QUEST: An Exploratory Approach to Robust Query Processing

Anshuman Dutt Sumit Neelam Jayant R. Haritsa*
Database Systems Lab, SERC/CSA
Indian Institute of Science, Bangalore, INDIA

1. INTRODUCTION

In modern database systems, accurate estimations of predicate selectivities are a basic requirement for the effective optimization of declarative SQL queries. For example, consider **EQ**, the simple SPJ query shown in Figure 1 – here, the optimizer needs to estimate the selectivities of a selection predicate ($p\_retailprice < 1000$) and two join predicates ($part \bowtie lineitem$, $lineitem \bowtie orders$).

```
select * from lineitem, orders, part
where p_partkey = l_partkey and l_orderkey = o_orderkey
      and p_retailprice < 1000
```

Figure 1: Example Query (EQ)

In practice, however, compile-time selectivity estimates are often significantly in error with respect to the actual values subsequently encountered during query execution. These errors mislead the optimizer into making poor execution plan choices, resulting in substantially inflated query response times.

Over the past few decades, the database research community has spent considerable efforts to address the above problem, which is of immediate relevance to currently operational systems. The proposed techniques (see [1] for a comprehensive survey) include sophisticated meta-data structures, feedback-based statistical adjustments, and on-the-fly re-optimization strategies. While this rich body of literature features several innovative formulations, a common limitation is their inability to provide *robust query processing*, as per the following definition.

Robustness Metric. Given a query Q with an associated error-prone selectivity space ESS, our notion of robustness is the maximum performance sub-optimality **(MSO)** that could occur due to selectivity estimation errors, as compared to an idealized system that magically knows the correct values of all selectivities. Specifically, let q_e denote the optimizer's estimated query location in the ESS, and q_a denote the actual run-time location. Also, denote the plan chosen by the optimizer at q_e by P_{oe}, and the optimal plan at q_a by P_{oa}. Finally, let $cost(P_j, q_i)$ represent the execution cost incurred at an arbitrary ESS location q_i by plan P_j. Then, robustness is defined by the normalized metric:

$$MSO(Q) = \max_{q_e, q_a \in ESS(Q)} [\frac{cost(P_{oe}, q_a)}{cost(P_{oa}, q_a)}]$$

which ranges over $[1, \infty)$. The reason for considering *all* possible (q_e, q_a) combinations is to ensure that robustness is defined independent of both the data distributions (which impact q_a) and the meta-data statistics (which impact q_e).

Plan Bouquets. We recently proposed in [2], a new query processing concept called "plan bouquets", which provides *guaranteed upper bounds on MSO* for the first time in the literature. The basic idea in the bouquet approach is to completely jettison the compile-time estimation process for error-prone selectivities. Instead, these selectivities are systematically *discovered* at run-time through a precisely calibrated sequence of cost-limited executions from a carefully chosen small set of plans, called the plan bouquet. With this approach, whose construction is delineated in Section 2, we can prove that $MSO \leq 4 * |PlanBouquet|$. Moreover, a detailed empirical assessment indicates that the actual MSO is usually well within the theoretical bound, even on high-dimensional ESS.

As a case in point, when Query 19 of the TPC-DS benchmark, which features 5 error-prone join selectivities, is executed on the native PostgreSQL optimizer, its MSO is an astronomical 10^6. However, with the bouquet approach, the MSO is apriori guaranteed to *not exceed 30*, and the empirical value is just 10!

Apart from robustness, another unique benefit of the bouquet mechanism's choosing to abjure selectivity estimation is that query execution strategies are *repeatable* across different invocations, making it particularly attractive in industrial applications.

Finally, the bouquet approach can be largely implemented by leveraging functions, such as *selectivity injection* and *abstract plan costing*, that have already found expression in modern database engines, making it relatively easy to port to current systems.

Demo Features. We have fully incorporated the plan bouquet concept on the PostgreSQL engine, in a prototype implementation called **QUEST** (QUery Execution without Selectivity esTimation). The various features of QUEST will be visually and interactively showcased during the demo, as described in detail in Section 3. These include: (a) The poor MSO of current database engines; (b) The MSO guarantees of the bouquet approach, and its execution repeatability; and (c) Enhancements that further improve the bouquet performance. A complete video of QUEST in operation is available at the project website [4].

*Contact Author: haritsa@dsl.serc.iisc.ernet.in

Proceedings of the VLDB Endowment, Vol. 7, No. 13

2. QUEST SYSTEM

The complete architecture of the QUEST system is shown in Figure 2, divided into a compile-time phase and a run-time phase.[1] We explain the working of these two phases through a restricted 1D version of the EQ example query (Figure 1) wherein only the p_retailprice selection predicate is error-prone.

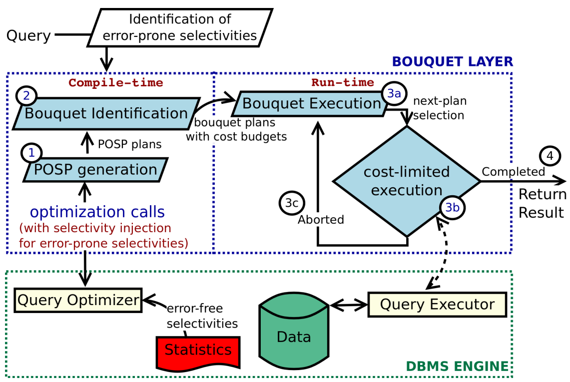

Figure 2: QUEST Architecture

2.1 Compile-time: Bouquet Identification

First, through repeated invocations of the optimizer, and explicit *injection* of selectivities, we identify the "Parametric Optimal Set of Plans" (POSP) over the entire selectivity range (0 – 100%) of the p_retailprice predicate. The costs of these five plans, P1 through P5, are enumerated in Figure 3, using abstract plan costing over the range. From these plots, we derive the "POSP Infimum Curve" (**PIC**), defined as the trajectory of the minimum cost from among the POSP plans – this curve represents the ideal performance.

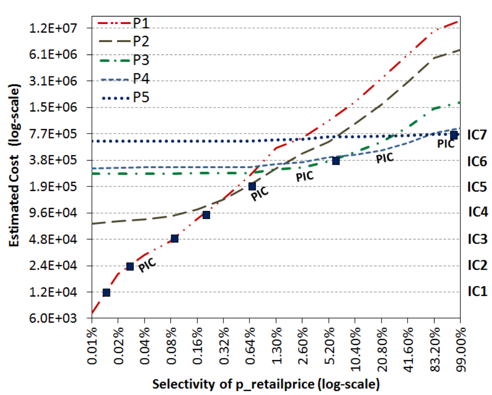

Figure 3: POSP performance (log-log scale)

The next action, which is a distinctive feature of the bouquet approach, is to *discretize* the PIC by projecting a graded progression of *isocost* (IC) steps onto the curve. For example, in Figure 3, the dotted horizontal lines represent a geometric progression of isocost steps, IC1 through IC7, with each step being *double* the preceding value. The intersection of each IC with the PIC (indicated by ■) provides an associated selectivity, along with the identity of the best POSP plan for this selectivity. For example, in Figure 3, the intersection of IC5 with the PIC corresponds to a selectivity of 0.65%, and POSP plan P2. The subset of POSP plans associated with the intersections forms the "plan bouquet" for the given query – in Figure 3, the bouquet consists of {P1, P2, P3, P5}.

[1] All diagrams in this paper should be viewed from a *color copy* to ensure clarity of their contents.

2.2 Run-time: Bouquet Execution

The bouquet execution is now presented through an EQ instance wherein the actual selectivity of p_retailprice is 5%, i.e. q_a=5%. We begin by partially executing plan P1, corresponding to the cheapest isocost step IC1, until the execution overheads reach IC1 (1.2E4 | 0.015%). Then, we extend our cost horizon to IC2, and continue executing P1 until the overheads reach IC2 (2.4E4| 0.03%), and so on until the overheads reach IC4 (9.6E4 | 0.2%). At this juncture, there is a change of plan to P2 as we look ahead to IC5 (1.9E5 | 0.65%), and during this switching all the intermediate results produced by plan P1 are *jettisoned*. The new plan P2 is executed till the associated overhead limit (1.9E5) is reached. The cost horizon is now extended to IC6 (3.8E5 | 6.5%), in the process jettisoning plan P2's intermediate results, and switching to executing plan P3 instead. In this case, the execution will complete before the cost limit is reached since the actual location, 5%, is within the selectivity range covered by IC6.

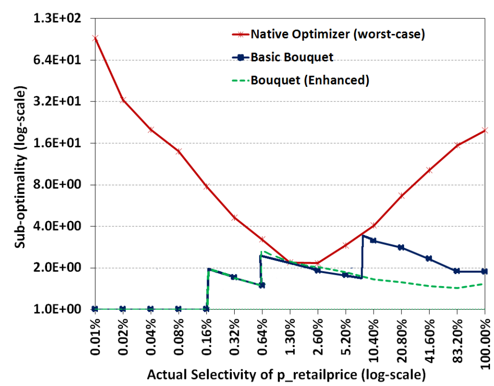

Figure 4: Sub-Optimality Profiles (log-log scale)

Performance Characteristics. The sub-optimality of the bouquet for the above ($q_a = 5\%$) execution turns out to be 1.78 since the exploratory overheads are 0.78 times the optimal cost, and the optimal plan itself is used for the final execution. In contrast, the native optimizer can suffer a sub-optimality of 3 when it estimates q_e to be in $(0, 0.3\%]$. Extending this comparison to the entire selectivity range for q_a results in Figure 4, where the global robustness improvement due to the bouquet approach is clearly demonstrated. In absolute terms, the MSO for the bouquet is only 3.6 (blue line) as compared to almost 100 for the native optimizer (red line). Moreover, if the enhancements enumerated in [2] are incorporated, the MSO for the bouquet drops even further to 3.1 (green line).

2.3 Extension to Multiple Dimensions

When the above approach is generalized to multi-dimensional ESS, the IC steps and the PIC curve become surfaces, and their intersections represent selectivity surfaces on which multiple bouquet plans may be present. Notwithstanding these changes, the basic mechanics of the bouquet algorithm remain very similar to the 1D case. The primary difference is that we jump from one IC surface to the next only after it is determined that *none* of the bouquet plans present on the current IC surface can completely execute the given query within the associated cost budget. The complete algorithmic details are available in [2].

3. QUEST DEMONSTRATION

In the demonstration, the audience will engage with a variety of visual scenarios crafted to highlight the selectivity estimation problems that plague current database optimizers, and the novel robustness characteristics that the bouquet technique brings to bear on these chronic problems. A two-dimensional ESS based on Query 5 of the TPC-H benchmark, with selection predicates on *Customer* and *Lineitem* as the error-prone selectivity dimensions, is used as a running example to explain these scenarios. The evaluation is carried out on fully-indexed 1 GB TPC-H databases[2] hosted on the PostgreSQL engine.

3.1 Sub-optimality of Native Optimizer

We begin with presenting plausible instances of the native optimizer making large estimation errors, covering both under-estimation and over-estimation. A sample instance of the corresponding QUEST interface is shown in Figure 5, where the audience can observe:

- An operator-level comparison between P_{oe} and P_{oa} – in this instance, P_{oe} features a series of *Nested Loop* joins while P_{oa} opts for *Hash Joins*, and the join orders are different.
- The locations of q_a and q_e in the ESS, and the large error gap between them – in this instance, q_a=(30.9%, 26.7%) while q_e is a significant underestimate, specifically (0.25%, 3.1%).
- The adverse performance impact due to the estimation error – in this instance, the sub-optimality is around 17.

We will also highlight a compelling example where the optimizer makes the same q_e estimate for a pair of similar-looking queries with only a miniscule difference in their query texts – however, in reality, their actual locations lie at *opposite corners* of the ESS.

User Interaction. The audience will be supplied with a parameterized query template for which they can provide their desired choices of constants, and the sub-optimality of the native optimizer will be evaluated on these queries.

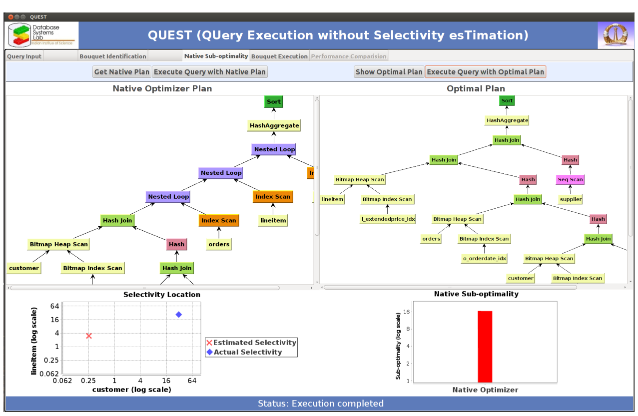

Figure 5: Sub-optimality of Native Optimizer

3.2 Bouquet Identification

Turning our attention to the bouquet technique, we start with the compile-time phase i.e. bouquet identification, whose graphical display is shown in Figure 6. Here, the left picture captures the three-dimensional PIC surface of the native optimizer, characterized by a large number of POSP plans and a steep cost-profile over the ESS. Since the bouquet's MSO guarantee is a direct function of the POSP cardinality, the dense cost diagram is subjected to *anorexic reduction* [3] – with this operation, the number of plans is reduced to a small number without substantively affecting the query processing quality of any individual query in the ESS. On this reduced diagram, the bouquet's distinctive feature of cost-based discretization using geometrically increasing isocost planes is applied – the combined effect of reduction and discretization is presented in the second picture of Figure 6.

In the example, the original cost diagram has 29 plans with a PIC covering the cost range from 1.1E4 to 3.2E5. After anorexic reduction with cost-increase threshold $\lambda = 20\%$ (as recommended in [3]), the plan cardinality goes down to 6 plans. Finally, the PIC is geometrically divided using 5 isocost contours with common ratio r=2, and the plan cardinality distribution on these contours is (4, 4, 4, 3, 1), respectively.

User Interaction. The audience will be able to provide their desired values for the reduction parameter λ and the discretization parameter r. For the chosen values, the resulting MSO guarantee will be evaluated and compared against that obtained with the default values ($\lambda = 20\%, r = 2$).

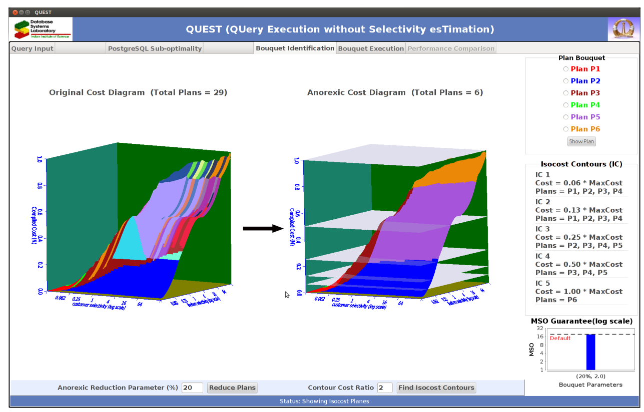

Figure 6: Bouquet Identification Interface

3.3 Bouquet Execution

The next segment is the main thrust of the demo – illustrating the bouquet technique's calibrated sequence of budgeted partial executions, starting with plans on the cheapest isocost contour, and then systematically working our way through the contours until one of the plans executes the query to completion within its assigned budget. The dynamism of this iterative process is captured in the interface shown in Figure 7, which is continually updated to indicate:

- The ESS region covered by each partial plan execution – subsequent to each such execution, the associated region is shadowed with the plan's color.
- The execution order timeline of the plans, along with their tree structures – this allows database analysts to carry out offline replays of the plan execution sequence.
- The contour budgets, which initially appear as white bars of geometrically increasing height, and are then filled with blue after the corresponding partial executions (in the figure, after 15 partial executions, plan P6 on Contour 5 completes the query within the assigned budget).
- The sub-optimality of bouquet execution (for the sample query, it is around 3.7, depicted by the dark blue bar in Figure 7).

User Interaction. Controls are provided to enable pausing the bouquet operation after each partial execution so that the specific progress made through each such execution can be fully assimilated before continuing to the next step.

[2]Both uniform and skewed versions of TPC-H are evaluated.

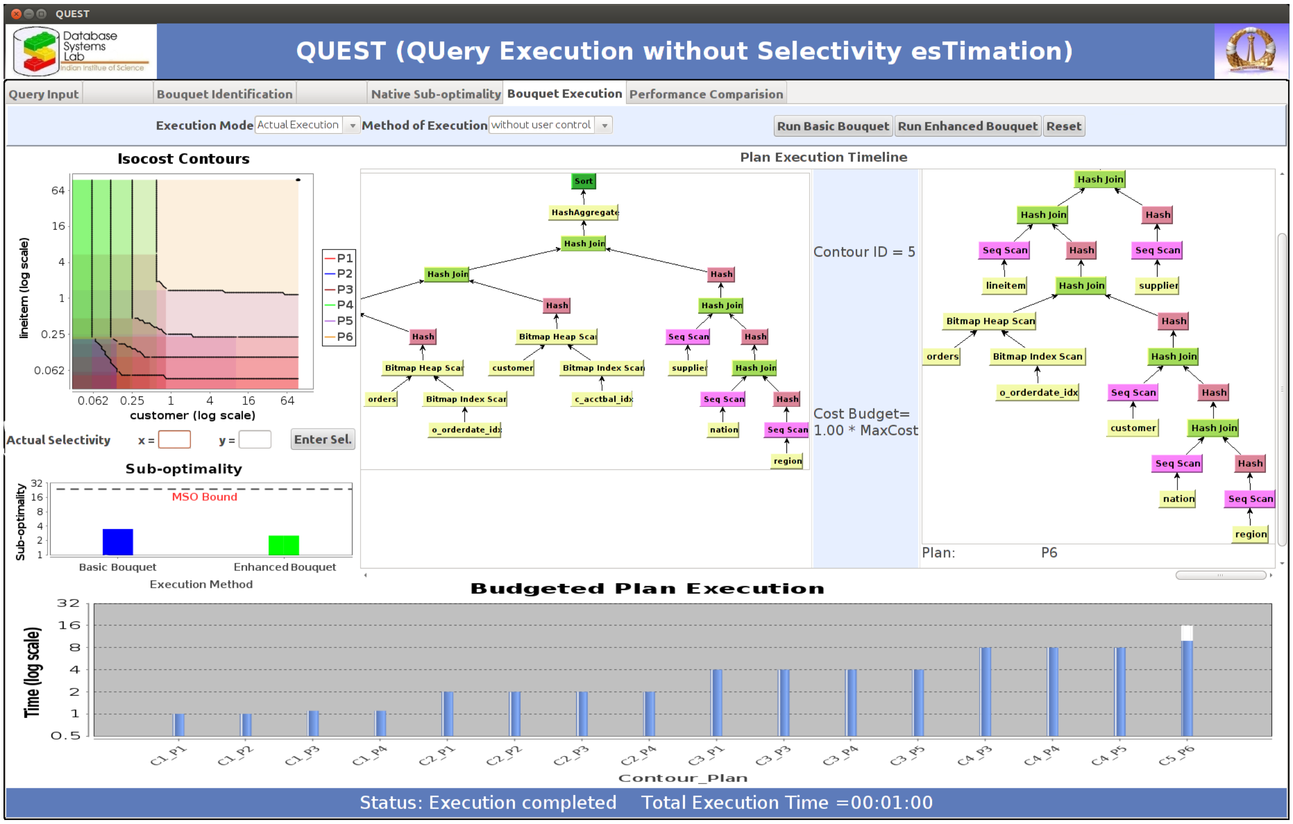

Figure 7: Bouquet Execution Interface

3.4 MSO Guarantees and Repeatability

In this scenario, the audience will have the opportunity to verify for themselves the MSO and repeatability guarantees offered by the bouquet technique. Firstly, with regard to the MSO guarantee, the audience can fill in any desired location of q_a in the text box shown in Figure 7 (below the isocost contours), and then invoke the bouquet algorithm on this query instance to confirm that the sub-optimality incurred is within the apriori stated bound (for the sample query, this MSO bound is less than 20, which is orders of magnitude lower than the empirically determined MSO of 10^4 obtained with the native optimizer).

Secondly, with regard to repeatability, our goal is to prove that, unlike the native optimizer, the bouquet execution sequence is only a function of q_a, and not of q_e. We will therefore allow the audience to radically alter, using the CODD metadata editing tool [5], the distribution histograms of the attributes featured in the query, while keeping the underlying data *unchanged*. Subsequent to the alteration, the bouquet algorithm will be re-executed and confirmed to behave identically to its prior incarnation.

3.5 Enhanced Bouquet Algorithm

In our final scenario, we showcase an enhanced version of the bouquet algorithm that explicitly *monitors* the selectivities encountered during the partial executions. The monitoring serves to reduce the number of plan executions incurred in crossing contours, as explained in [2]. With this reduction in overheads, the sub-optimality comes down to just 2.7, depicted by the bright green bar in Figure 7.

The audience can visually observe the impact of the enhancements through a graph that continually tracks the monitored selectivity movement in the ESS, as shown in Figure 8. Here, the dotted line characterizes the trajectory followed by the bouquet in moving from the origin to the destination q_a location.

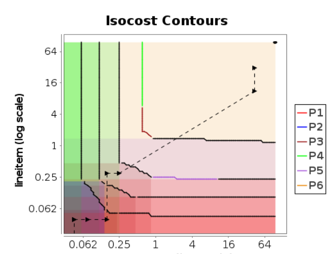

Figure 8: Enhanced Bouquet with Selectivity Monitoring

Closing Remarks. The QUEST demo provides a visual and interactive tour of how the recently proposed plan bouquet technique delivers novel performance guarantees that open up new possibilities for robust query processing.

4. REFERENCES

[1] A. Deshpande, Z. Ives and V. Raman, "Adaptive Query Processing", *Foundations and Trends in Databases*, Now Publishers, 2007.

[2] A. Dutt and J. Haritsa, "Plan Bouquets: Query Processing without Selectivity Estimation", *ACM SIGMOD Conf.*, 2014.

[3] Harish D., P. Darera and J. Haritsa, "On the Production of Anorexic Plan Diagrams", *VLDB Conf.*, 2007.

[4] dsl.serc.iisc.ernet.in/projects/QUEST

[5] dsl.serc.iisc.ernet.in/projects/CODD