 10 of the queries, the total cost of initial query plan was within a factor of 0.02 of that of the optimal plan throughout the experiment.

The queries that displayed the most variation in global relative cost were queries 8, 19 and 20. We used DB2's `EXPLAIN PLAN` facility to examine the candidate optimal query plans for these three queries. In the case of queries 8 and 19, we found that the choice of join method for a join between the `LINEITEM` and `PART` tables was relatively sensitive to the relative cost of sequential and random I/O. Recall from Section 3.1 that we model a disk drive d using two parameters, d_s and d_t. When the "seek time" parameter d_s was relatively high, the optimizer choose to join `LINEITEM` and `PART` using a hash join. When the "transfer time" parameter d_t was relatively high, the optimizer favored using a foreign key index on `LINEITEM` to perform index nested loops join with the `PART` table. With query 20, the optimizer initially chose to use indexes to filter the `PARTSUPP` and `SUPPLIER` tables before joining them. When random I/O became expensive, the optimizer switched to table scans. Removing the filter operation changed the effective cardinalities of these relations, causing a different join order to become optimal. However, this new join order did not significantly alter the amount of data read from the other tables in the query.

8.1.2 All Tables and Indexes on Different Devices

In our second worst-case analysis experiment, we simulated a database system that places all of its tables and indexes on different storage devices. We allowed the cost of accessing each table and the cost of accessing each table's indexes to vary independently in this experiment. We also allowed the cost of accessing temporary space to vary independently of all other storage costs. Due to limitations of the database system we used, we modeled all indexes for a given table as being on the same storage device. We also kept

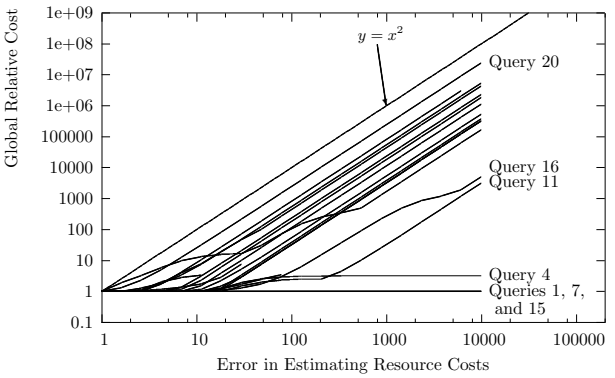

Figure 6: Worst-case global relative cost of the TPC-H queries with all tables and their indices on different storage devices. The x-axis measures the error in the optimizer's estimates of resource costs. At point δ on the x-axis, the true cost corresponding to each estimated resource cost c_i is allowed to range from $\frac{c_i}{\delta}$ to $\delta \times c_i$. Each line represents the worst-case relative performance of one of the 22 TPC-H query plans that the optimizer would choose with increasingly inaccurate resource cost estimates. Errors in estimating the costs of accessing resources can cause the query optimizer's chosen plan to be suboptimal by an large margin. *Note that the y-axis of this plot is on a logarithmic scale.*

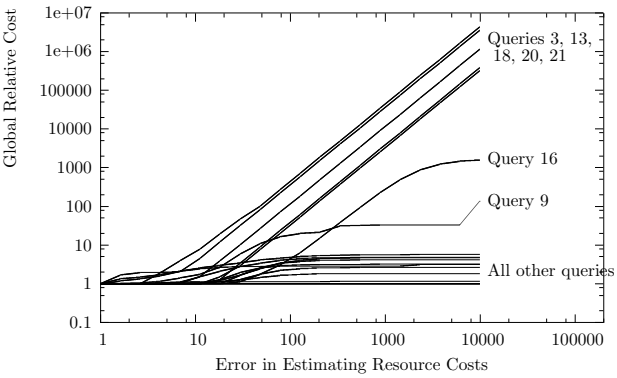

Figure 7: Worst-case global relative cost of the TPC-H queries with a separate storage device for each table and its corresponding indexes. The x-axis measures the error in the optimizer's estimates of resource costs. Each line represents the worst-case relative performance of one the 22 TPC-H query plans that the optimizer would choose with increasingly inaccurate resource cost estimates. Errors in estimating the costs of accessing resources had an effect in between the effects in Figures 5 and 6.

the disk parameters d_s and d_t (See Section 3.1.) in a fixed ratio to reduce the running time of the experiment. For a k-table query, this analysis effectively considered the cost of accessing $2k + 2$ resources: one resource for each table and one resource for each table's indexes, plus a single temporary resource and a single CPU resource.

The results of this second analysis are quite different from the first (See Section 8.1.1.). As the variation in access cost, δ, increases, the worst-case global relative cost increases quadratically with δ for 18 of the 22 queries. This asymptotic behavior indicates that the queries had complementary plans and followed the quadratic bound given in Section 5.4.

Of particular interest in this analysis were queries 11, 16, and 20. The global relative cost of queries 11 and 16 increased slowly until δ reached approximately 100, at which point the global relative cost increased quadratically with δ. This behavior is due to the presence of alternative query plans with varying ratios between corresponding resource usage vector elements (See Section 5.5.). At low values of δ, an alternative plan that was not complementary to the initial plan became optimal. At higher δ values, another alternative plan that was complementary to the initial plan but had higher overall resource usage to begin with eventually began to dominate.

Query 20, on the other hand, was almost an order of magnitude more sensitive than the other queries to changes in resource costs. Examining `EXPLAIN` output showed that this sensitivity was due to the optimizer's choice of join method for joining the `PART` and `PARTSUPP` tables. The initial optimal plan used an index on `PARTSUPP` to perform index nested loops join. Increasing the cost of accessing this index penalized this plan, causing the optimizer to switch to using hash join. Since the cost of joining `PART` and `PARTSUPP` dominated the costs of the candidate optimal plans for this query, the choice of query plans was especially sensitive to the cost of accessing the index on `PARTSUPP`.

8.1.3 One Device Per Table and Corresponding Indexes

After obtaining the results in Section 8.1.3, we wondered how much of the worst-case global relative cost we observed was due to placing tables on separate storage devices from their indexes, as opposed to being due to placing the tables on separate devices from each other. We reasoned that the costs of accessing a table or its indexes are likely to be correlated. For example, in a distributed database, the indexes for a table are likely to be stored at the same site as the table itself. To determine the effects of separating tables and indexes, we designed a third experiment that was intermediate between the first two experiments.

In this experiment, we simulated a database system that places each table of the TPC-H schema, along with the indexes for the table, on a different storage device. That is, we varied the cost of accessing each table by the same amount as we varied that of accessing the table's indexes. For a k-table query, this analysis effectively considered the cost of accessing $k+2$ resources: one resource for each table, plus a temporary resource and a single CPU resource.

The results from this third worst-case analysis were intermediate between those of the first two experiments. Worst-case global relative cost for 15 of the queries approached a constant, following the bound in Theorem 2. Five of the other seven queries saw worst-case global relative cost rise proportional to the square of the error in estimating resource costs, approaching the bound in Theorem 1. The global relative cost of query 9 reached a constant at a δ value of approximately 500 and then began increasing quadratically when δ reached approximately 5000. As in Section 8.1.2, this behavior is indicative of a complementary alternative query plan with relatively high usage of the non-complementary resources. Query 16, which had previously exhibited quadratic behavior, saw global rel-

ative cost tail off at approximately 1000, indicating that there was an alternative query plan that accessed a particular resource 1000 times less than the initial plan.

Tying the access costs of tables and their corresponding indexes together reduced the number of complementary plans for each query and also reduced the ratios between corresponding elements of the non-complementary plans. Intuitively, many of the differences between the candidate optimal plans in Section 8.1.2 involved choices between index and table access. Making these two types of access equally expensive removed what had been an important differentiating factor between plans.

8.2 Analysis of Resource Usage Vectors

Using the technique described in Section 6.2.1, we located the candidate optimal plans for the TPC-H queries for each of the cases we tested in our first set of experiments. For the experiments described in Sections 8.1.1 and 8.1.3, we found all candidate optimal plans and their resource usage vectors. For the experiments in Section 8.1.2, we obtained candidate optimal plans and resource usage vectors for 16 of the 22 queries.

On the experimental setup that simulates placing all of the database's tables on the same device, we found no complementary candidate optimal plans for any query. For each of the queries, the maximum ratio between corresponding dimensions closely matched the worst-case global costs found in Section 8.1.1.

On the experimental setup that simulates placing each table and each table's indexes on a separate storage device, we found a large number of complementary plans. For those queries that experienced worst-case global costs of greater than 100 in Figure 6, more than half of the possible pairings of candidate optimal plans for a given query were complementary or had ratios of greater than an order of magnitude between corresponding elements of their resource usage vectors. Further analysis revealed that all of these complementary and near-complementary plans were access path complementary or temp complementary (See Section 5.6.). No pair of candidate optimal plans was table complementary.

As with the worst-case analyses, the experimental setup that simulates placing tables on different storage devices but co-locating them with their respective indexes produced results that were in between those that the first two setups produced. Correlating the costs of accessing tuples via indexes and via the tables themselves eliminated access path complementary plans (See Section 5.6.), but we still found a large number of temp complementary plans.

8.3 Discussion

When on-disk data structures all reside on the same storage device, our experiments indicate that the DB2 query optimizer will choose a plan with a cost that is within a small constant factor of the optimal plan, even if the optimizer's estimates of the relative costs of accessing storage resources are off by a large margin. We attribute this result to the non-complementary nature of the query plans that the optimizer considers with this storage configuration. In our experiment, there are essentially only three resources that can vary in cost: CPU time, number of disk seeks d_s, and number of blocks transferred d_t (See Section 3.1.). As noted in Section 8.2, every candidate optimal plan used a nonzero amount of each resource. Furthermore, these plans used the resources in relatively fixed ratios. As a result, the candidate optimal plans for the queries were highly non-complementary, and the global relative cost of the initial query plan approached the bound in Theorem 2.

As we decoupled the access costs of different on-disk data structures, the query optimizer became progressively more sensitive to the accuracy of its cost estimates. When we placed all tables and their indexes on separate disks, the optimizer found complementary candidate optimal plans (See Section 5.5.) for 18 of the 22 queries in the TPC-H benchmark. The results of our experiments demonstrate that, for the TPC-H queries and schema, a commercial query optimizer can reach the upper bounds on error in cost estimation that we show in Sections 5.4 and 5.5.

9. CONCLUSION

The proliferation of managed and shared storage resources in enterprise computing is changing the way that database systems access their hard drives. This new generation of storage systems can make the throughput and latency of virtual storage devices change dramatically and sometimes unexpectedly.

In this paper, we have articulated a vector-based framework for studying the sensitivity of relational query optimization to storage access costs. Using this framework and a set of experiments on a commercial query optimizer, we have examined the quality of the optimizer's plan choices when the optimizer has inaccurate information about the cost of accessing storage resources. Our results indicate that, when data structures such as tables, indexes, and sorted runs reside on different storage devices, a query optimizer can choose plans that are suboptimal by a significant margin if its estimates of the costs of accessing these storage devices are inaccurate. Users of databases that are laid out in this manner may achieve noticeable performance improvements by providing their query optimizers with accurate and timely information about the current status of their storage devices.

Acknowledgements

We would like to thank Guy Lohman for many technical insights, suggestions, and pointers to literature regarding query optimization; Norm Pass, Bill Tetzlaff, Ed Lassetre, and Jai Menon for getting the autonomic project started and for many discussions on the topic; Bernie Schiefer, Ping Li, Haider Rizvi, and Desmond Daly of the IBM DB2 performance team for providing the IBM TPC-H benchmark operating data; Manoj Naik, Nadav Eiron and David Chamblis for discussions on shared library implementaton and validation; Peeter Joot for providing numerous DB2 OS details; and finally Joe Hellerstein and Ashraf Aboulnaga for helpful criticism of drafts of the paper. We would also like to thank the United States Department of Defense for supporting Reiss through the NDSEG Fellowship Program.

10. REFERENCES

[1] A. Brown and D. A. Patterson. Towards avilability benchmarks: A case study of software RAID systems. In *Proceedings of the USENIX Technical Conference*, pages 263–276, 2000.

[2] S. Chaudhuri and V. Narasayya. Automating statistics management for query optimizers. *IEEE Transactions on Knowledge and Data Engineering*, 13:7–20, 2001.

[3] C.-M. Chen and N. Roussopoulos. Adaptive selectivity estimation using query feedback. In R. T. Snodgrass and M. Winslett, editors, *Proceedings of the 1994 ACM SIGMOD*

International Conference on Management of Data, Minneapolis, Minnesota, May 24-27, 1994, pages 161–172. ACM Press, 1994.

[4] F. Chu, J. Halpern, and J. Gehrke. Least expected cost query optimization: What can we expect? In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 293–302, 2002.

[5] F. Chu, J. Halpern, and P. Seshadri. Least expected cost query optimization: An exercise in utility. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 138–147, 1999.

[6] R. L. Cole and G. Graefe. Optimization of dynamic query evaluation plans. In *Proceedings of the ACM SIGMOD Conference*, pages 150–160, 1994.

[7] I. B. M. Corporation. *IBM DB2 Universal Database Command Reference*. International Business Machines Corporation, 2001. http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi.

[8] I. B. M. Corporation. Autonomic computing: IBM's perspective on the state of information technology. Technical report, 2002. http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf.

[9] I. B. M. Corporation. DB2 universal database version 8.1 for Linux, 2002.

[10] I. B. M. Corporation. TPC benchmark H full disclosure report: IBM eServer xSeries 350 using IBM DB2 Universal Database 7.2. Technical report, Transaction Processing Performance Council, 2002. http://www.tpc.org/results/FDR/tpch/ x350_100GB_16proc_FDR.pdf.

[11] T. P. P. Council. `dbgen`. http://www.tpc.org/tpch/spec/20000511.tar.z.

[12] T. P. P. Council. TPC Benchmark H. Technical report, Transaction Processing Performance Council, 2002. http://www.tpc.org/tpch/spec/tpch150.pdf.

[13] A. Deshpande and J. Hellerstein. Decoupled query optimization for federated database systems. In *ICDE*, 2002.

[14] S. Ganguly. Design and analysis of parametric query optimization algorithms. In *Proceedings of the Very Large Database Conference*, pages 228–238, 1998.

[15] S. Ganguly, W. Hasan, and R. Krishnamurthy. Query optimization for parallel execution. In *Proceedings of the ACM SIGMOD Conference*, pages 9–18, 1994.

[16] P. Gassner, G. M. Lohman, K. B. Schiefer, and Y. Wang. Query optimization in the IBM DB2 family. *Data Engineering Bulletin*, 16:4–18, 1993.

[17] G. Golub and C. van Loan. *Matrix Computations*. Johns Hopkins Press, 1983.

[18] G. Graefe. The cascades framework for query optimization. *Data Engineering Bulletin*, 18:19–29, 1995.

[19] A. Hulgeri and S. Sudarshan. Parameteric query optimization. In *Proceedings of the Very Large Database Conference*, pages 167–178, 2002.

[20] Y. E. Ioannidis. Query optimization. *ACM Computing Surveys*, 28:121–123, 1996.

[21] Y. E. Ioannidis, R. T. Ng, K. Shim, and T. K. Sellis. Parametric query processing. In *Proceedings of the Very Large Database Conference*, pages 103–114, 1992.

[22] M. Jarke and J. Koch. Query optimization in database systems. *ACM Computing Surveys*, 16:111–152, 1984.

[23] H. Lu, K.-L. Tan, and S. Dao. The fittest survives: An adaptive approach to query optimization. In *Proceedings of the Very Large Database Conference*, pages 251–262, 1995.

[24] D. A. Patterson *et al*. Recovery-Oriented Computing (ROC): Motivation, definition, techniques, and case studies. Technical Report UCB/CSD-02-1175, UC Berkeley Computer Science Department, 2002.

[25] V. G. V. Prasad. Parametric query optimization: A geometric approach. Master's thesis, Indian Institute of Technology, Kanpur, India, 1999.

[26] F. Preparata and M. Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, 1988.

[27] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, New York, NY, 1988.

[28] S. V. U. M. Rao. Parametric query optimizationi: A non-geometric approach. Master's thesis, Indian Institute of Technology, Kanpur, India, 1999.

[29] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27:17–28, 1994.

[30] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the ACM SIGMOD Conference*, pages 23–34. ACM, 1979.

[31] T. Shirai, L. Dilworth, R. Reutlinger, S. Kumar, D. Bernabe, B. Wilkins, and B. Cassells. *DB2 UDB V7.1 Performance Tuning Guide*. IBM International Technical Support Organization, December 2000.

[32] M. Steinbrunn, G. Moerkotte, and A. Kemper. Heuristic and randomized optimization for the join ordering problem. *VLDB Journal*, 6:191–208, 1997.

[33] M. Stillger and J. C. Freytag. Testing the quality of a query optimizer. *Data Engineering Bulletin*, 18:41–48, 1995.

[34] M. Stillger, G. Lohman, V. Markl, and M. Kandil. LEO — DB2's learning optimizer. In *Proceedings of the Very Large Database Conference*, pages 19–28, 2001.

[35] G. Strang. *Linear Algebra and its Applications*. Harcourt, Brace and Jovanovich, Inc., Orlando, FL, 1988.

[36] G. Weikum, A. Moenkeberg, C. Hasse, and P. Zabback. Self-tuning database technology and information services: from wishful thinking to viable engineering. In *Proceedings of the Very Large Database Conference*, pages 20–31, 2002.

[37] B. L. Worthington, G. R. Ganger, Y. N. Patt, and J. Wilkes. On-line extraction of scsi disk drive parameters. In *ACM SIGMETRICS Performance Evaluation Review*, pages 146–156, 1995.