CliffGuard: Towards Workload-Resilient Database Designs

Barzan Mozafari (mozafari@umich.edu)

ABSTRACT

A fundamental problem in database systems is choosing *the best physical design*, i.e., a small set of auxiliary structures that enable the fastest execution of future queries. Almost all commercial databases come with designer tools that create a number of indices or materialized views (together comprising the *physical design*) that they exploit during query processing. Existing designers are what we call *nominal*; that is, they assume that their input parameters are precisely known and equal to some nominal values. For instance, since future workload is often not known *a priori*, it is common for these tools to optimize for past workloads in hopes that future queries and data will be similar. In practice, however, these parameters are often noisy or missing. Since nominal designers do not take the influence of such uncertainties into account, they find designs that are sub-optimal and remarkably brittle. Often, as soon as the future workload deviates from the past, their overall performance falls off a cliff, leading to customer discontent and expensive re-designs. Thus, we propose a new type of database designer that is *robust* against parameter uncertainties, so that overall performance degrades more gracefully when future workloads deviate from the past. Users express their risk tolerance by deciding on how much nominal optimality they are willing to trade for attaining their desired level of robustness against uncertain situations. To the best of our knowledge, this project is the first to adopt the recent breakthroughs in the *theory of robust optimization* to build a practical framework for solving some of the most fundamental problems in databases, replacing today's brittle designs with a principled world of robust designs that can guarantee predictable and consistent performance.

1. INTRODUCTION

Database management systems are among the most mission-critical software components in our world today. Many important applications across enterprise, science, and government depend on database technology to derive insight from their data and make timely decisions. To fulfill this crucial role, a database (or its administrator) must make many important decisions on how to provision and tune the system in order to deliver the best performance possible, such as which materialized views, indices, or samples to build. While these auxiliary structures can significantly improve performance, they also incur storage and maintenance overheads. For example, most practical budgets only allow for building a handful of indices and a dozen materialized views out of an exponential number of possible structures (e.g., for a data-warehouse with 100 columns, there are $O(2^{100})$ different projections to choose from). Thus, a fundamental database problem is finding the best *physical design*; that is, finding a set of indices and/or materialized views that optimizes the performance of future queries.[1]

[1]Physical design can also refer to the entire process of translating a logical data model into a set of SQL statements defining the database. Here, by (physical) design we only mean the non-trivial part of this process, i.e., choosing an optimal set of indices and materialized views [1].

Modern databases come with designer tools (a.k.a. auto-tuning tools) that take certain parameters of a target workload (e.g., queries, data distribution, and various cost estimates) as input, and then use different heuristics to search the design space and find an optimal design (e.g., a set of indices or materialized views) within their time and storage budgets. However, these designs are (at best) only optimal for the input parameters provided to the designer. Unfortunately, in practice, these parameters are subject to many sources of uncertainty, such as noisy environments, approximation errors (e.g., in the query optimizer's cost or cardinality estimates [3]), and missing or time-varying parameters. Most notably, since future queries are unknown, these tools usually optimize for past queries in *hopes* that future ones will not be too different.

Existing designer tools (e.g., Tuning Wizard in Microsoft SQL Server 2000 [2], IBM DB2's Design Advisor [29], SQL Tuning Adviser in Oracle 10g [13], Vertica's DBD [20, 25, 27], and Parinda for Postgres [21]) do not take into account the *influence of such uncertainties* on the optimality of their design, and therefore, produce designs that are *sub-optimal* and *remarkably brittle*. We call all these existing designers ***nominal***. That is, all these tools assume that their input parameters are precisely known and equal to some nominal values. As a result, overall performance often plummets as soon as future workload deviates from the past (say, due to the arrival of new data or a shift in day-to-day queries). These dramatic performance decays are severely disruptive for time-critical applications. They also waste critical human and computational resources, as dissatisfied customers request vendor inspections, often resulting in re-tuning/redesigning the database to restore the required level of performance.

Our Goal — To overcome the shortcomings of nominal designers, we propose a new type of designers that are immune to parameter uncertainties as much as desired; that is, they are ***robust***. Our robust designer gives database administrators a *knob* to decide exactly how much nominal optimality to trade for how much robustness. For instance, users may demand a set of optimal materialized views with an assurance that they must remain robust against change in their workload of up to 30%. A more conservative user may demand a higher degree of robustness, say 60%, at the expense of less nominal optimality. Robust designs are highly superior to nominal ones, as:

(a) Nominal designs are inherently brittle and subject to performance cliffs, while the performance of a robust design will degrade *more gracefully*.

(b) By taking uncertainties into account, robust designs can guard against worst-case scenarios, delivering more consistent and predictable performance to time-sensitive applications.

(c) Since databases are highly non-linear and complex (and possibly non-convex) systems, a workload can have more than one optimal design. Thus, it is completely conceivable that a robust design may be nominally optimal as well (see [5, 6] for such examples in other domains).

(d) A robust design can significantly reduce operational costs by requiring less frequent database redesigns.

Previous Approaches — There has been some pioneering work on incorporating parameter uncertainties in databases [3, 10, 12, 15, 22, 24]. These techniques are specific to run-time query optimization and do not easily extend to physical designs. Other heuristics have been proposed to improve physical design through workload compression (i.e., omitting workload details) [9, 19] or modifying the query optimizer to return richer statistics [16]. Unfortunately, these approaches are not principled and thus do not necessarily guarantee robustness. To avoid these limitations, adaptive indexing schemes such as Database Cracking [18, 17] take the other extreme by completely ignoring the past workload in deciding which indices to build; instead of an *offline* design, they incrementally create and refine indices as queries arrive, *on demand*. However, these techniques are admittedly [17] more suitable for in-memory query processing and can cause many disk writes for disk-based processing (e.g., datasets that do not fit in memory). Also, despite their many merits, on-demand and continuous physical re-organizations are not acceptable in many applications, which is why nearly all commercial databases still rely on their offline designers. (We discuss the merits of previous work in Section **??**.) Instead of completely relying on past workloads or abandoning the offline physical design, in this talk we present a principled framework for directly maximizing robustness, which enables users to decide on the extent to which they want to rely on past information, and the extent of uncertainty they want to be robust against.

Our Approach — Recent breakthroughs in Operations Research on robust optimization (RO) theory have created new hopes for achieving robustness and optimality in a principled and tractable fashion [5, 6, 11, 28]. In this talk, we present the first attempt at applying RO theory to building a practical framework for solving one of the most fundamental problems in databases, namely finding the best physical design. Developing this robust framework is a departure from the traditional way in which databases are designed and tuned: from today's brittle designs to a principled world of robust designs that guarantee predictable and consistent performance.

Contributions — In this talk, we present these contributions:

- We formulate the problem of robust physical design using RO theory.
- We design a principled algorithm, called `CliffGuard`, by adapting the state-of-the-art framework for solving non-convex RO problems. `CliffGuard`'s design is generic and can potentially work with any existing designers and databases without modifying their internals.
- We implement and evaluate our designer using the Vertica database, on 2 synthetic workloads as well as a real workload of 430+K queries issued by the largest customer of a major database vendor over a 1-year period.

In summary, compared to Vertica's state-of-the-art commercial designer [20, 27], our robust designer reduces the average and maximum latency of queries on average by $4\times$ and $7\times$ (and up to $5\times$ and $11\times$), respectively.

2. REFERENCES

[1] S. Agrawal and et. al. Automated selection of materialized views and indexes in sql databases. In *VLDB*, 2000.

[2] S. Agrawal and et. al. Materialized view and index selection tool for microsoft sql server 2000. In *SIGMOD*, 2001.

[3] B. Babcock and S. Chaudhuri. Towards a robust query optimizer: A principled and practical approach. In *SIGMOD*, 2005.

[4] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust optimization*. Princeton University Press, 2009.

[5] D. Bertsimas and et. al. Theory and applications of robust optimization. *SIAM*, 53, 2011.

[6] D. Bertsimas, O. Nohadani, and K. M. Teo. Robust nonconvex optimization for simulation-based problems. *Operations Research*, 2007.

[7] D. Bertsimas and M. Sim. The price of robustness. *Operations research*, 52(1):35–53, 2004.

[8] J. R. Birge and et. al. Improving thin-film manufacturing yield with robust optimization. *Applied optics*, 50, 2011.

[9] S. Chaudhuri, A. K. Gupta, and V. Narasayya. Compressing sql workloads. In *SIGMOD*, 2002.

[10] S. Chaudhuri, H. Lee, and V. R. Narasayya. Variance aware optimization of parameterized queries. In *SIGMOD*, 2010.

[11] X. Chen, M. Sim, and P. Sun. A robust optimization perspective on stochastic programming. *Operations Research*, 55, 2007.

[12] F. Chu, J. Y. Halpern, and P. Seshadri. Least expected cost query optimization: An exercise in utility. In *PODS*, 1999.

[13] B. e. Dageville. Automatic sql tuning in oracle 10g. In *VLDB*, 2004.

[14] K. Deb. Geneas: A robust optimal design technique for mechanical component design. pages 497–514, 1997.

[15] D. Donjerkovic and R. Ramakrishnan. Probabilistic optimization of top n queries. In *VLDB*, 1999.

[16] K. E. Gebaly and A. Aboulnaga. Robustness in automatic physical database design. In *Advances in database technology*, 2008.

[17] F. Halim and et. al. Stochastic database cracking: Towards robust adaptive indexing in main-memory column-stores. *PVLDB*, 5, 2012.

[18] S. Idreos, M. L. Kersten, and S. Manegold. Database cracking. In *CIDR*, 2007.

[19] A. C. Konig and S. U. Nabar. Scalable exploration of physical database design. In *ICDE*, 2006.

[20] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandiver, L. Doshi, and C. Bear. The vertica analytic database: C-store 7 years later. *Proceedings of the VLDB Endowment*, 5(12):1790–1801, 2012.

[21] C. Maier and et. al. Parinda: an interactive physical designer for postgresql. In *ICEDT*, 2010.

[22] V. Markl and et. al. Robust query processing through progressive optimization. In *SIGMOD*, 2004.

[23] G. Palermo, C. Silvano, and V. Zaccaria. Robust optimization of soc architectures: A multi-scenario approach. In *Embedded Systems for Real-Time Multimedia, 2008. ESTImedia 2008. IEEE/ACM/IFIP Workshop on*, pages 7–12. IEEE, 2008.

[24] V. Raman and et. al. Constant-time query processing. In *ICDE*, 2008.

[25] A. RASIN and et. al. Automatic vertical-database design, Feb. 8 2008. WO Patent.

[26] J. Tu, K. K. Choi, and Y. H. Park. A new study on reliability-based design optimization. *Journal of Mechanical Design*, 121(4):557–564, 1999.

[27] R. Varadarajan, V. Bharathan, A. Cary, J. Dave, and S. Bodagala. Dbdesigner: A customizable physical design tool for vertica analytic database. In *ICDE*, pages 1084–1095, March 2014.

[28] Y. Zhang. General robust-optimization formulation for nonlinear programming. *JOTA*, 132, 2007.

[29] D. C. Zilio and et. al. Recommending materialized views and indexes with the ibm db2 design advisor. In *ICAC*, 2004.