Robust Query Processing: Mission Possible

Jayant R. Haritsa
Database Systems Lab
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.

PVLDB Reference Format:
Jayant R. Haritsa. Robust Query Processing: Mission Possible. *PVLDB*, 13(12): 3425-3428, 2020.
DOI: https:/doi.org/10.14778/3415478.3415561

1. 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.

In spite of this sustained research, the unfortunate reality is that 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?"*! [32]. Similar sentiments have been expressed by other academic and industrial database experts as well [13, 10].

Proceedings of the VLDB Endowment, Vol. 13, No. 12
ISSN 2150-8097.
DOI: https:/doi.org/10.14778/3415478.3415561

Due to the above-mentioned 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 the performance degradation – the total cost of ownership is significantly increased due to over-provisioning, lost efficiency, and increased human administrative costs [42].

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 past decade [1, 2, 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.

2. TUTORIAL CONTENTS

The definition of robustness has itself 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. [41]), *statistical* (e.g. [45]) or *learning-based* (e.g. [33]) approaches to incorporate the robustness features at these various levels.

We will cover representative techniques along these various dimensions in the tutorial. The big picture is that a rich variety of possibilities are currently available, and a judicious selection could 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 six stages, whose contents are summarized below.

2.1 Stage I: Robust Operators

A fertile area of RQP 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** [17], 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 scalable distributed joins. A novel approach to achieve this goal is **Flow-join** [39], wherein instead of constructing detailed statistics, a lightweight approach that trades off communication for computation is proposed. 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.

An innnovative approach for INL joins was proposed in [3], based on dynamic routing of individual tuples, reminiscent of Eddies [5]. Here, multiple plans proceed in parallel, and through a system of "back pressure", tuples are preferentially led away from inefficient plans towards efficient alternatives. Further, the execution is always making forward progress. 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.

2.2 Stage II: Robust Plans

We next consider techniques that aim to provide robustness at the granularity of entire plans. Contemporary optimizers typically approximate the distributions of run-time parameters with representative values -– 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 values encountered at run-time are significantly different from the representative values. Therefore, an alternative strategy proposed in [11] is to instead optimize for the "least expected cost" **(LEC)** plan, where the expectation is computed over the full distribution of the input parameters.

Determining 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, called **CostGreedy**, proposed in [18]. Here, the set of candidate plans is reduced to a small absolute number through plan 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 [19], that combines 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 optimizer's 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. Further, a particularly attractive feature is that the plan replacements never seriously harm, but often significantly help, with respect to the original plan choices.

2.3 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)**. This metric 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 [36]. 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 then went on to provide 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 results in small MSO values was presented in [14]. In this execution technique, called **PlanBouquet**, 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 [25]. Specifically, its MSO is bounded by $\mathbf{D^2 + 3D}$, where D is 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 selectivity space*.

2.4 Stage IV: Robust Cost Models

Apart from the cardinality model, robustness can also be adversely impacted by errors in the *operator cost model*, and this issue has been the subject of several studies during the past decade. 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 learning-based 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 [45] 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 such usages for a given query. In follow-up papers [44, 47], 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 additional 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 in **Froid** [38], implemented in SQL Server.

2.5 Stage V: Machine Learning Approaches

During the past few years, there has been an outpouring of papers advocating *machine learning-based* approaches to query processing (e.g. [16, 22, 23, 27, 28, 34, 37, 40, 48]). Most use deep neural network-based learning techniques, modeling the query optimization components as regression problems. Within this corpus, two broad classes have emerged – *query-based* and *data-based*. The former is an example of supervised learning, with models constructed by training on a large set of queries and leveraging the actual cardinalities observed during execution as the labels. On the other hand, the data-based techniques fall under unsupervised learning, and model the joint probability density functions of the underlying data to capture distributions and correlations.

We will cover an exemplar from each of these classes in the tutorial. Specifically, for the query-based class, we present **MSCN** (multi-set convolutional neural network) [28] where relations, joins and filters are represented as modules comprised of two-layer neural networks with shared parameters. The module outputs are averaged, then concatenated, and finally fed to an output network. The approach accurately predicts join-crossing correlations in the data and and addresses the inherent low-frequency weaknesses of traditional sampling-based estimation.

For the data-based class, we will overview **Naru** (neural relation understanding) [48], a cardinality estimator that leverages high-capacity deep autoregressive models in combination with Monte Carlo integration-based sampling techniques to efficiently and accurately handle the rich multivariate distributions of high-dimensional databases.

2.6 Stage VI: Future Research Directions

In the final stage of the tutorial, we will outline a set of open technical problems and future research directions, including:

Geometries of Plan Cost Functions. While prior work has only assumed monotonicity for plan cost functions, these functions often exhibit greater regularity in practice. 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 for Parametric Query Optimization (PQO). 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 – for instance, in [26], exponential reductions in overheads are obtained with linear relaxation in MSO guarantees.

Join-Graph-Sensitive Robustness. The robustness techniques developed in the literature 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 this information could perhaps 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.

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.

Robustness Benchmarks. A pre-requisite for confirming the robustness offered by new approaches are 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** [31], **JOB** [29] and **OTT** [46], but there remain several aspects of robustness that are yet to be covered.

3. TARGET AUDIENCE

Robust support for declarative query processing has been a long-standing concern for the database community, so we expect this tutorial to have widespread appeal among the VLDB 2020 attendees. 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.

Database researchers can expect to find the tutorial providing fresh and radical perspectives on a classical research topic, and serving to stimulate work on the further development of stable and efficient database engines. From the perspective of system developers and practitioners, the concepts and techniques presented in the tutorial can serve as potent mechanisms for the redesign of their systems. Finally, for database instructors and students, the coverage will help in comprehending and appreciating the complexities and subtleties of industrial-strength query processing, going far beyond the toy examples typically covered in a classroom setting.

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.

4. PRESENTER DETAILS

Jayant Haritsa is on the faculty of the Dept. of Computational & Data Sciences and the Dept. of Computer Science & Automation at the Indian Institute of Science, Bangalore. He received his undergraduate degree from IIT Madras, and the PhD degree from UW Madison. He is a Fellow of ACM and IEEE for contributions to the design and implementation of database engines. He was the Program Co-Chair for VLDB 2016 and ICDE 2010, and a recent Trustee of the VLDB Endowment.

Acknowledgments: This work was supported in part by a J. C. Bose Fellowship from Dept. of Science and Technology, Govt. of India, and a GCP grant from Google India.

5. REFERENCES

[1] Robust Query Processing. *Dagstuhl Seminar 10381*, 2010.

[2] Robust Query Processing. *Dagstuhl Seminar 12321*, 2012.

[3] Robust Performance in Database Query Processing. *Dagstuhl Seminar 17222*, 2017.

[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):521-545, 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): 11:1–11:37, 2016.

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

[16] A. Dutt, C. Wang, A. Nazi, S. Kandula, V. Narasayya and S. Chaudhuri. Selectivity estimation for range predicates using lightweight models. *PVLDB*, 12(9):1044-–1057, 2019.

[17] G. Graefe. New algorithms for join and grouping operations. *Computer Science – R&D*, 27(1):3–27, 2012.

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

[19] Harish, D., P. Darera and J. Haritsa. Identifying Robust Plans through Plan Diagram Reduction. *PVLDB*, 1(1):1124–1140, 2008.

[20] H. Harmouch and F. Naumann. Cardinality Estimation: An Experimental Survey. *PVLDB*, 11(4):499–512, 2017.

[21] S. Hasan, S. Thirumuruganathan, J. Augustine, N. Koudas and G. Das. Deep Learning Models for Selectivity Estimation of Multi-attribute Queries. *SIGMOD*, 2020.

[22] D. Havenstein, P. Lysakovski, N. May, G. Moerkotte and G. Steidl. Fast Entropy Maximization for Selectivity Estimation of Conjunctive Predicates on CPUs and GPUs. *EDBT*, 2020.

[23] R. Hayek and O. Shmueli. Improved Cardinality Estimation by Learning Queries Containment Rates. *EDBT*, 2020.

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

[25] S. Karthik, J. Haritsa, S. Kenkre, V. Pandit and L. Krishnan. Platform-independent Robust Query Processing. *IEEE TKDE*, 31(1):17–31, 2019.

[26] S. Karthik, J. Haritsa, S. Kenkre and V. Pandit. A Concave Path to Low-overhead Robust Query Processing. *PVLDB*, 11(13):2183-2195, 2018.

[27] M. Kiefer, M. Heimel, S. Bress and V. Markl. Estimating Join Selectivities using Bandwidth-Optimized Kernel Density Models. *PVLDB*, 10(13):2085–2096, 2017.

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

[29] 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):643-668, 2018.

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

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

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

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

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

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

[36] G. Moerkotte, T. Neumann and G. Steidl. Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors. *PVLDB*, 2(1):982–993, 2009.

[37] P. Negi, R. Marcus, H. Mao, N. Tatbul, T. Kraska and M. Alizadeh. Cost-Guided Cardinality Estimation: Focus Where it Matters. *IEEE ICDE Workshops*, 2020.

[38] 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):432-444, 2017.

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

[40] J. Sun and G. Li. An End-to-End Learning-based Cost Estimator. *PVLDB*, 13(3):307–319, 2019.

[41] K. Tzoumas, A. Deshpande and C. Jensen. Efficiently adapting graphical models for selectivity estimation. *VLDBJ*, 22(1):3–27, 2013.

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

[43] F. Wolf, M. Brendle, N. May, P. Willems, K. Sattler and M. Grossniklaus. Robustness Metrics for Relational Query Execution Plans. *PVLDB*, 11(11):1360–1372, 2018.

[44] W. Wu, Y. Chi, H. Hacigumus and J. Naughton. Towards predicting query execution time for concurrent and dynamic database workloads. *PVLDB*, 6(10):925–936, 2013.

[45] 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.

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

[47] W. Wu, X. Wu, H. Hacigumus and J. Naughton. Uncertainty Aware Query Execution Time Prediction. *PVLDB*, 7(14):1857–1868, 2014.

[48] Z. Yang et al. Deep Unsupervised Selectivity Estimation. *PVLDB*, 13(3):279–292, 2019.