Testing Cardinality Estimation Models in SQL Server

Campbell Fraser, Leo Giakoumakis, Vikas Hamine, Katherine F. Moore-Smith
Microsoft Corporation
One Microsoft Way
Redmond WA, 98052, USA
{cfraser, leogia, vhamine, kmoore}@microsoft.com

ABSTRACT
Reliable query optimization greatly depends on accurate Cardinality Estimation (CE), which is inherently inexact as it relies on statistical information. In commercial database systems, cardinality estimation models are sophisticated components that over years of development can become very complex. The code that implements cardinality estimation models, like most complex software systems that handle a large space of possible inputs and conditions, can deviate from its original architecture and design points over time. Hence, it is often necessary to refactor and redesign the entire system to accommodate new inputs and conditions, and also to reflect existing ones in a more intentional way. In this paper, we describe such an exercise: the replacement and validation of a new cardinality estimation model in Microsoft SQL Server. We describe the motivation behind this change, and provide a high level sketch of the empirical methods used to ensure that the new cardinality estimation model satisfies its goals while minimizing the potential risk of plan regressions for existing customers.

Categories and Subject Descriptors
H.2.4 [**Database Management**]: System – Query processing

General Terms
Reliability, Verification.

Keywords
Experimental methodology, software quality, query optimization, cardinality estimation

1. INTRODUCTION
Query optimizers rely on accurate cardinality estimation (CE) models to produce reliable and efficient execution plans. Typically, cardinality estimation models are based on a set of basic assumptions. Some of these assumptions originate from the underlying histogram structures on which they rely, e.g. uniform frequency, continuous values [3], etc. While such assumptions may work reasonably well for most inputs, i.e. query shapes, data types, and data characteristics, there are special cases and biases that are later accommodated with extensions. Such extensions aim to enhance the cardinality estimation model to accommodate the breadth of the SQL language features, and to better model real-world data characteristics. For example, columns with ever-increasing values need to be treated specially, since insertions always land at the end of the table and query predicates on such columns tend to filter for the most recently inserted data [4]. Cardinality estimation models may have a bias towards overestimating cardinality in certain cases; for example, in joins. Since allocating enough memory up-front for some join algorithms, like hash join, is more efficient than dynamically extending the initial memory allocation or, depending on the query execution policies such as spilling the hash table to disk.

As query optimizers are tested under different query workloads and data characteristics, the development team makes extensions and variations to the cardinality estimation component. Some of these extensions originate from customer problems that may be considered as special cases/one-offs at the time. Such extensions and adjustments to the cardinality estimation model are often conditionally enabled by a switch or a version/compatibility feature in order to minimize the risk of regressions to other customers. After multiple releases of the DBMS system, it's not unlikely for a cardinality estimation model to contain many such adjustments and extensions. After multiple years of cumulative development and support activity, like with any other complex software system, it's worth considering a refactoring or a complete redesign of the cardinality estimation component for the purposes of code hygiene and maintainability. Additionally, refactoring may be necessary in order to promote some of the special cases, added after the original release, to first-class functions of the cardinality estimation model.

Although such efforts aim to produce a more sophisticated, more accurate, more complete, and more architecturally sound cardinality estimation component, the statistical nature of the algorithms involved means that it is very likely that legitimate improvements will cause unintended and unpredictable execution plan regressions. Therefore, a big part of the quality assurance (QA) process for such plan-affecting changes involves measuring the potential regression risk to existing applications. It is a common practice for QA teams to use a set of customer workloads and synthetic benchmarks as quality criteria for such changes [1][5][7]. Based on the results of those tests, the cardinality estimation model is further tuned to ensure a high degree of backward compatibility with its previous version.

This paper describes the process of refactoring and validating the CE model of Microsoft SQL Server after its 2012 release. The new CE model is planned as a future service release of the Microsoft SQL Azure service. A complete description of the motivation for the refactoring exercise, and the specific design on the new CE model, are beyond the scope of this paper: it could be the topic of a separate future paper. In this paper, we will focus on sharing our experience with the quality assurance process that was used to validate our new cardinality estimation model.

The rest of the paper is organized as follows: in section 2, we will briefly touch on the motivation for redesigning the cardinality estimation model. In section 3, we will discuss the main elements of the quality assurance process, and show some snapshots of the quality criteria used during product stabilization. In section 4, we will draw comparisons with related published work. Finally, we will conclude in section 5.

2. MOTIVATION FOR A NEW CARDINALITY ESTIMATION MODEL

The cardinality estimation component of the SQL Server Query Optimizer was initially developed for SQL Server 7.0, which was released in December, 1998. Over multiple subsequent releases of the product, ad-hoc extensions to the model of CE, special case fixes and extensions to provide cardinality estimation for new T-SQL features, made the CE component very hard to debug, predict, or understand.

A fundamental limitation of the original architecture, which made the provision of non-simple extensions difficult to achieve, resulted in some extensions being made in ways that made their application inconsistent. Specifically, the original architecture did not effectively separate the different tasks of: (i) deciding how to compute a particular estimate (gathering and evaluating the available statistical and constraint information) and (ii) actually computing the estimate.

As the user base of a database system grows, it becomes increasingly difficult to make changes to the Query Optimizer without causing negative performance consequences to some existing users. The final choice of a query execution plan is dependent upon potentially thousands of estimations, and improving a subset of those estimations can, paradoxically, lead to a less efficient query plan being chosen. Prior to improving that subset, we say that *“two (or more!) wrongs made a right”*. To mitigate this, some fixes and improvements in the CE component were not active in the product by default: they required a switch to be turned on in order to activate them. Thus, users who required an improvement could turn it on and users who did not would leave it off and not face the inherent risk of a change in the model.

A cardinality estimation model uses statistical and constraint information about user data to compute its estimates. Where there is a gap in knowledge, assumptions are made. The closer those assumptions are to reality, the more accurate the estimations will be. Most query optimizers use the following simplifying assumptions (or variants thereof) for cardinality estimation [8][11]:

1. **Uniformity:** In the absence of contrary statistical information, values in a column are distributed uniformly. Values within a histogram step are assumed to be uniformly distributed.
2. **Independence:** Within each relation, values from distinct attributes/columns are independent, absent statistical information to the contrary.
3. **Containment**: When two values “might be the same”, they are assumed to be the same. This means that when joining two tables, values from the table with fewer distinct join column values are typically assumed to be contained in the other table. When filtering against a constant (e.g. where T.c = 100), the constant is assumed to be present within the table, if statistical information and constraints do not rule it out.

Experience with hundreds of customer queries has taught us that, in practice, these assumptions are frequently incorrect. However, the original architecture of the component made changing any of the fundamental assumptions in a directed way impossible. It was not obvious how to best adjust them anyway. Thus, the goals of the rewrite were:

- Separate the tasks of deciding how to compute an estimate and actually performing the computation to make future extensions easier to complete in an architecturally sound fashion.
- Allow for the fundamental assumptions of the CE model to be easily changed so that they could be assessed with real user workloads.
- Incorporate prior extensions into the model as first class citizens so that their behavior is more predictable.

3. QUALITY ASSURANCE PROCESS

Any change made within the optimizer that impacts plan selection runs the risk of regressing existing customer applications. Since in this case we are discussing a complete overhaul of a central component of the query optimizer, such regressions are a certainty, no matter how elaborate or thorough we are in the quality assurance process. The impact of regressions to existing customer applications can be mitigated by a versioning mechanism that allows customers to choose between the old and new CE models. This was one of the key decision and design points during the project. While having a versioning mechanism certainly allows customers to manage and control the risk of plan regressions, it does not reduce the scope of in-house testing. The validation process of the new cardinality estimation model will be the main focus of this section. We will describe the high-level test strategy, the test challenges we faced, and how we tackled them.

At a high level our quality assurance process consisted of the following two elements:

1. **Customer Workloads and Benchmarks:** Measure the risk of regressing existing customer applications by evaluating the new model against a variety of customer workloads and standard benchmarks. This will ensure a high degree of effectiveness and backward compatibility.
2. **Synthetic Functional Testing**: Ensure that the implementation of the new CE model is functionally correct. It needs to handle the entire spectrum of inputs and conditions and behave as designed.

3.1 Customer Workloads and Benchmarks

One of the main success criteria for this work was that the new CE model should be on par or better, when compared to the old model, against a wide set of workloads covering a variety of scenarios and product features. We selected a diverse set of customer workloads across different segments (ranging from OLTP to DSS) of the database application space as shown in Figure 1 [1]. The combination of in-house customer queries/tests, standard TPC benchmarks, and real customer workloads used for these comparisons amounted to more than 30 different test workloads. The comparison was based on metrics such as query response time, query compile-time, and compile-time memory footprint. This process, which we describe below, was slow and iterative, but in our opinion, valuable and necessary.

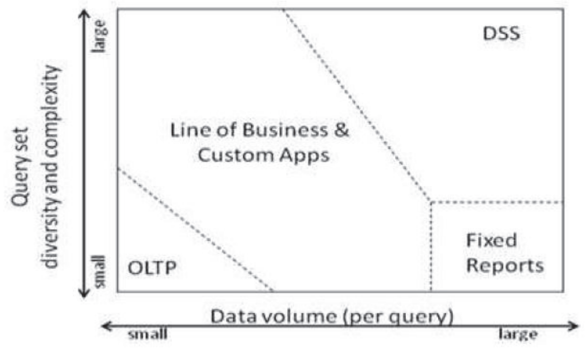

Figure 1: A categorization of database application space.

As noted earlier, one of the primary purposes of the new CE model was to incorporate one-off additions that were made over the years, while also re-establishing some of the core assumptions such us data uniformity, independence, and containment. Its performance, in terms of producing accurate cardinality estimates, is a function of how well queries and data sets satisfy those fundamental assumptions. At the end of the first iteration of development, test results against in-house customer workloads and standard benchmarks indicated several issues. We found that a large number of workloads showed performance regressions between 20-50%, with severe regressions for some individual queries. We present some of the results in Figure 2 which shows the performance of new CE using the old CE as a baseline. We used end-to-end query response times averaged over multiple runs as the comparison metric.

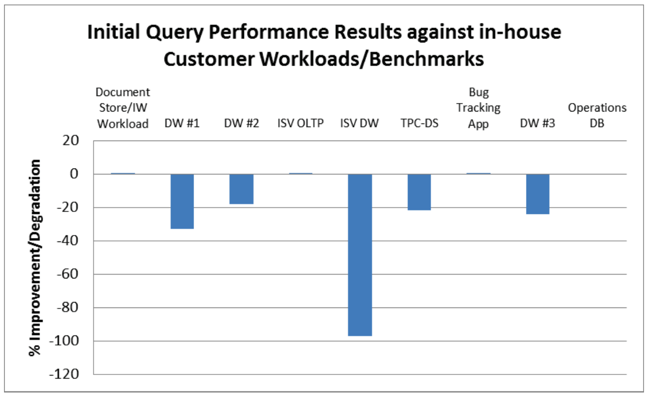

Figure 2: Initial query performance (response times) comparison between old and new CE models

The results motivated a phase of elaborate investigations and root cause analysis. The analysis showed that some of the heuristics compensating for data outside of our core assumptions needed to be removed or relaxed. Instead of simply re-implementing all the special cases of the old CE model in the new one, we used a more formalized concept in the new design; we will refer to it as: *model variations.* The variations are based on a mechanism of pluggable heuristics and biases for the various core cases of CE, e.g. CE for joins. The new design of the CE model allowed us to conditionally enable and disable those variations on demand. This was very useful for testing. Describing the design that allowed the capability described above and the complete set of CE model variations that we used is beyond the scope of this paper. Instead, for the purposes of providing the necessary context to the reader, we will describe only a few of these as examples below. A more comprehensive description could be the topic of a future paper.

During the development cycle, there was a point where we had 11 model variations being tested simultaneously, both alone and in promising combinations. The model variations that we considered systematically relaxed the "purity" of the new CE model in ways that would fit the characteristics of real data and satisfy some of the workloads and benchmarks being tracked. For instance, an entirely pure model would assume independence between predicates on a single table. This meant that selectivity of the conjunction of filter predicates on a single table would be the product of individual selectivities. We experimented with a model change that would assume complete correlation between the filter predicates and would choose the minimum of the individual selectivities as the selectivity of their conjunction. This model change essentially relaxed the independence assumption of the "pure" model and could be potentially used with other compatible model variations. In addition to the two extremes: independence of or complete correlation between filter predicates, we also experimented with variations that explored the spectrum between independence and complete correlation. The Overpopulated Primary Key and Simple Join heuristics described below are additional examples of model variations we experimented with for estimating join cardinalities. Please note that detailed descriptions and math behind these model variations is beyond the scope of this paper.

Overpopulated Primary Key (OPK): The OPK heuristic boosts the join estimate of the "pure" (new CE) model in some circumstances. The heuristic is applied only in scenarios where the number of distinct values for the joined column(s) in one table (the Primary Key table) is larger than those in the other table (the Foreign Key table). The rationale is that when the primary key values are filtered prior to the join, the filter is typically biased towards removing those values that are not present on the foreign key side. The filtering of the primary key side and the join predicate are therefore not assumed to be independent. This commonly occurs with date dimensions that span time periods larger than the fact table.

Simple Join: Simple join ignores histograms from the two relations, and employs a containment model that simply assumes all distinct values from one child are contained in the other (originally introduced in [8]). This simplistic strategy typically sacrifices estimation quality for faster optimization time, providing a balance that better suits some workloads.

Through careful exploration and combining of model variations, we settled on a set of modifications to the pure model that surpassed the performance of execution plans produced by the old CE model. Query performance was better on most benchmark tests. A snapshot of results against in-house workloads and benchmarks on a recent product build is shown in Figure 3. As can be seen from Figure 3, most DW workloads with the exception of the "ISV DW" workload show minor to significant improvements in query response times. Other workloads remain at par with the old CE model. We are currently experimenting with other variations aimed at fixing the performance gap for the "ISV DW" workload. This gap manifests as a result of specific data skew that does not fit well with any model variation tested so far.

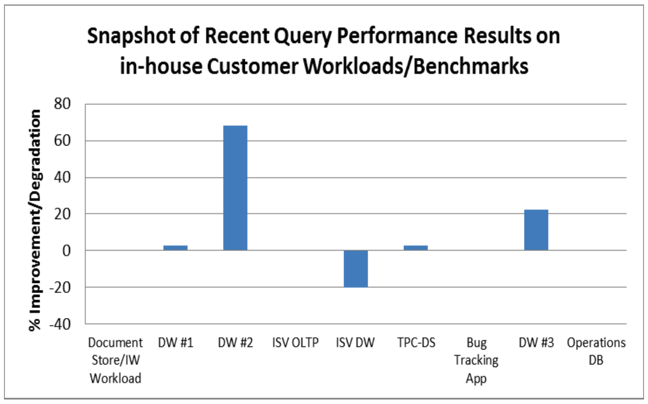

Figure 3: Snapshot of query performance (response times) comparison between old and new CE on a recent build.

3.2 Synthetic Functional Testing

The second part of our quality assurance process was to ensure functional correctness of the new model implementation(s). We mainly achieved this through:

1. Large-scale stochastic testing, where we used random query generators to tackle the large input space [2].
2. Extensions to existing estimation quality and stability tests originally developed for validating the old CE model. We modified these tests to validate the behavior of new CE model. They use simple queries and synthetic data fitting the model assumptions, and make estimation accuracy verifications.
3. Targeted tests that validate the behavior of the new model with respect to the underlying assumptions it made – data uniformity, independence, and containment – using synthetic data and queries.

The remainder of this section will focus on the last of these three test approaches.

Section 3.1 described how candidate models were evaluated against a standard set of benchmarks and real customer database queries for performance gains/regressions. While standard benchmarks/workloads are good measures of success, and a critical part of the quality assurance process, they could lead to solutions that are tailored to fit these workloads. The result could be models that are "over-fitted" to the available test workloads [1]. Hence, in addition to the standard benchmarks and customer workloads, there is a clear need to evaluate competing models in a more controlled fashion, and to characterize their behavior with respect to the simplifying assumptions they make about the underlying data. The synthetic experiments take these assumptions into account and evaluate competing models against data that conform to the model assumptions to varying degrees. The rest of this section describes our methodology for these synthetic experiments and provides examples of how we applied it.

The methodology we used had the following important dimensions:

1. **Experiment/Scenario definition:** Defines what scenario and model(s) of interest will be evaluated as part of the experiment.
2. **Engine for generating space of queries:** This piece of the framework generates a set of meaningful queries for the current scenario. All the models in the experiment are evaluated against the same set of queries.
3. **Engine for generating a set of database instances:** This piece of the framework allows us to generate "interesting" database instances on which the models will be evaluated with the queries. The database instances would conform to the core model assumptions to varying degrees.
4. **Evaluation Methodology:** For each model, ***M,*** of interest in our experiment, on each database instance, ***D,*** (generated by #3 above), run each query in the query set, ***Q,*** (generated by #2) against ***M***, and store the estimated and actual cardinalities for all queries in set ***Q*** on database ***D*** for model ***M***.

As a concrete example, we will describe a simple application of this methodology and explain how we used it to characterize behaviors of model variations of interest. Specifically, our focus will be on applying this methodology towards cardinality estimation for joins for model variations that depend on the containment assumption (other experiments targeting uniformity and independence assumptions can be formulated along similar lines). The remainder of this section provides details of a simple experiment formulated around the realistic scenario of two-table, one-to-many join. The experiment will evaluate four models of interest – old CE, pure new CE (without any variations), OPK, and Simple Join (refer section 3.1 for brief descriptions on OPK and Simple Join).

Experimental Setup: Each database instance in our experiment consists of two tables, let's call them **Dim** and **Fact**. Table **Dim** has two columns dimId and dimValue, where both dimId and dimValue are integer columns having unique distributions. The **Fact** table has only one column refDimId which is populated using values from dimId in **Dim** table. The **Dim** table contains 1000 rows and the **Fact** table is populated with 100,000 rows such that the refDimId column in the **Fact** table has a uniform distribution. We create single column statistics (histograms) using full scans on each of the three columns – Dim.dimId, Dim.dimValue, and Fact.refDimId.

The queries of interest in our query set ***Q*** contain 1000 queries of the following form:

```
SELECT Count(*)
FROM   Dim, Fact
WHERE  Dim.dimId = Fact.refDimId and
       Dim.dimValue <= {constant}
```

The queries in ***Q*** correspond to the 1000 distinct values in dimValue column. We evaluate the models/variations of interest using queries in ***Q*** against 10 different database instances. The database instances differ in the degree of containment between the **Dim** and **Fact** tables. Specifically, we vary the degree of containment between the **Dim** and **Fact** tables from 10% to 100% in increments of 10%. When the degree of containment is 10%, the **Fact** table is populated with 100,000 rows using only 10% of the values from the dimId column. In this instance of the database, each distinct value in the refDimId column of the **Fact** table is expected to have a frequency of 1000, since the number of distinct values in the **Fact** table is expected to be 100. When the degree of containment is 100%, the **Fact** table will be populated with 100,000 rows using all values from the dimId column in which case, the average frequency of the refDimID column will be 100.

Using this experimental setup we extract cardinalities estimated by each model against queries in ***Q***, and plot the average of the estimated cardinalities for queries in ***Q*** across all instances of databases. Such a plot is presented in Figure 4.

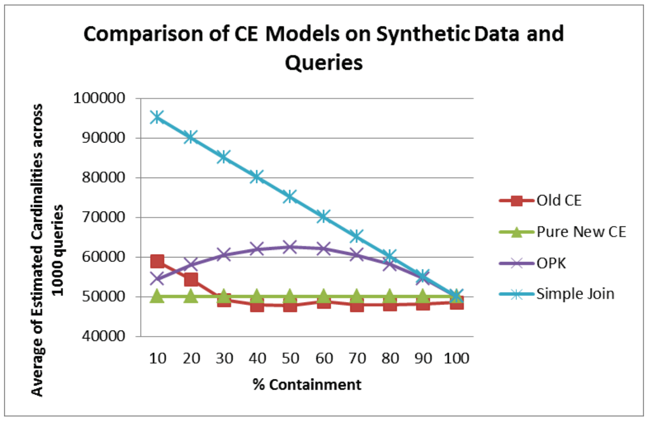

Figure 4: Characterizing estimations made by four estimation models.

The following observations can be made from Figure 4:

- The simple join model produces larger estimates when the degree of containment between **Dim** and **Fact** tables is small. This is expected given the experimental setup and the model definition. Note that the average frequency of values in the refDimId column decreases as the % containment increases. That combined with the definition of Simple Join model explain the observed behavior.
- The OPK model boosts join estimates obtained from the new CE model as expected and converges to the "pure" CE model's estimates when all dimId values from **Dim** table are also present in the **Fact** table – 100% containment.
- The less regular pattern for the old CE reflects its inconsistency (which motivated this work in the first place).

As can be seen from the results, the new models (pure new CE, OPK, and Simple Join) are well-behaved: their behavior can be rationalized in the context of such a simple experimental setting. We used these experiments in conjunction with testing against standard workloads and benchmarks (as described in Section 3.1) to ensure that the variations we introduce in the system are well-behaved, at least in simple settings. This process was inter-dependent; we analyzed regressions found from testing against in-house customer queries, brainstormed and introduced model variations for fixing the regressions, and then devised simple experiments on synthetic data and queries that mimicked the essence of the customer scenario for which the model variations were introduced. The experimental methodology used here was simple enough for us to later "freeze" the results as actual functional tests that verified behavior of the model variations that made up the final CE model.

Note that experimental results from Figure 4 highlight estimated cardinalities for the model variations of interest and completely disregard absolute/percentage estimation errors for these variations. The reason being: the purpose of these experiments was to characterize and validate model behaviors, but not to select and settle on a "winning" model with lowest overall estimation error. Estimation errors are a function of actual cardinalities which in turn are a function of the underlying data distribution, and hence evaluating model variations based on estimation errors on a *specific* synthetic data distribution could be very misleading.

It is important to note that the behavior of an estimation model is dependent only on the model assumptions and its statistical inputs (single column histograms in our case). By varying the underlying data in a manner that does not change statistical inputs to our models, we can make any model look arbitrarily good or bad with respect to accuracy of its estimates. To illustrate this point further, we evaluate our models of interest against three different data distributions (on the same experimental setup), obtained by shuffling values in dimValue column in a way that impacts pairing of the <dimId, dimValue> tuples. Note that shuffling values within the dimValue column would not impact its histogram. Figure 5 plots the average of *actual cardinalities* across 1000 queries from **Q**, against three such data distributions. Specifically, ActualCards1 corresponds to a distribution when both dimId and dimValue columns contain values in the range [1, 1000], but each tuple in the **Dim** table satisfies the predicate *dimId = dimValue*. In other words, both the dimId and dimValue columns take on the same values in each tuple: the two columns have a strong positive correlation. The **Fact** table in each of these three distributions is populated using top N% ordered values from dimId column, where N corresponds to % containment. As can be seen from Figure 4 and Figure 5, the synthetic data distribution corresponding to ActualCards1 favors the simple join model (and the OPK model to a lesser extent). This is because data filtered out by the filter predicates in the test queries in set ***Q*** are not likely to be in the **Fact** table, a scenario well captured in the assumptions made by Simple Join and OPK models.

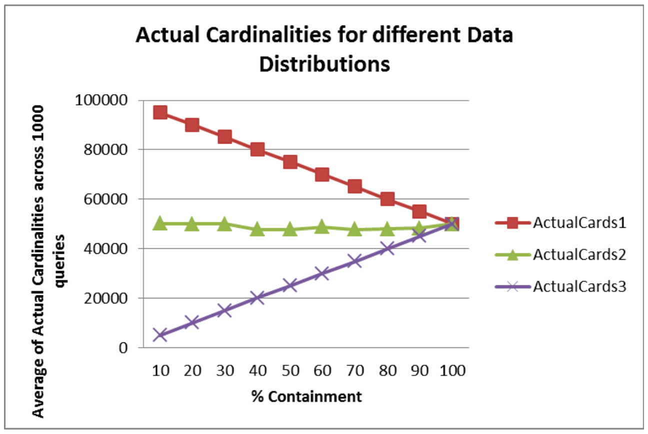

Figure 5: Actual Cardinalities for different Data Correlations between dimId and dimValue columns.

The ActualCards2 curve corresponds to a different distribution, where we pick a random ordering of values in the dimValue column with respect to how they appear together with values from the dimId column, keeping everything else the same. The dimId and dimValue columns would have a weak correlation or in other words they can be considered independently distributed. This distribution favors the Pure New CE and Old CE models. The ActualCards3 curve corresponds to a distribution where we reverse the ordering of values in dimValue column so that dimId and dimValue columns have a strong negative correlation. Each tuple in **Dim** table satisfies the predicate *dimId = (dimValue %*

1000) + 1. This distribution does not favor any of the four CE models (but thankfully it does appear pathological in nature).

It can be seen that Simple Join, for example, looks flawless against the distribution corresponding to ActualCards1, but can be made to look arbitrarily bad for other data distributions. These results illustrate the potential pitfalls in using synthetic data for comparing CE models with respect to accuracy of their estimates. Nevertheless, these experiments form an excellent tool for characterizing model behaviors and evaluating trade-offs in the context of simple, yet realistic, scenarios.

To summarize our test efforts, we chose a "winning" CE model by evaluating performance of several model variations against a diverse set of real customer workloads and benchmarks (as described in section 3.1). To ensure that the winning model is well-behaved, we devised various synthetic experiments that characterized and validated behavior of the model against its underlying assumptions (one of which is described in section 3.2). To ensure that the model implementation is reliable, we performed large-scale stochastic testing as well as targeted functional testing for query graphs of interest (the details of which were skipped in the interest of space and time).

4. RELATED WORK

Whilst there is much literature on the construction of histograms for Cardinality Estimation (see [9] for a history) and particular subsets of the problem (see [10] for example), there is relatively little on using histograms for cardinality estimation in a relational database with its wide variety of stacked filter, join, and group by operators. There is even less on the QA processes used to validate cardinality estimation models in the database industry.

Reddy and Haritsa [6], proposed the use of plan diagrams to provide insights about an optimizer's plan space. We also used that approach for this work, and found that it nicely complemented our other methods. Experimental results based on plan diagram analyses are presented in the Appendix. Zait et al [5], discussed the use of real customer workloads to validate Oracle's database engine, and to detect plan changes. Similarly, Grabs et al [7], discussed a similar approach: they used real customer workloads as part of an iterative development process for new query processor features.

Looking ahead, an effort that would formalize and document the majority of cases where the basic assumptions of cardinality estimation models fail, would create an interesting benchmark that could be used to measure the effectiveness of cardinality estimation models and optimizers. We believe that this would be a very interesting topic for future research.

5. CONCLUSIONS

While developing an established relational database product, the parts of the product are extended and modified to better reflect new customer requirements. As the engineering team works to address such customer requirements over multiple years of product development, the design and the code may deviate from the original design. At the same time, new workload patterns (data, queries, hardware capacity, and characteristics) emerge, and the engineering team extends the product to accommodate these patterns. To enable the continued support and expansion of the product, it is often required to redesign and refactor components of the system. In this paper, we described our experience from such a case: the redesign, implementation, and validation of SQL Server's cardinality estimation model. Our testing methodology consists of two primary parts: (a) real workloads that measure the end-to-end quality of the query optimization process in presence of the new cardinality estimation model, and (b) synthetic tests that validate our implementation is in line with our intentions.

6. REFERENCES

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

[2] H. Bati, L. Giakoumakis., S. Herbert, and A. Surna. A genetic approach for random testing of database systems. *In Proceedings of the 33rd international conference on Very large data bases (VLDB '07)*. VLDB Endowment 1243-1251.

[3] Y. Ioannidis and V. Poosala. Histogram-Based Solutions to Diverse Database Estimation Problems. EE Data Eng. Bull.}. 18, 3 (1993), 10-18.

[4] I. Jose. Ascending Keys and Auto Quick Corrected Statistics, Ian Jose's Weblog, http://blogs.msdn.com/b/ianjo/archive/2006/04/24/582227.aspx, 2006.

[5] M. Zait, A. Lee, K. Yagoub, R. Sahani, H. Casaletto, and L. Kumar. Testing on a budget: integrating e-business certification into the Oracle DBMS testing. In *Proceedings of the Second International Workshop on Testing Database Systems (DBTest '09)*. ACM, New York, NY, USA, Article 2

[6] N. Reddy and. J. Haritsa. Analyzing plan diagrams of database query optimizers. In *Proceedings of the 31st international conference on Very large data bases (VLDB '05)*. VLDB Endowment 1228-1239.

[7] T. Grabs, S. Herbert, and X. Zhang. Testing challenges for extending SQL server's query processor: a case study. In *Proceedings of the 1st international workshop on Testing database systems (DBTest '08)*. ACM, New York, NY, USA, Article 2

[8] P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, T. Price. Access path selection in a relational database management system. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 23-34, 1979.

[9] Y. Ioannidis. The history of histograms (abridged). In *Proceedings of VLDB Conference*, 2003.

[10] C. Estan and J. Naughton. End-biased samples for join cardinality estimation. In *Proceedings of the 22nd International Conference on Data Engineering*, 2006.

[11] A. Swami and K. B. Schiefer. On the estimation of join result sizes. In Proceedings of the 4th international conference on extending database technology, EDBT 1994.

APPENDIX

A. ANALYZING PLAN DIAGRAMS

In addition to customer workloads and our various types of synthetic testing, we also mapped out the plan space of both of our CE versions using the Picasso tool [6]. Figure 6 shows the plan space of TPC-H, query 9 for the old cardinality estimation model, and Figure 7 shows the plan space of the same query using the chosen new cardinality estimation model.

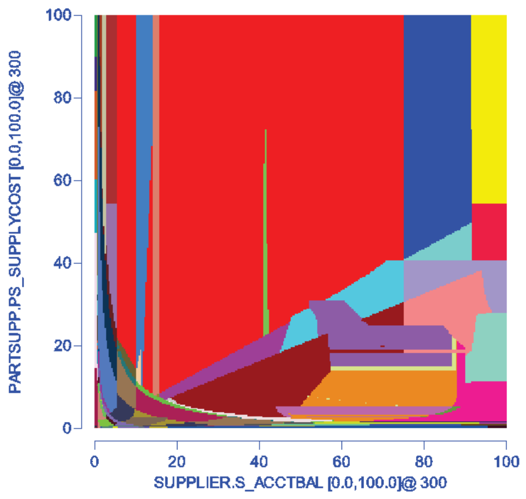

Figure 6: Plan diagram of TPC-H, Query 9, for the old cardinality estimation model. Number of distinct plans: 78.

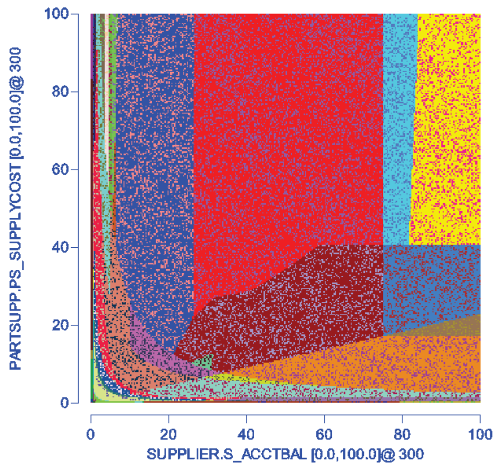

Figure 7: Original (Snowy) Plan Diagram of TPC-H, Query 9, for the new cardinality estimation model. Number of distinct plans: 83.

In these diagrams, the y-axis corresponds to varying selectivity of the predicate on PARTSUMM.PS_SUPPLYCOST, and the x-axis represents varying selectivity on the SUPPLIERS.S_ACCTBAL predicate. Each color region corresponds to a query plan. Also note that color schemes are not shared between figures – the plan from the red region in Figure 6 is not necessarily the same plan as that indicated by the red region in Figure 7.

When comparing Figure 6 and Figure 7, we consider: the number of overall plans (simplicity) and the continuity of the plan regions (stability). Simplicity is desirable up to a point – while we don't want to see a single plan covering the entire space for a complex query, we also don't want to see a diagram cluttered with too many plans. Stability is quite desirable.

Between these two diagrams, it is clear that the plan diagram depicted in Figure 6 contains several discontinuous regions. However, the plan diagram in Figure 6 contains fewer plans (76 as opposed to 83) than the one in Figure 7, which also exhibits what we call "snow" (or speckled pattern as termed in [6]) – plan regions where one basic plan switches intermittently to another plan. We found the snow effect very troubling, as it indicates a lack of plan choice stability. Upon investigation, we discovered that this snow was caused by floating point inconsistencies in the code, and we were able to adjust the order of calculations to reduce this phenomenon. We suspect that this snow could crop up in any complex CE model that relies on floating point arithmetic and could be mitigated by careful ordering of all composite floating point calculations. The fixed code for the new CE produces a much simpler and more regular plan diagram which is depicted in Figure 8. The total number of distinct plans in Figure 8 is 46, which compares to 78 for the old CE. We believe this reduction in distinct plans will manifest itself in less volatile query plan selection. While the plan diagrams presented here are specific to TPC-H query 9, we performed similar analyses for other queries from the TPC-H benchmark and we are quite pleased with the superior behavior exhibited by the new CE model with respect to simplicity and stability of plan diagrams it yields for those queries.

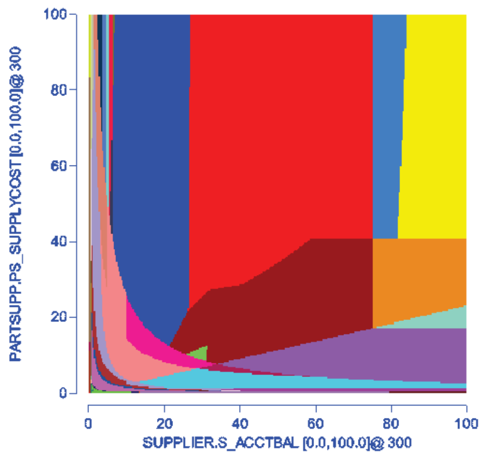

Figure 8: Plan diagram of TPC-H, Query 9, for the new cardinality estimation model. Number of distinct plans: 46.