Robust Query Processing: Mission Possible

Jayant R. Haritsa
Dept. of Computational & Data Sciences
Indian Institute of Science
Bangalore, India
haritsa@iisc.ac.in

***Abstract*—Robust query processing with strong performance guarantees is an extremely desirable objective in the design of industrial-strength database engines. However, it has proved to be a largely intractable and elusive challenge in spite of sustained efforts spanning several decades. The good news is that in recent times, there have been a host of exciting technical advances, at different levels in the database architecture, that collectively promise to materially address this problem. In this tutorial, we will present these novel research approaches, characterize their strengths and limitations, and enumerate open technical problems that remain to be solved to make robust query processing a contemporary reality.**

***Keywords*—Declarative Queries, Optimization, Robust Execution, Performance Guarantees**

I. Overview

An organic reason for the ubiquitous popularity of database management systems is their support for declarative user queries, typically expressed in SQL. In this framework, the user only specifies the end objectives, leaving it to the database system to identify and execute the most efficient means, called "plans", to achieve them. These two steps are performed by the *query optimizer* and the *query executor* components, respectively, within the core of the database engine. Over the past half-century, research on the design and implementation of these components has been a foundational topic for both the academic and industrial database communities.

The unfortunate reality is that, in spite of this sustained and long-standing research, the resulting solutions have largely remained a "black art". This is due to the well-documented complexities and challenges of database query processing [8], [9]. In fact, even as recently as 2014, a highly-respected industry veteran was provoked to lament: *The wonder isn't "Why did the optimizer pick a bad plan? Rather, the wonder is "Why would the optimizer ever pick a decent plan?"*! [29]. Similar sentiments have been expressed by other academic and industrial experts as well, including: *"Query optimizers do a terrible job of producing reliable, good plans (for complex queries) without a lot of hand tuning."* [13], and *"Almost all of us who have worked on query optimization find the current state of the art unsatisfactory with known big gaps in the technology."* [10].

It is important to note that, due to the above problems, the scale of performance degradation faced by database queries can be *huge* – often in *orders of magnitude* as compared to an oracular ideal that magically knows the correct inputs required for optimal query processing. As a case in point, when Query 19 of the TPC-DS benchmark is executed on PostgreSQL, the worst-case slowdown, relative to the hypothetical oracle, can exceed a *million*! [14]. Moreover, apart from the obvious negative impacts on user productivity and satisfaction, there are also financial implications of this performance degradation – the total cost of ownership is significantly increased due to over-provisioning, lost efficiency, and increased human administrative costs [38].

In the midst of this gloom and doom, the positive news is that in recent times, there have been a host of exciting research advances, which collectively promise to provide strong foundations for designing the next generation of query processing engines. The expectation is that these advances will eventually organically support **robust query processing (RQP)**, relegating to the past the above-mentioned cynicism on this bedrock objective. Many of the new ideas owe their genesis to a series of influential and well-attended Dagstuhl Seminars on the topic of Robust Query Processing over the last decade [1]–[3]. Further, they have arisen from research teams located at diverse locations across the world, including the US, Europe and Asia.

In this tutorial, we will provide a detailed and holistic coverage of these contemporary RQP innovations, highlight their strengths and limitations, and outline a set of open technical problems and future research directions.

II. Tutorial Contents

The definition of robustness itself has been a subject of intense debate for a long time, and a consensus has been difficult to achieve [1]. For instance, if worst-case performance is improved at the expense of average-case performance, is that an acceptable notion of robustness? Or, would graceful degradation, as opposed to "performance cliffs", be the right perspective? Alternatively, is it the ability to seamlessly scale with workload complexity, database size and distributional skew? Or, could we settle for providing strong theoretical guarantees relative to the oracular ideal? Perhaps, the real answer is that robustness encompasses all of these scenarios and more, with the specific choice being application-dependent.

The above semantic tangle is further complicated by the different levels at which notions of robustness can be introduced – for instance, at the granularity of individual *operators* (e.g [7]), or through entire query *plans* (e.g [11]), or over end-to-end query *executions* (e.g. [14]). Moreover, one can take *algorithmic* (e.g. [37]), *statistical* (e.g [40]) or *learning-based* (e.g. [30]) approaches to incorporate the robustness features at these various levels.

We plan to cover representative techniques along all these various dimensions in the tutorial. The big picture is that a rich variety of possibilities are currently available, and a judicious selection could potentially lead to the desired robustness. Moreover, with the impending advent of the so-called Big Data world, wherein data will be the engine driving virtually all aspects of human endeavour, the role of RQP will soon assume critical proportions.

The tutorial is organized in a sequence of five stages, whose contents are summarized below.

A. Stage I: Robust Operators

A fertile area of research has been the introduction of robustness into specific operators that appear in the query execution plan. For instance, **SmoothScan** [7] is an adaptive access path that, based on the statistical properties of the data, continually morphs between the sequential scan and index access options to a table. Specficially, at small selectivity values, it behaves similar to an index scan, whereas for higher selectivities, it progressively changes its behavior towards a sequential scan.

Another unified operator is **G-join** [16], which brings the popular join algorithms (*nested-loops*, *sort-merge* and *hash*) into a common framework. This merger makes it unnecessary for the optimizer to have to choose between the alternatives, thereby preventing mistakes. Similar unifications have been developed for the grouping and duplicate elimination operators as well.

Handling data skew in high-performance systems is an essential requirement for distributed joins that hope to achieve load balancing and thereby scalability. A novel approach to achieve this goal is **Flow-join** [36], wherein instead of constructing detailed statistics, or carrying out extra analysis, they opt for a lightweight approach that trades off communication for computation. Specifically, they detect "heavy hitter" tuples in the initial runtime phase using small approximate histograms, and subsequently avoid load imbalances by *broadcasting* tuples that join with these heavy hitters.

Yet another innnovative approach for indexed nested-loop joins was proposed recently in [3], based on dynamic routing of individual tuples, reminiscent of the well-known Eddies framework [5]. Here, multiple plans are allowed to proceed in parallel, and through a system of back pressure, tuples are preferentially led away from the inefficient plans towards efficient plans. Further, the execution is always making forward progress, without ever backtracking or throwing away partial results. A preliminary evaluation showed a worst case degradation of two times compared to the ideal, and orders of magnitude improvement as compared to the native optimizer choices.

B. Stage II: Robust Plans

We turn our attention in the second stage to techniques that attempt to provide robustness at the granularity of entire plans. When faced with unknown input parameter values that become available only at run-time, contemporary optimizers typically approximate the parameter's distribution using some representative value - for example, the mean or mode - and then always choose the corresponding plan to execute the query. But this will obviously not work well if the actual value encountered at run-time is significantly different from the representative value. Therefore, an alternative strategy proposed in [11] is to instead instead optimize for the "least expected cost" **(LEC)** plan, where the expectation is computed over the full distribution of the input parameters.

Computing the LEC plan involves substantial computational overheads when the number of plans over the parameter space is large. Further, it also assumes that the candidate plans have all been modeled at the same level of accuracy, rarely true in practice. However, these limitations can be addressed by leveraging the *anorexic plan reduction* technique proposed in [17], wherein the number of candidate plans is significantly reduced to a small absolute number through substitution, without materially affecting the query processing quality at any location in the parameter space.

An alternative approach to plan robustness, called **SEER**, was taken in [18], that combined the above-mentioned anorexic plan reduction with a generalized mathematical characterization of plan cost behavior over the parameter space. This formulation lends itself to efficiently establishing guarantees on the behavior of the substitute plans as compared to the optimizers standard choices. In particular, they proved the powerful result that the behavior on the corners of the parameter space can be used to deterministically predict the behavior in the interior of the space, resulting in efficient replacement strategies. A particularly attractive feature is that the plan replacements never materially harm, but often significantly help with respect to the original optimizer choices.

C. Stage III: Robust Execution

In this stage, we move on to the robust execution of entire queries. The specific performance metric used here is **Maximum Sub-Optimality (MSO)**, which is defined as the worst-case slowdown, evaluated over the entire selectivity space, relative to an oracular ideal that magically knows the correct selectivities.

An early work that attempted to provide MSO guarantees with regard to query performance was described in [33]. Here, they first came up with constraints on the magnitude of the selectivity estimation inaccuracy, measured using the *q-error* metric, such that the finally chosen plan was equivalent to the optimal. They also then went on to give a bound on the performance sub-optimality incurred for an arbitrary estimation error, and showed that the sub-optimality was within a *quartic* dependency on this error. However, this high-degree polynomial dependency makes the guarantee to be impractically large when the estimation error is significant, as is often the case. Moreover, it is often not possible to apriori know the value of the error, making it infeasible to provide a bound to the user at query submission time.

A radically different approach to bounding suboptimality that is *error-independent* and small in absolute value was presented in [14]. In this execution technique, called **Plan-Bouquet**, the brittle selectivity estimation process is completely abandoned, and replaced with a calibrated "trial-and-error" discovery mechanism. This technique lends itself to provable MSO guarantees even in situations where state-of-the-art systems may suffer from arbitrarily poor execution. An improved version of PlanBouquet, called **SpillBound**, which significantly accelerates the selectivity discovery process, and provides platform-independent performance guarantees, was recently presented in [22]. Specifically, its MSO is bounded by $\mathbf{D^2 + 3D}$, where D is the dimensionality of the `ESS`, i.e. the expected number of error-prone predicates in the input query. So, for instance, if $D = 3$, the sub-optimality can never exceed 18, *irrespective of the query location in the space*.

D. Stage IV: Robust Cost Models

Thus far, our focus has been primarily on incorporating robustness with respect to the *operator cardinality model*, which is primarily responsible for the poor choice of runtime plans. However, an alternate source of errors that can adversely impact robustness is the *operator cost model*, and this issue has also been the subject of several studies during the past decade. Note that these two models address very different aspects of the data processing environment – the cardinality model reflects the ability to capture the distributions and correlations present in the data, whereas the cost model registers the ability to capture the behavior of the underlying hardware and physical operator implementations.

The effectiveness of machine learning techniques to predict query execution time, by using both plan-level and operator-level models, was demonstrated in [4]. Their features included optimizer cost estimates, query parameters and the actual runtime statistics. In marked contrast, another research group showed in [40] that, with proper augmentation and tuning, existing statistical models could themselves produce satisfactory estimates of query execution times. In their approach, an initial offline profiling phase was used to to establish the unit temporal costs for utilizing various system resources. Subsequently, online sampling was employed to estimate the number of usages for a given query. In follow-up papers [41], [42], stronger statistical models were incorporated in their algorithmic suite to maintain the prediction quality in the presence of uncertainty and concurrency.

The above results were for pure SQL queries, but extensions such as UDFs create challenges of costing imperative code. A promising mechanism to circumvent this problem is to convert UDFs into equivalent SQL expressions that are inlined into the calling query, as proposed recently in [35].

E. Stage V: Future Research Directions

In the final stage of the tutorial, we will summarize the results and conclusions of the previous stages, and then outline a set of open technical problems and future research directions. Sample problems include the following:

Geometries of Plan Cost Functions: Most of the prior work has only assumed that plan cost functions are monotonic with regard to selectivities. However, in practice, these functions often exhibit greater regularity in their behavior. As a case in point, plan costs are modeled in [15] as low-order polynomial functions of plan selectivities, leading to the *Bounded Cost Growth* property, which is leveraged to achieve bounded suboptimalities. An even stronger constraint that is found to generally hold in practice is *concavity*, resulting in monotonically non-increasing slopes. Such profiles can be utilized to improve the robustness of the query processing solutions, or the associated overheads, as shown recently in [23].

Query-Graph Sensitive Robustness: The robustness techniques developed in the literature thus far are largely agnostic to the join-graphs of the queries under consideration. Howver, these graphs often exhibit a regular structure such as *chain*, *cycle*, *star*, *clique*, etc., and it is likely that this information could be gainfully used to improve the robustness of the resulting execution. For instance, it should be simpler to assure good performance for chain queries, where the optimization choices are comparatively limited, as opposed to star queries.

Robustness Benchmarks: A pre-requisite for confirming the robustness offered by new approaches is the development of principled benchmarks that exercise and push the system to its limits. This is critical since the standard benchmarks, such as TPC-DS, measure performance, not robustness. Some recent efforts in this direction include **OptMark** [28], **JOB** [26] and **OTT** [43], but there remain aspects of robustness that are yet to be covered.

Machine Learning Techniques for Component Selection: We advocate the database engine to have a *multiplicity* of components that separately but cooperatively cater to the various query processing environments. For instance, if the cardinality estimates are expected to be reasonably accurate, then the native query optimizer is appropriate for choosing the plan. On the other hand, if the estimates are expected to be brittle, then the SpillBound technique can be invoked instead. An obvious question that arises with such an architecture is how to determine the specific environment that is currently operational, and hence the associated component. Machine learning techniques could be used to judiciously make this choice, similar to the exercise recently carried out in [20] in the context of analytical data flows.

Graceful Performance Degradation: A major problem faced in real deployments is the presence of "performance cliffs", where the performance suddenly degrades precipitously although there has only been a minor change in the operational environment. This is particularly true with regard to hardware resources, such as memory. So, an important future challenge is to design algorithms that *provably* degrade gracefully with regard to all their performance-related parameters.

III. Target Audience

Robust support for declarative query processing has been a long-standing concern for the database community. In particular, the target audience for this tutorial includes researchers, developers and students with an interest in the internals of database engines. The background expected is that of an introductory database systems course covering relational data models, declarative query languages, and basic query optimization and processing techniques.

The primary source material for the tutorial consists of the papers highlighted in the above presentation, complemented by supporting inputs from the rich corpus of literature on query optimization and processing. A sampling of relevant publications is given in the reference list, with emphasis on recent contributions to the field.

IV. Presenter Details

Jayant Haritsa has been on the faculty of the Dept. of Computational & Data Sciences and the Dept. of Computer Science & Automation at the Indian Institute of Science, Bangalore, since 1993. He received his undergraduate degree from the Indian Institute of Technology – Madras, and the MS and PhD degrees from the Univ. of Wisconsin – Madison.

Jayant Haritsa is a Fellow of ACM and IEEE for his contributions to the design and implementation of database engines. He was the Program Co-Chair of VLDB 2016, ICDE 2010 and DASFAA 2008, and a PC vice-chair for ICDE 2005 and ICDM 2007. Also, he has been the Tutorials Chair for WWW 2010 and DASFAA 2005, and the Demonstrations Chair for VLDB 2009. He has served on the Editorial Boards of the TKDE, VLDBJ, JDPD and RTSS journals, and is currently a Trustee of the VLDB Endowment.

Acknowledgment

This work was supported by a J. C. Bose Fellowship from the Dept. of Science and Technology, Govt. of India.

References

[1] Robust Query Processing. Dagstuhl Seminar, 2010. www.dagstuhl.de/en/program/calendar/semhp/?semnr=10381.

[2] Robust Query Processing. Dagstuhl Seminar, 2012. www.dagstuhl.de/en/program/calendar/semhp/?semnr=12321,

[3] Robust Performance in Database Query Processing. Dagstuhl Seminar, 2017. www.dagstuhl.de/en/program/calendar/semhp/?semnr=17222,

[4] M. Akdere, U. Cetintemel, M. Riondato, E. Upfal and S. Zdonik. Learning-based query performance modeling and prediction. *ICDE*, 2012.

[5] R. Avnur and J. Hellerstein. Eddies: Continuously Adaptive Query Processing. *SIGMOD*, 2000.

[6] S. Babu, P. Bizarro and D. DeWitt. Proactive Re-optimization. *SIGMOD*, 2005.

[7] R. Borovica-Gajic, S. Idreos, A. Ailamaki, M. Zukowski and C. Fraser. Smooth Scan: Robust Access Path Selection without Cardinality Estimation. *VLDBJ*, 27(4), 2018.

[8] S. Chaudhuri. An Overview of Query Optimization in Relational Systems. *PODS*, 1998.

[9] S. Chaudhuri. Query Optimizers: Time to rethink the contract? *SIGMOD*, 2009.

[10] S. Chaudhuri. Interview in XRDS, 19(1), 2012.

[11] F. Chu, J. Halpern and P. Seshadri. Least Expected Cost Query Optimization: An Exercise in Utility. *PODS*, 1999.

[12] F. Chu, J. Halpern and J. Gehrke. Least Expected Cost Query Optimization: What can we expect? *PODS*, 2002.

[13] D. Dewitt. Interview in Sigmod Record, 31(2), 2002.

[14] A. Dutt and J. Haritsa. Plan Bouquets: A Fragrant Approach to Robust Query Processing. *ACM TODS*, 41(2), 2016.

[15] A. Dutt, V. Narasayya and S. Chaudhuri. Leveraging re-costing for online optimization of parameterized queries with guarantees. *SIGMOD*, 2017.

[16] G. Graefe. New algorithms for join and grouping operations. *Computer Science – R&D*, 27(1), 2012.

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

[18] Harish, D., P. Darera and J. Haritsa. Identifying Robust Plans through Plan Diagram Reduction. *PVLDB*, 1(1), 2008.

[19] H. Harmouch and F. Naumann. Cardinality Estimation: An Experimental Survey. *PVLDB*, 11(4), 2017.

[20] F. Hueske. Specification and Optimization of Analytical Data Flows. *PhD Thesis*, 2016. nbn-resolving.de/urn:nbn:de:101:1-201804165046.

[21] N. Kabra and D. DeWitt. Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans. *SIGMOD*, 1998.

[22] S. Karthik, J. Haritsa, S. Kenkre, V. Pandit and L. Krishnan. Platform-independent Robust Query Processing. *IEEE TKDE*, 31(1), 2019.

[23] S. Karthik, J. Haritsa, S. Kenkre and V. Pandit. A Concave Path to Low-overhead Robust Query Processing. *PVLDB*, 11(13), 2018.

[24] M. Kiefer, M. Heimel, S. Bress and V. Markl. Estimating Join Selectivities using Bandwidth-Optimized Kernel Density Models. *PVLDB*, 10(13), 2017.

[25] A. Kipf, T. Kipf, B. Radke, V. Leis, P. Boncz and A. Kemper. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. *CIDR*, 2019.

[26] V. Leis, B. Radke, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper and T. Neumann. Query Optimization through the looking glass, and what we found running the Join Order Benchmark. *VLDBJ*, 27(5), 2018.

[27] V. Leis, B. Radke, A. Gubichev, A. Kempers and T. Neumann. Cardinality Estimation Done Right: Index-based Join Sampling. *CIDR*, 2017.

[28] Z. Li, O. Papaemmanouil and M. Cherniack. OptMark: A Toolkit for Benchmarking Query Optimizers. *CIKM*, 2016.

[29] G. Lohman. Is Query Optimization a Solved Problem?. *ACM Sigmod Blog*, 2014. wp.sigmod.org/?p=1075.

[30] T. Malik, R. Burns and N. Chawla. A Black-Box Approach to Query Cardinality Estimation. *CIDR*, 2007.

[31] R. Marcus and O. Papaemmanouil. Towards a Hands-Free Query Optimizer through Deep Learning. *CIDR*, 2019.

[32] V. Markl, V. Raman, D. Simmen, G. Lohman, H. Pirahesh and M. Cilimdzic. Robust query processing through progressive optimization. *SIGMOD*, 2004.

[33] G. Moerkotte, T. Neumann and G. Steidl. Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors. *PVLDB*, 2(1), 2009.

[34] T. Neumann and B. Radke. Adaptive Optimization of Very Large Join Queries. *SIGMOD*, 2018.

[35] K. Ramachandra, K. Park, K. Emani, A. Halverson, C. Galindo-Legaria and C. Cunningham. Froid: Optimization of Imperative Programs in a Relational Database. *PVLDB*, 11(4), 2017.

[36] W. Rodiger, S. Idicula, A. Kemper and T. Neumann. Flow-join: Adaptive skew handling for distributed joins over high-speed networks. *ICDE*, 2016.

[37] K. Tzoumas, A. Deshpande and C. Jensen. Efficiently adapting graphical models for selectivity estimation. *VLDBJ*, 22(1), 2013.

[38] J. Wiener, H. Kuno and G. Graefe. Benchmarking Query Execution Robustness. *TPCTC*, 2009.

[39] F. Wolf, M. Brendle, N. May, P. Willems, K. Sattler and M. Grossniklaus. Robustness Metrics for Relational Query Execution Plans. *PVLDB*, 11(11), 2018.

[40] W. Wu, Y. Chi, S. Zhu, J. Tatemura, H. Hacigumus and J. Naughton. Predicting query execution time: Are optimizer cost models really unusable? *ICDE*, 2013.

[41] W. Wu, Y. Chi, H. Hacigumus and J. Naughton. Towards predicting query execution time for concurrent and dynamic databae workloads. *PVLDB*, 6(10), 2013.

[42] W. Wu, X. Wu, H. Hacigumus and J. Naughton. Uncertainty Aware Query Execution Time Prediction. *PVLDB*, 7(14), 2014.

[43] W. Wu, J. Naughton and H. Singh. Sampling-based Query Re-optimization. *SIGMOD*, 2016.