Testing the Accuracy of Query Optimizers

Zhongxian Gu
University of California Davis
zgu@ucdavis.edu

Mohamed A. Soliman
Greenplum/EMC
mohamed.soliman@emc.com

Florian M. Waas
Greenplum/EMC
florian.waas@emc.com

ABSTRACT

The accuracy of a query optimizer is intricately connected with a database system performance and its operational cost: the more accurate the optimizer's cost model, the better the resulting execution plans. Database application programmers and other practitioners have long provided anecdotal evidence that database systems differ widely with respect to the quality of their optimizers, yet, to date no formal method is available to database users to assess or refute such claims.

In this paper, we develop a framework to quantify an optimizer's accuracy for a given workload. We make use of the fact that optimizers expose switches or hints that let users influence the plan choice and generate plans other than the default plan. Using these implements, we force the generation of multiple alternative plans for each test case, time the execution of all alternatives and rank the plans by their effective costs. We compare this ranking with the ranking of the estimated cost and compute a score for the accuracy of the optimizer.

We present initial results of an anonymized comparisons for several major commercial database systems demonstrating that there are in fact substantial differences between systems. We also suggest ways to incorporate this knowledge into the commercial development process.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems - Query Processing

General Terms

Design, Measurement, Performance

Keywords

Optimizer, Accuracy, Testing, Ranking, Plan

1. INTRODUCTION

Comparing query optimizers objectively is a difficult undertaking. Benchmarks developed for assessing the query performance of databases test the system as a whole end-to-end, e.g., TPC-H. However, to date no benchmark is available that conclusively tests a query optimizer in isolation and thus enables the comparison of optimizers of different database systems. Probably the most performance-critical element in a cost-based optimizer is the accuracy of its cost model as it determines how prone to misestimates, and thus bad plan choices, an optimizer is. The lack of such a benchmark may appear surprising given that the development of an optimizer is usually one of the most costly elements in the construction of a database system. Also, the optimizer is one of the most performance-sensitive components as differences in query plans may result in several orders of magnitude of difference in query performance–significantly more than any other contributing factor.

Being able to compare the accuracy of optimizers across different products independently is highly desirable. In particular because:

- Systems with more accurate optimizers outperform other systems. This effect is often magnified substantially by complex analytics queries.
- Inaccuracy leads to heightened effort required to make a system perform well. Often described as *query tuning*, this constitutes a significant contribution to the total cost of ownership for a system.
- During development measuring the accuracy helps guide the development process and may prevent regressions.

Objective assessment of the optimizer's accuracy can be used as a guidance to help customers make purchase decisions for one or the other database system.

There is no standard way to test an optimizer's accuracy. The cost units used in the cost model and displayed with the plan do not reflect anticipated wall clock time but are used only for comparison of alternative plans pertaining to the same input query. Comparing this cost value with the actual execution time does not permit conclusions about the accuracy of the cost model. Moreover, the optimization results are highly system-specific and therefore defy the standard testing approach where results are compared to a reference or baseline to check if the optimizer finds the "correct" solution: the optimal query plan for System A may widely differ from that for System B because of implementation differences in the query executors and the optimizers. These differences can lead to choosing radically different plans.

In this paper we develop TAQO a framework for *Testing the Accuracy of Query Optimizers* that compares the accuracy of different optimizers with regards to their plan choices. Using the basic principle of Plan Space Analysis [11], we compare optimizers based on

DBTest 2012, May 21, 2012, Scottsdale, AZ, U.S.A.

the costs they assign to all or a sample of the alternatives they consider. To force different plans, we use a set of hints or switches that optimizers expose in the query language. For example, nearly all commercial database systems have means to enable or disable certain features such as particular join implementations or aggregation techniques. For the purpose of our assessment, the actual nature of the switches does not matter so long as they can be used to generate different plans. We compare the ranking of plans with regards to actual execution cost to the ranking corresponding to their estimates and compute an accuracy score for each query of the test workload.

Roadmap. The remainder of this paper is organized as follows. In Section 2, we survey related work and discuss alternative approaches. In Section 3, we present the details of our approach and score function. We discuss the architecture of our framework in Section 4. In Section 5, we present preliminary results obtained by testing several commercial database systems. Section 6 concludes this paper and outlines possible directions for future work.

2. RELATED WORK

Testing and quality assurance have a long-standing tradition in database development. Few other general purpose software systems satisfy similarly high quality standards. And while rarely documented, all database vendors have heavily invested in quality assurance over the past decades as evidenced by publications in major conferences (e.g., [5, 1]) as well as previous editions of the DBTest workshop series [3]. However, testing optimizers in particular has received much less attention than one should expect given the high number of research publications about optimization technologies. As far as we can tell from conversations with colleagues in the industry, testing query optimizers is regarded one of the most challenging problems in optimizer development to date, see also [2, 7, 8, 9], and no generally agreed upon benchmark is available that assesses the abilities of an optimizer and enables competitive comparison.

The most common practice in optimizer testing is focusing on the default plan, that is, the plan produced by the optimizer relying on a cost model and usually highly sophisticated enumeration and plan selection algorithms. In case the default plan is suspected to be sub-optimal, it is manually inspected by a subject matter expert and opportunities for improvement are investigated. This method is often used in connection with regression tests and can be effective when sub-optimality can easily be detected by comparing the current query run time with previously measured executions. On the other hand, this method does not proactively identify problems nor does it assess the quality of an optimizer. Due to its reliance on manual inspection, this approach cannot be automated and is confined to small instances only.

Haritsa *et al.* have developed several introspective methods for optimizer testing that provide synopses for optimizer behavior in the form of functions of changing query parameters [4, 5]. Its resulting visualization helps developers to investigate suspicious optimizer behaviors. This method is generally applicable as it does not rely on specific optimizer implementations. While generally highly insightful, this method does not identify "bad" optimizer decisions *per se* but helps developers spot and subsequently focus on peculiarities of the optimizer.

Our approach is an extension of [11] where authors proposed a test methodology called *Plan Space Analysis (PSA)*. PSA provides an early warning system to detect potential plan regressions during the course of a development cycle. PSA relied on the ability to sample uniformly from the space of all plans the optimizer considers. To the best of our knowledge, only one commercial system is equipped with the necessary technology to implement this approach. In TAQO we overcome the system lock-in and provide a generally useful and applicable method for the assessment of cost-based optimizers.

3. MEASURING ACCURACY

We cannot expect the cost model to predict exactly the wall clock time for the execution of a plan, as the cost model is usually calibrated using dedicated lab equipment, and so the calibration almost never carries over to the hardware platform on which the system is eventually installed. Rather, we can define accuracy as the cost model's ability to order any two given plans correctly, i.e., the plan with the higher estimated cost will indeed run longer.

3.1 Approach

Given a query Q, let p_i and p_j be two plans in the search space considered when optimizing Q. Let p_i's estimated cost be denoted by e_i, and p_i's actual execution cost be denoted by a_i. We can declare an optimizer to be perfectly accurate if the following holds:

$$\forall i, j : e_i \leq e_j \iff a_i \leq a_j$$

Note that a perfectly accurate optimizer will always choose the optimal plan. The converse need not hold.

Using this concept we can determine the cost model's accuracy by costing and executing all plans the optimizer considers when optimizing a given query. However, this approach is impractical because of the following 2 problems:

- The search space is exponential in size; hence, evaluating the cost and timing the execution of each single plan in the search space is infeasible, in general.
- When evaluating a commercial database system, the query optimizer is a black box that does not permit generating all plans of the search space.

These limitations can be overcome by sampling plans uniformly from the search space using mechanisms like the one presented in [10]. Unfortunately, the ability of the optimizer to produce a uniform sample of the plan space cannot be generally assumed. While a uniform sample is certainly preferable, we observe that accuracy must also hold for biased samples. This observation is key to our approach. Almost all commercial optimizers provide certain controls that let users or administrators influence the plan choice for any given query. These controls are often referred to as hints, knobs, or switches in the literature. Typical examples include enabling and disabling of certain optimizations or use of specific implementations such as hash join or sort-based aggregation. Our approach exploits the existence of such controls to force a number of plan alternatives. Consider Figure 1 which depicts a sample plan space of a query in the form of a scatter plot. A plan p_i is represented as a point (a_i, e_i), where a_i and e_i are the actual cost and the estimated cost of p_i, respectively.

In practice, no optimizer achieves perfect accuracy for non-trivial queries. Hence, instead of checking whether or not an optimizer is perfectly accurate, we will develop a metric that allows us to measure accuracy in a nuanced way. In other words, given a query Q and a sample $S_Q = \{p_1, \ldots, p_n\}$ of plans from the plan space, our goal is to compute a correlation score between the ranking of plans in S_Q based on estimated costs and their ranking based on actual costs. Actual and estimated plan costs encompass quantitative information that can be utilized to better evaluate the accuracy of the optimizer. We consider the following factors for such a correlation metric:

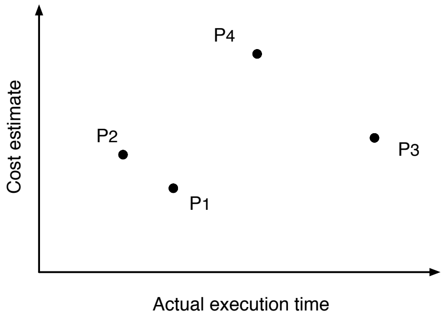

Figure 1: Scatter plot for plan alternatives

Discordance of plan pairs. The metric penalizes discordant pairs of plans, i.e., pairs whose ordering according to estimates does not reflect their actual ordering according to execution time. This can be achieved using a statistical standard tool such as Kendall's Tau or Spearman's Rank Coefficient.

Relevance of plan. The metric penalizes ranking errors involving important plans, i.e., close to the optimal plan, more than those involving insignificant plans. In our example in Figure 1, p_1 and p_2 are the most significant plans. When the optimizer makes a ranking error involving one of these important plans, the accuracy drops more than it would drop when making ranking errors involving non-important plans.

Pairwise distance. If the actual execution cost for two plans is very close, we might not be able to order them conclusively in an experiment. In this case, a ranking mistake for two plans should not be weighted the same as if they were two distant points. For example in Figure 1, incorrectly ranking plan pair (p_1, p_2) is less significant than getting pair (p_3, p_4) wrong.

3.2 Rank Correlation Metric

We now assemble these components into a proper metric. Without loss of generality, we assume $a_1 \leq a_2 \leq \ldots \leq a_m$ for a set of plans $S_Q = \{p_1, \ldots, p_n\}$. We use Kendall's Tau rank correlation as the basis for our metric [6]:

$$\tau = \sum_{i<j} \operatorname{sgn}(e_j - e_i) \tag{1}$$

To penalize the incorrect ranking of bad plans over good plans, we add the weight of plans to the metric. We define weight of a plan relative to the optimal as:

$$w_m = \frac{a_1}{a_m} \tag{2}$$

This assigns the optimal plan in S_Q weight 1. Plans with greater actual costs have lower weights. Next, in order to incorporate the notion of distance into the metric, we define the pairwise distance D_{ij} for any two plans p_i, p_j as normalized Euclidean distance:

$$d_{ij} = \sqrt{\left(\frac{a_j - a_i}{a_n - a_1}\right)^2 + \left(\frac{e_j - e_i}{\max_k(e_k) - \min_k(e_k)}\right)^2} \tag{3}$$

Plans with closer estimate and actual costs have lower pairwise distances, thus the penalties for incorrectly ranking them will be less. The final rank correlation score is computed as:

$$s = \sum_{i<j} w_i w_j d_{ij} \cdot \operatorname{sgn}(e_j - e_i) \tag{4}$$

This metric fulfills the previously outlined desiderata for a measure of accuracy. The lower the value of the metric, the higher the accuracy of the optimizer.

3.3 Normalization Across Systems

Remember that our goal is to create a portable metric that allows us to compare the accuracy of different query optimizers. As the above equations show, we already normalize both the plan weight and pairwise distance to $[0, 1]$. However, the factor not taken into account is the actual number of plans an optimizer considers: in its current shape the overall score depends on the size of the sample. Moreover, a number of good rankings can make up for egregious mistakes. Therefore, we must ensure to use the same sample size for every system.

As will become clear when we discuss the architecture of our framework in the next section, the number of plans we can get access to is not known *a priori*, i.e., we generate a candidate set S_i first for all systems we compare and select a fixed number k of plans with

$$k = \min_i(|S_i|)$$

to compute the correlation score afterward. For the selection of the k representatives, we apply the standard outlier detection algorithm of k-medoids. Like similar clustering algorithms, k-medoids is partitioning, i.e., breaks the dataset into k groups and identifies centroids in each group. k-medoids is robust to noise and outliers, thus it is more likely to preserve outliers in the dataset.

We chose to use outliers based on the following intuition: judging by the frequency and nature of support cases, the cases in which the inaccuracy of an optimizer causes a problem almost always occur in corner cases of the cost model. That is, the accuracy of a cost model is to be measured by its most significant mistakes rather than by average cases. We found the k-medoids clustering to be an effective and very robust way to choose the k inputs from a larger candidate set.

4. ARCHITECTURE

Figure 2 provides an overview over the architecture of TAQO and its components. TAQO is executed as a stand-alone tool and run against a given target database using a standard JDBC interface.

The workflow is as follows: given a workload, database connection details and a configuration of optimizer hints to use, TAQO creates a set of distinct plans for each query by choosing different combinations of optimizer hints, e.g., enable or disabling individual join implementations such as hash or nested-loops joins. For every distinct plan, TAQO determines its estimated cost from the query plan and computes the actual cost by executing the query. The data is processed using the metrics developed in the previous section and a measure for the accuracy of the database system's optimizer is returned. In addition, TAQO generates test reports including graphs to illustrate the analysis that facilitate an in-depth investigation by database implementers.

4.1 Components

In the following we discuss the individual components of TAQO in more detail.

Configuration Generator. Our approach relies on a standard feature in modern databases: optimizer hints that affect the plan choice. While the exact syntax is different for different systems, setting a specific switch α is usually accomplished either by issuing a simple command in the form '`set` $\alpha = v$', where v is a value picked from the domain of α, or by incorporating a similar construct into the query text.

The Configuration Generator computes the matrix of valid combinations of optimizer hints for a system given a configuration file that contains the names of the hints and their possible values. The

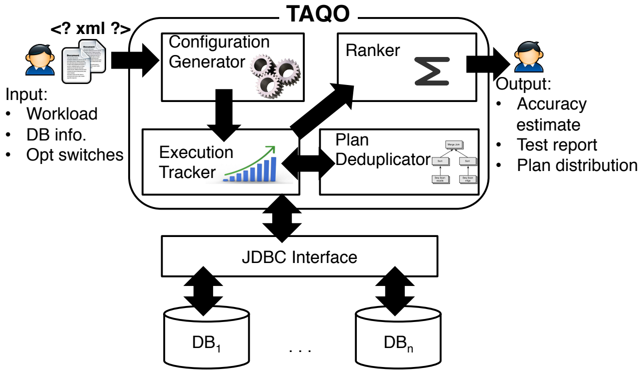

Figure 2: Architecture of TAQO

configuration is described in an XML file that also contains the corresponding syntax to be used, in addition to the hints/value combinations.

Given a set of switches $\alpha_1, \ldots, \alpha_n$ with domains $D_1, \ldots, D_n$, respectively, a possible configuration is an assignment of each α_i to a value from D_i. In general, the number of valid configurations is exponential in the number of switches. TAQO allows user to enumerate the entire configuration space exhaustively, or restrict it to a random sample of the configuration space with a given size.

Plan Deduplicator. Different combinations of switch values might lead to the same query plan, e.g., aggregation hints are ineffective if the query does not contain an aggregation operation. The Plan Deduplicator provides facilities to eliminate duplicate plans from the experiment. It is activated by registering a *plan parser*, which is responsible for extracting plan's estimated cost and plan's body from the textual output of the underlying query optimizer. By comparing the bodies of different plans, the plan deduplicator filters out identical plans. TAQO's framework includes pre-configured plan parsers for a number of commercial database systems. In addition, the plan parser API is exposed as an interface that users can implement to test other systems.

Execution Tracker. Given a query plan, the Execution Tracker obtains the plan's actual cost by physically running the plan and timing its execution until termination. Due to the system workload and cache factors, the execution time of a query plan might fluctuate over a number of values. To avoid this turbulence, the Execution Tracker runs the query plan for several times and stores the best execution time. TAQO allows users to set the number of repetitions in the input configuration file. Another concern is that some bad plans might take a very long time to terminate. The Execution Tracker stops an execution if its time exceeds a given time-out value, also specified in the input configuration file. The Execution Tracker marks such plans as 'time-out plans' and records their number as part of the final report.

Ranker. The Ranker is responsible for computing a correlation score between the rankings of plans based on actual and estimated costs. By default, TAQO uses a Kendall Tau-based rank correlation metric, as discussed in Section 3.2. To allow users to experiment with other metrics, the score computation API is exposed, and it can to be implemented as needed. In addition to the rank correlation metric, graphs may be intuitive to explain optimizer's behavior. The plan distribution plot (e.g., Figure 6) is generated by the Ranker for each tested query as part of the final report.

4.2 Portability and Extensibility

TAQO was designed with configurability in mind. In particular, two areas of configurability have been critical to this study:

Query	Opt-A		Opt-B		Opt-C		Opt-D	
	OK	T/O	OK	T/O	OK	T/O	OK	T/O
Q1	1	5	1	1	31	2	6	10
Q2	73	6	161	40	236	120	87	14
Q3	24	8	19	37	27	67	49	15
Q4	26	2	2	2	24	64	49	15
Q5	34	8	22	133	32	57	140	36
Q6	3	0	1	1	16	0	4	0
Q7	34	6	45	105	36	57	126	26
Q8	20	18	25	163	18	44	65	32
Q9	4	46	14	144	3	50	96	23
Q10	34	6	65	98	29	64	56	9
Q11	36	2	173	23	1	0	33	12
Q12	15	3	8	13	60	31	80	24
Q13	19	13	14	2	28	6	60	24
Q14	13	1	11	4	29	14	27	4
Q15	6	0	2	0	1	0	69	8
Q16	54	6	62	9	1	0	24	23
Q17	6	18	10	14	1	0	24	23
Q18	31	19	28	84	1	0	53	13
Q19	12	2	11	13	86	1	26	6
Q20	83	5	61	87	178	58	94	49
Q21	18	38	75	59	98	70	98	72
Q22	12	8	28	4	1	0	37	12

Table 1: Number of plans generated for TPC-H queries

- Portability: TAQO uses a JDBC interface to hide database interaction details. To test an optimizer, user need only provide a JDBC driver, configure hints, and provide a plan parser.
- Extensibility: TAQO exposes the API used to compute accuracy measure to the user. This allows experimenting with different accuracy measurement techniques, which is important to support different use cases.

We demonstrate our framework's versatility in Section 5 where we report results on using TAQO to evaluate four different commercial query optimizers.

5. EVALUATION

TAQO is part of Greenplum's development and test cycles. We present experiments we conducted using several commercial database systems to demonstrate the effectiveness of TAQO. In the following discussion, we anonymized these systems and refer to them simply as Opt-A, Opt-B, Opt-C, and Opt-D. To preserve anonymity, we cannot disclose the number nor the names of the optimizer hints used as this would allow users familiar with any of the systems to infer their true identities.

5.1 Setup

We conducted our experiments on a server equipped with a 2.4 GHz Quad CPU and 8 GB of memory. As test dataset we chose TPC-H at scale factor 1.0, i.e., that is 1 GB of data size. In preliminary tests, we found the size of the database to be of little relevance and, hence, chose a scale factor that allowed us to run large number of experiments efficiently.

For each experiment TAQO executes a query 5 times and retains the best execution time after eliminating the top result to allow for cache warming. Queries running longer than 30 seconds are declared 'timed out'. This allows attempting the execution of even highly costed plans to check whether their execution time is actually short in reality. The maximum number of configuration considered is capped at 500 and configurations are chosen with uniform probability.

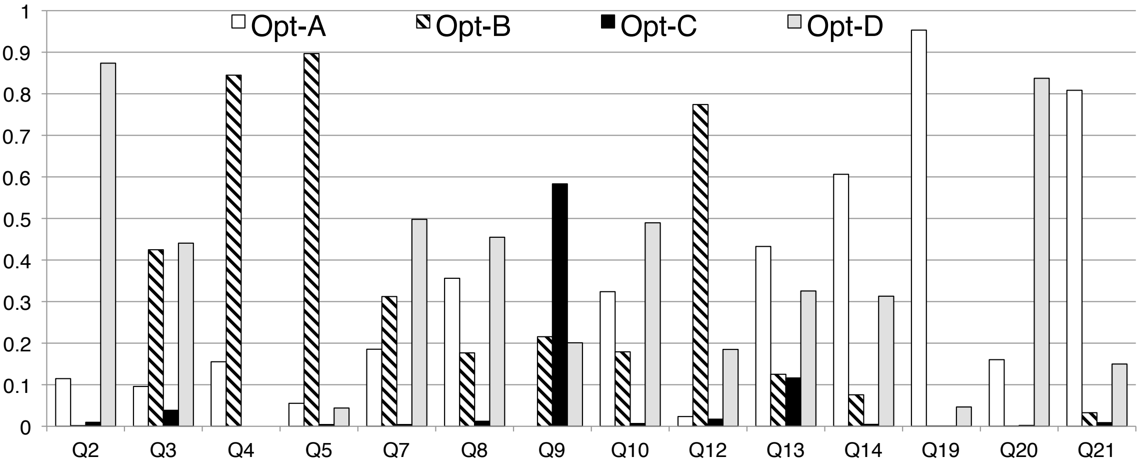

Figure 3: Correlation scores for different TPC-H queries. Low score indicates high accuracy

5.2 Sampling the Plan Space

Table 1 shows the number of generated plan alternatives for each query. The 'OK' column has the number of plans that finished execution regularly, the 'T/O' column has the number of timed-out plans. In case of time-out, we use the time-out threshold as the actual execution cost. As discussed in Section 3.2, we use a k-medoids algorithm to cluster the results and chose k to be the minimum number of plans any of the test subjects produces for this query but no less than 5. This enables a fair comparison between the different optimizers even in the cases where one of them produces only a single plan.

The data in Table 1 provides several insights that underline the validity of our approach:

- Our methodology is able to generate non-trivial numbers of plan alternatives in general across all systems and across all queries; (we discuss exceptions below).
- The number of plans found reflects largely the queries' complexity, e.g., $Q1$ is of trivial nature, while $Q2$ is generally considered more complex.
- The choice of optimizer hints in each system provides a significant number of plans that do not time out and hence can be considered reasonable plan alternatives given the data size.

The data also identifies several peculiarities, especially regarding Opt-C: while we used more switches on it than on any other optimizer we were not able to generate more than 1 plan alternative for a number of queries: $Q11$, $Q15$, $Q16$, $Q17$, 18, and $Q22$. We have consulted the development team of Opt-C and deployed additional switches according to their recommendation but were still not able to force more than 1 plan in these situations.

Opt-B generates a significant number of plans that do not finish within the timeout limit. This indicates that its optimizer hints are more heavy-handed than those of other systems and are likely to be more difficult to tune in practice.

5.3 Optimizers Comparison

Next, we compare the four systems based on their rank correlation scores. Figure 3 shows the scores for all queries and for each system per query. A lower score indicates higher accuracy according to our methodology. At first glimpse, the graph reveals that no system outperforms the others on all queries, or, conversely, for each optimizer there exists a query where its accuracy is the lowest among all contenders.

Figure 4 shows the relative rank correlation scores of the four optimizers, defined as $s(O_i)/\sum_i s(O_i)$. For each query shown in Figure 4, the wider the area an an optimizer occupies, the less accurate the optimizer is. We eliminate from the comparison queries where at least one of the optimizers produced only one plan. For queries where it produces more than one plan alternative, Opt-C outperforms the other systems with the exception of $Q9$.

In addition to the overall correlation score, we also examined the *best plan found* according to [11], i.e., the default plan an optimizer would generate in absence of any hints. Surprisingly, in a large number of cases, the default plan is not the optimal plan overall: for several queries, forcing different plans revealed better solutions than the optimizer found on its own.

Figure 5 sums up this insight and contrasts it with overall accuracy of each optimizer. We computed the average correlation score for each optimizer over all queries and plot optimizer's accuracy, defined as $1 - avg(s)$, against optimality, defined as the number of times the default plan returned by an optimizer is the actual best plan in our sample. The size of a bubble in the plot indicates the average number of plans generated by each optimizer.

The plot emphasizes our earlier observations. Opt-C scores highest on both accuracy and optimality. Among the 22 TPC-H queries, the default plan returned by Opt-C is the optimal plan in 11 queries.

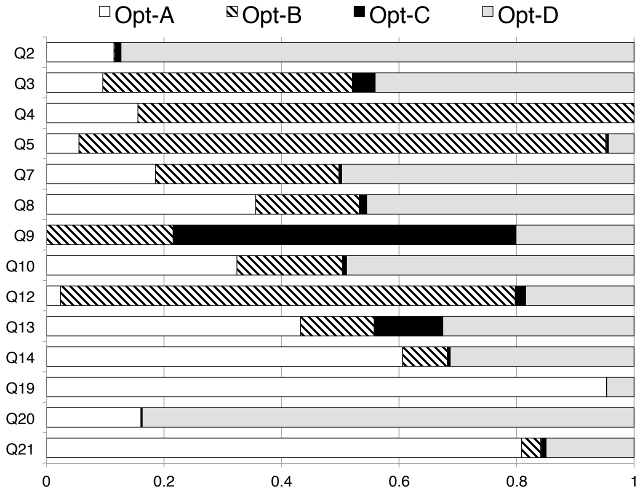

Figure 4: Optimizers relative scores. Low score indicates high accuracy

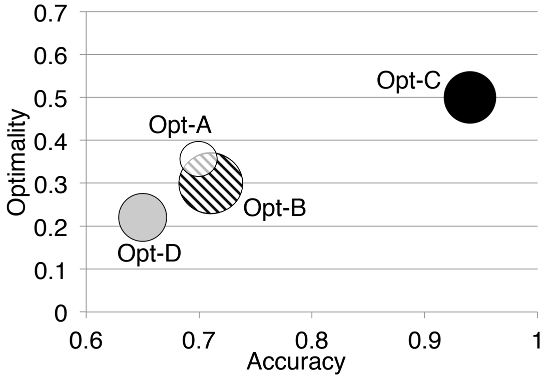

Figure 5: Accuracy vs. optimality

Opt-A and Opt-B score significantly lower in both accuracy and optimality but are very close to each other. Opt-A performs slightly better in terms of optimality. The average number of plans for Opt-A is the lowest with 35 generated plans, while Opt-B generated 85 plans, Opt-C generated 74 plans, and Opt-D generated 81 plans.

5.4 In-depth Analysis

To demonstrate how the quantitative score can provide better insight into the optimizer's accuracy, we used TAQO to generate plan distribution plots. Figure 6 shows the plan distribution plots of the four optimizers for TPC-H $Q14$. We show for each optimizer the distribution of generated plans, and the centroids—shown as the inner points—picked by the clustering algorithm during the normalization step. When an optimizer consistently ranks plan alternatives correctly, the plan distribution plot shows a monotonic trend. That is, as estimated plan cost increases, the actual cost also increases. Opt-C clearly demonstrates this trend. In contrast, Opt-D incorrectly assigns a relatively high estimated cost to the actual best plan and low estimated costs to slow plans, shown at the bottom-right corner of the plot. These ranking errors contribute to the low accuracy of Opt-D.

This type of analysis helps detecting problematic queries that pose challenges to the cost model. Also, engineers can use the rank correlation as a measure to judge whether a code-level change made to an optimizer may cause noticeable plan regressions.

TAQO was primarily developed to provide engineers with a generic, fully automatic, yet technology agnostic way to test the accuracy and optimality of the product. It is becoming a vital tool in the development of Greenplum's query optimizer.

6. CONCLUSION AND FUTURE WORK

Testing and quality assessment of query optimizers are critical elements for both the development of database systems as well as the advancement of research in the field of query optimization. Among several important design goals for an optimizer, the accuracy of the cost model and the optimizer's ability to distinguish a superior from an inferior plan are crucial to its success.

In this paper we presented TAQO, a general and portable framework for testing the accuracy of query optimizers. TAQO leverages system-specific hints or tuning mechanisms in order to force the optimizer to choose different plans. We define a metric over the plans found by comparing the estimated with the actual cost and measure the correlation of plans for queries and workloads. This metric is rather intuitive as it summarizes how likely a given optimizer is making bad optimization decisions. The results we presented in this paper by comparing major database systems suggest the abilities of commercial query optimizers differ widely. To the best of our knowledge, this is the first study of its kind.

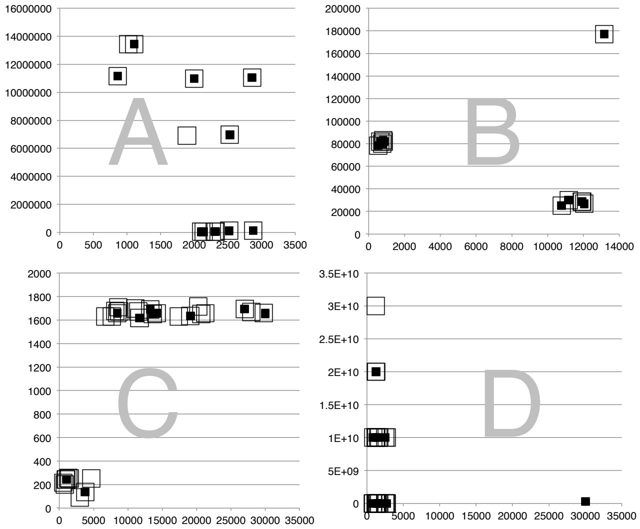

Figure 6: Plan distribution plots for $Q14$. Actual execution time is on the x-axis, and estimated cost is on the y-axis

Acknowledgements

The authors would like to thank Srinivasa Meka for his technical support for the different database systems we tested, as well as the members of the Query Processing team at Greenplum for valuable feedback and support.

7. REFERENCES

[1] S. Chaudhuri, L. Giakoumakis, V. Narasayya, and R. Ramamurthy. Rule Profiling for Query Optimizers and their Implications. In *Proc. ICDE*, 2010.

[2] L. Giakoumakis and C. Galindo-Legaria. Testing SQL Server's Query Optimizer: Challenges, Techniques and Experiences. *IEEE Data Eng. Bulletin*, 31(1), 2008.

[3] G. Graefe and K. Salem, editors. *Proc. of the Fourth International Workshop on Testing Database Systems, DBTest 2011, Athens, Greece*. ACM, 2011.

[4] D. Harish, P. N. Darera, and J. Haritsa. Identifying robust plans through plan diagram reduction. *Proc. of the VLDB End. (PVLDB)*, 1(1):1124–1140, 2008.

[5] J. Haritsa. The Picasso Database Query Optimizer Visualizer. *Proc. of the VLDB End. (PVLDB)*, 3(2):1517–1520, 2010.

[6] M. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[7] L. Mackert and G. Lohman. R* optimizer validation and performance evaluation for distributed queries. In *VLDB*, 1986.

[8] L. Mackert and G. Lohman. R* optimizer validation and performance evaluation for local queries. In *SIGMOD*, 1986.

[9] M. Stillger and J.-C. Freytag. Testing the Quality of a Query Optimizer. *IEEE Data Eng. Bulletin*, 18(3):41–48, 1995.

[10] F. Waas and C. A. Galindo-Legaria. Counting, Enumerating, and Sampling of Execution Plans in a Cost-Based Query Optimizer. In *Proc. ACM SIGMOD*, 2000.

[11] F. Waas, L. Giakoumakis, and S. Zhang. Plan Space Entropy: An Early Warning System to Detect Plan Regressions in Cost-based Optimizers. In *Proc. DBTest*, 2011.