A Case for Database Engine Research

Jayant R. Haritsa

(Invited Paper)

1 Introduction

Research related to database systems has a rich history, going back to almost the origins of computing itself, and the fruits of this research have been an organic part of the computer science curriculum for over four decades now. In the 1960s and 70s, the focus was on developing expressive data models, with the relational model emerging triumphant as the workhorse of enterprise data processing. The following decade of the 80s saw the transaction concept gaining centrestage, the emphasis being on developing efficient mechanisms to provide the powerful ACID[1] semantics that were the hallmark of this concept. Included in its ambit were data recovery mechanisms, concurrency control techniques, indexing strategies, and memory management. Concurrently, the automated identification of efficient execution strategies for declarative query processing gained tremendous ground through the development of dynamic-programming based techniques for navigating the exponentially large strategy search space.

The common thread among these various efforts was that they dealt with aspects of the database *engine*, that is, the components that constitute the database kernel. However, over the last two decades, and especially so in recent years, the research focus of the international community has largely shifted ground to the *middleware*, encompassing areas such as data mining, data warehousing, information retrieval, document processing, knowledge management and bio-informatics. In these domains, the database engine is essentially viewed as a black box that merely functions as a efficient data supplier to the middleware. Certainly the importance and impact of the newfangled topics cannot be discounted or disputed. However, what is worrisome is that the movement away from engine issues has begun to assume alarming proportions, placing in jeopardy the long-term future of database systems.

As a case in point, even a casual survey of the topics on which articles are submitted to this journal, shows that India has been equally susceptible to this unfortunate international trend! Papers on middleware subjects such as soft computing, data mining, web services and so on, are submitted in an unending flood, but manuscripts related to engine concerns are conspicuous by their almost total absence. In fact, the situation is probably worse in India because of a computer science curriculum, and a software industry, that thrive almost entirely on middleware, applications and services. Barring a few elite academic institutions, engine aspects are given no more than cursory attention in the vast multitude of engineering institutes that dot our countryside.

Perhaps a primary reason for middleware scoring over "engine-ware", so to speak, is that there are significantly fewer barriers to entry – usually, a good background in mathematics, data structures and algorithms suffices for being able to comprehend, assimilate and contribute to these areas. Whereas, in contrast, engine-ware requires considerable grounding in not only database systems, but also the underlying platforms with regard to computer

• *Jayant Haritsa is with the Database Systems Lab, Indian Institute of Science, Bangalore 560012.*
E-mail: haritsa@dsl.serc.iisc.ernet.in

1. Atomicity, Consistency, Isolation, Durability

architecture and operating systems, apart from an in-depth understanding of complex historical concepts and physical implementations. Further, a considerable body of prior research literature needs to be assimilated before venturing into working on these topics.

While the above-mentioned aspects of engine-related research may certainly deter apprehensive young minds, we would claim that the potential impact of novel engine ideas is substantively more since the functionalities and performance benefits of the new technologies are automatically bestowed on all applications. Therefore, aspiring database researchers may find it well worth their while to stay the arduous course and contribute their influential mite to the rich engine tradition that has been established during the last half-century.

Another, more pernicious, reason for the downslide in engine research could be the widespread, and utterly wrong, perception that engine design is essentially a "finished art" wherein all the major problems have already been solved, leaving little scope to make meaningful fresh contributions. However, nothing could be farther off the mark – while engine problems are certainly classical, they are equally amenable for fresh investigations either because: (a) the underlying platforms are changing, invalidating long-standing design assumptions in the process, or (b) novel design and analysis techniques have appeared on the scene, delivering new perspectives and solution methodologies.

Finally, yet another popular myth is that engine-ware is all about nuts-and-bolts programming and grungy implementation details – again, the truth is very different. Database engine design draws richly on all branches of computer science, including complexity theory, data structures, algorithms, statistics and experimental methodologies. This is borne out by several computer science legends, including names like Jeffrey Ullman, Christos Papadimitriou and Abraham Silberschatz, having deeply influenced the field – in fact, it has even produced its own Turing award winners: Edgar Codd in 1981 for the relational model, and James Gray in 1999 for the transaction concept.

In the remainder of this article, we will attempt to present concrete examples of how database engine design provides a rich source for challenging research problems and impactful contributions. We begin with a short overview of an imminent architectural change – *phase change memory* – that could well turn out to be an inflection point in the continuing saga of database engine design. This is followed by a detailed description of work executed in our lab over the last few years wherein a potent visual metaphor – *plan diagrams* – is brought to bear on characterizing the behavior of declarative query optimizers. In the process, serious design lacunae that are manifest even in the best of today's commercial database systems are highlighted, and the application of computer science fundamentals to resolve them is demonstrated. We conclude by showing how the plan diagram metaphor can be theoretically and practically leveraged for robustly addressing a fundamental estimation problem that has plagued database developers for several decades.

2 Phase Change Memory

A new memory technology, based on *chalcogenide glass*, wherein the physical condition of the material – *amorphous* or *crystalline* – is used to indicate a 0 or 1 binary state, has begun to see the light of day in recent times. This phase change memory, or PCM, as it is more commonly known [16], occupies the middle ground between traditional DRAM and traditional hard disks – specifically, PCM offers large-scale persistent storage like hard disks, but with random access and transfer speeds that are closer to DRAM. A summary view, sourced from [3], which succinctly captures PCM's characteristics as compared to prior memory technologies, including indicative parameter values, is shown in Table 1.

Given this new layer in the memory hierarchy, the entire design of the database engine needs to be revisited to ensure that the benefits are realized by the applications running on the system. As a case in point, an idiosyncratic feature of PCM is that each write to a memory cell causes "wear and tear", resulting in the

	DRAM	PCM	NAND Flash	HDD
Read energy	0.8 J/GB	1 J/GB	1.5 J/GB	65 J/GB
Write energy	1.2 J/GB	6 J/GB	17.5 J/GB	65 J/GB
Idle power	100 mW/GB	1 mW/GB	1-10 mW/GB	10 W/TB
Endurance	∞	10^6 – 10^8	10^4 – 10^5	∞
Page size	64 B	64 B	4KB	512 B
Page read latency	20–50ns	50ns	25 μs	5 ms
Page write latency	20–50ns	1 μs	500 μs	5 ms
Write bandwidth	GB/s	50-100 MB/s	5-40 MB/s	200 MB/s
Erase latency	N/A	N/A	2 ms	N/A
Density	1 X	2 – 4 X	4 X	N/A

TABLE 1
PCM Comparative Characteristics [3]

lifetime of an individual cell being limited to a few million writes (see the *Endurance* row in Table 1). Therefore, an interesting research problem from a design perspective is how to reengineer the basic database operators – for example, sorting – such that they consciously become read-intensive as opposed to the heavily write-based algorithms (e.g. quicksort) that are the norm in current implementations. A careful tradeoff has to be established in the new regime between access performance and cell longevity. But just reducing the number of writes may not be enough, it may also become necessary to explicitly implement "wear leveling" to evenly spread the writes across the entire memory – this may require copying of data from one location to another simply to ensure that no individual location becomes too worn out.

A related distinctive feature of PCM is that the energy consumption for writes is substantially higher than that for reads (see the *Read energy* and *Write energy* rows in Table 1) – this serves as yet another motivation for reducing the number of writes to the minimum possible, given the growing clamour for developing "green" computing devices.

A first attempt at a PCM-conscious redesign of the database system internals is described in [3], and innovative ideas have been proposed. But there is still a long way to go, and hard research challenges to be addressed, before the new technology can seamlessly become an integral part of database product offerings.

3 Database Query Optimizers

Queries to database systems are usually expressed in the Structured Query Language (SQL) [18]. A particularly appealing feature of this language is that it is "declarative", meaning that the user only states *what* is wanted, without having to specify the *procedure* for obtaining the information.

To make the declarative notion concrete, consider the sample university database schema shown in Figure 1. Here, information is maintained in three relations: STUDENT, COURSE and REGISTER, which tabulate data about students, courses, and the course registrations of students, respectively. The user's goal is to extract the names of the students and the courses for which they are registered, and an SQL query that achieves this goal is shown in Figure 2, where the desired information is obtained by combining the data across the three tables, using the roll numbers and the course numbers as the connectors. Note that in this formulation, the *sequence* in which the tables are combined ((STUDENT $\bowtie$ COURSE) $\bowtie$ REGISTER, or (REGISTER $\bowtie$ COURSE) $\bowtie$ STUDENT, etc.), as well as the *mechanism* to be used for each combination (NESTED-LOOPS JOIN, SORT-MERGE JOIN, HASH JOIN, etc.) is left unspecified, resulting in the declarative tag.

3.1 Query Optimization

Given a generic SQL query that requires combining information across n relations, there are in principle $n!$ different permutations of the

STUDENT	RollNo	StudentName	Address	Program
COURSE	CourseNo	CourseName	Credits	Content
REGISTER	RollNo	CourseNo		

Fig. 1. **University Database Schema**

```
select StudentName, CourseName
from STUDENT, COURSE, REGISTER
where STUDENT.RollNo = REGISTER.RollNo and
      REGISTER.CourseNo = COURSE.CourseNo
```

Fig. 2. **User SQL Query**

combination sequence, implying that the strategy search space is at least *exponential* in the query size. The automated identification of an efficient procedure or strategy from this search space is the responsibility of an internal DBMS component called the "query optimizer". The efficiency of these strategies, called "plans", is usually costed in terms of the estimated query response time. Optimization is a mandatory exercise since the difference between the cost of the best plan and a random choice could be in orders of magnitude, but is computationally extremely expensive due to the combinatorially large search space of plan alternatives, as explained above. The role of query optimizers has become especially critical in recent times due to the high degree of query complexity characterizing current decision-support applications, as exemplified by the industry-standard TPC-H and TPC-DS performance benchmarks [19], [20].

Plans are typically comprised of a *tree* of data processing operators that are logically evaluated in a bottom-up paradigm. A sample plan is shown in Figure 3 for the example query of Figure 2, where the STUDENT and REGISTER relations are first combined with a NESTED-LOOPS JOIN operator, and this intermediate result is then combined with the COURSE relation using a HASH JOIN operator. The bracketed numbers within each operator node indicate the estimated aggregate processing costs incurred until this stage in the bottom-up query evaluation.

The design of effective query optimizers that

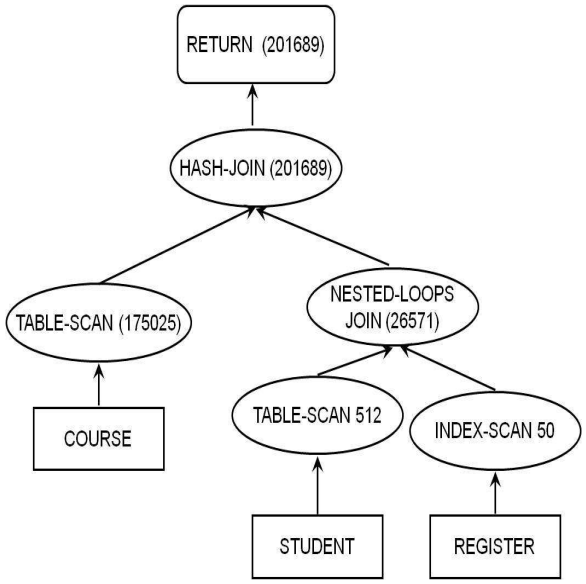


Fig. 3. **Sample Plan**

quickly identify low cost plans has been diligently addressed by the database research community over the last few decades [2]. However, due to its inherent complexities and challenges, this area has largely remained a "black art", and the quality of the query optimizer continues to be a key differentiator between competing database products, with large R & D teams involved in their design and implementation. Over the past few years, a fresh perspective has been brought to bear on the behavior of modern query optimizers through the introduction and development of the **"plan diagram"** concept. A plan diagram is a *visual* representation of the plan choices made by the optimizer over a parameter space, and is generated by

leveraging the optimizer's API functions. In the remainder of this section, we provide an overview of plan diagrams, their processing, and their applications – this material is largely sourced from [7].

3.2 Plan Diagrams

A query optimizer's execution plan choices, for a given database and system configuration, are primarily a function of the *selectivities* of the base relations featuring in the query. The selectivity of a relation is the estimated fraction of rows of the relation that are relevant to producing the final result. In [13], plan diagrams were introduced to denote color-coded pictorial enumerations of the plan choices of the optimizer for parametrized SQL query templates over the relational selectivity space. For example, consider QT8, the parametrized two-dimensional query template shown in Figure 4, based on Query 8 of the TPC-H benchmark (the query determines the market share of Brazil within the American continent for cheap anodized steel parts). The template has selectivity variations on the SUPPLIER and LINEITEM relations through the s_acctbal :varies and l_extendedprice :varies predicates, which apply one-sided range constraints on the supplier's account balance and the extended price of the lineitem, respectively (e.g. s_acctbal < 1000 and l_extendedprice < 2000).

The associated plan diagram for QT8 is shown in Figure 5(a) – this picture was produced on a commercial database engine using the Picasso visualization software tool [17]. In this picture, a set of 89 different optimal plans, P1 through P89, cover the selectivity space. The value associated with each plan in the legend indicates the percentage area covered by that plan in the diagram – P1, for example, covers about 22% of the space, whereas P89 is chosen in only 0.001% of the space. In a nutshell, plan diagrams visually capture the geometries of the optimality regions of the *parametric optimal set of plans* (POSP) [10].

It is vividly evident from Figure 5(a) that plan diagrams can be extremely complex and dense, with a large number of plans covering the space – several such instances spanning a representative set of query templates over a suite of industrial-strength optimizers, are available at [17]. In fact, they often appear similar to *cubist paintings* [15], with a variety of intricate tessellated patterns, including *speckles*, *stripes*, *blinds*, *mosaics* and *bands*, in the diagrams! Further, the boundaries of the plan optimality regions can be highly irregular, which seem to indicate the presence of strongly non-linear and discretized cost models. Finally, the diagrams also demonstrate that the basic mathematical assumptions – *plan convexity*, *uniqueness* and *homogeneity* – underlying the rich body of research literature on parametric query optimization (e.g. [10], [11]), rarely hold in practice.

While *individual* queries have been analyzed in great detail in the past, plan diagrams initiated the characterization and investigation of the behavior of a set of queries over a *parameter space* in an industrial-strength environment. Therefore, in spite of query optimization having been studied for several decades, the discovery of the above-mentioned complex patterns has proved to be rather surprising and thought-provoking for the database research community.

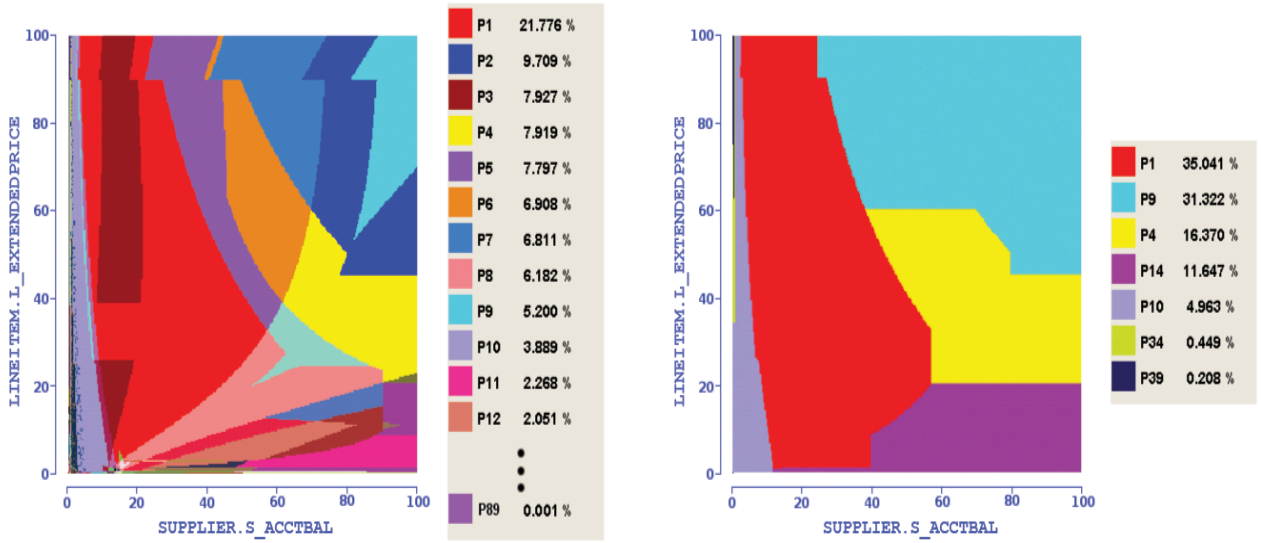
Plan diagrams are currently in vogue at various industrial and academic sites for a diverse set of applications including analysis of existing optimizer designs; visually carrying out optimizer regression testing; debugging new query processing features; comparing the behavior between different optimizer versions; investigating the structural differences between neighboring plans in the space; evaluating the variations in the plan choices made by competing optimizers; etc. As a case in point, visual examples of *non-monotonic* cost behavior in commercial optimizers, potentially indicative of modeling errors, were highlighted in [13].

3.3 Anorexic Reduction of Plan Diagrams

The next phase of our investigation showed that dense plan diagrams could typically be "reduced" to much simpler pictures featuring significantly fewer plans, *without materially degrading the processing quality of any individual query*. For example in Figure 5(a), if users are

```
select o_year, sum(case when nation = 'BRAZIL' then volume else 0 end)
       / sum(volume)
from
       (select YEAR(o_orderdate) as o_year, l_extendedprice * (1 -
              l_discount) as volume, n2.n_name as nation
       from part, supplier, lineitem, orders, customer, nation n1,
              nation n2, region
       where p_partkey = l_partkey and s_suppkey = l_suppkey and
              l_orderkey = o_orderkey and o_custkey = c_custkey
              and c_nationkey = n1.n_nationkey and n1.n_regionkey =
              r_regionkey and s_nationkey = n2.n_nationkey and r_name
              = 'AMERICA' and p_type = 'ECONOMY ANODIZED STEEL' and
              s_acctbal :varies and l_extendedprice :varies
       ) as all_nations
group by o_year
order by o_year
```

Fig. 4. **Example Query Template (QT8)**



(a) **Plan Diagram** (b) **Reduced Diagram (λ = 10%)**

Fig. 5. **Sample Plan Diagram and Reduced Plan Diagram (QT8)**

willing to tolerate a minor cost increase, denoted by λ, of at most 10% for any query in the diagram, relative to its original cost, the picture could be reduced to Figure 5(b), where only 7 plans remain – that is, most of the original plans have been "completely swallowed" by their siblings, leading to a highly reduced plan cardinality.

A detailed study of the plan diagram reduction problem from both theoretical and empirical perspectives was presented in [8]. The analysis first showed that finding the optimal (wrt minimizing the number of plans) reduced plan diagram is NP-Hard through a reduction from the classical Set Cover problem [5]. This result motivated the design of **CostGreedy**, a greedy heuristic algorithm whose complexity is $O(nm)$, where n is the number of plans and m is the number of query points in the diagram ($n \ll m$). Hence, for a given picture resolution, CostGreedy's performance scales *linearly* with the number of plans in the diagram. Further, from the reduction quality perspective, CostGreedy provides a tight performance guarantee

of $O(\ln m)$, which cannot be improved upon by any other deterministic algorithm.

Through experimental analysis on the plan diagrams produced by industrial-strength optimizers with multi-dimensional benchmark-based query templates, it was shown that plan reduction can be carried out efficiently. This is because attention is limited to only the set of plans appearing in the original plan diagram, making it unnecessary to revisit the optimizer's combinatorially large search space of plan alternatives. Further, it was found that the CostGreedy algorithm typically gives a near-optimal reduction or the optimal reduction itself.

Most importantly, these results demonstrated that a cost-increase threshold of *only 20 percent* is usually amply sufficient to bring down the *absolute* number of plans in the final reduced picture to *within or around ten*. In short, that complex plan diagrams can be made "anorexic" (small in absolute sense) while retaining acceptable query processing performance, even for high dimensional query templates.

Carrying out anorexic plan reduction on dense plan diagrams has a variety of useful implications for improving both the efficiency of the optimizer and the choice of execution plan. Its most important utility, as described in the following subsection, is that it supports the identification of plans that are *robust* to errors in selectivity estimates. Selectivity estimation errors are a chronic problem faced by database query optimizers over the past several decades, and arise due to a variety of reasons, including outdated statistics, attribute-value independence assumptions, and coarse summaries [14]. The goal in the anorexic approach is to identify *robust plans* that are relatively less sensitive to such selectivity errors. In a nutshell, to "aim for resistance, rather than cure", by identifying plans that provide comparatively good performance over large regions of the selectivity space. Such plan choices are especially important for industrial workloads where global stability is as much a concern as local optimality [12].

3.4 Selecting Robust Plans

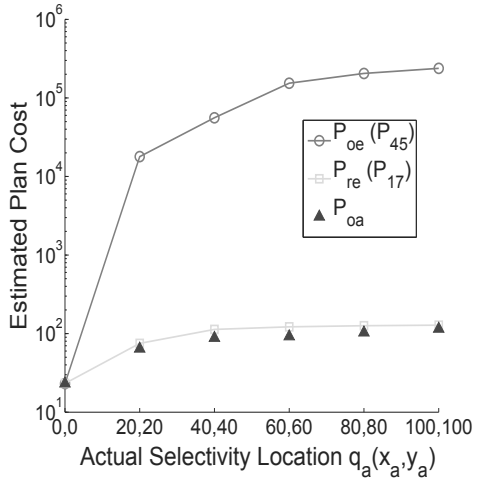
The selectivity error issue was investigated from both theoretical and empirical perspectives in [9]. Through extensive experimentation on a leading commercial optimizer with a rich suite of multi-dimensional query templates operating on a variety of logical and physical database schemas, it was demonstrated that *plan diagram reduction typically produces plan choices that substantially curtail the adverse effects of selectivity estimation errors*. Therefore, it clearly has the potential to improve performance in general, especially for errors that lie within the swallower's optimality region, i.e. its "endo-optimal" region.

Consider a query instance whose optimizer-estimated location in the selectivity space is q_e, and denote the optimizer's optimal plan choice at q_e by P_{oe}. Due to errors in the selectivity estimates, the *actual* location of q_e could be different at execution-time – denote this location by q_a, and the optimizer's optimal plan choice at q_a by P_{oa}. Assume that P_{oe} has been swallowed by a sibling plan during the reduction process and denote the replacement plan assigned to q_e by P_{re}.

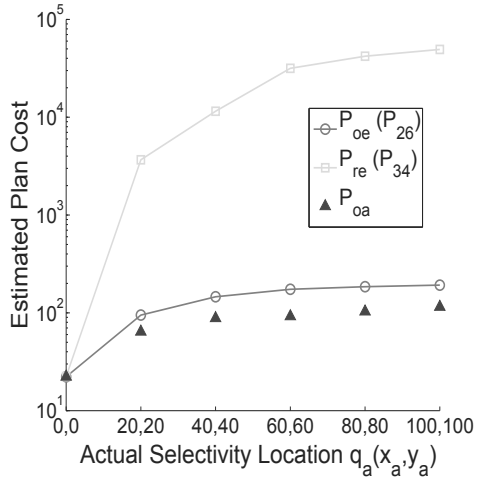
Replacement Benefits

Our first scenario, typical of that seen in most of the experiments, demonstrates how the replacement plan P_{re} can provide extremely substantial improvements *throughout the selectivity space*. The specific example chosen is a plan diagram obtained with a 2D query template based on TPC-H Q5, with selectivity variations on the CUSTOMER and SUPPLIER relations – the reduction was subsequently carried out with $\lambda = 10\%$. On this plan diagram, consider the estimated location $q_e = (0.36, 0.05)$ and a sample set of actual locations q_a – for instance, along the principal diagonal of S. For this scenario, the costs of P_{oe} (P45), P_{re} (P17) and P_{oa} (the optimal plan at each q_a location) are shown in Figure 6(a) – note that the costs are measured on a *log scale*.

It is clear from Figure 6(a) that the replacement plan P_{re} provides *orders-of-magnitude* benefit with respect to P_{oe}. In fact, the error-resistance is to the extent that it virtually pro-



(a) **Beneficial Impact**



(b) **Adverse Impact**

Fig. 6. **Impact of Plan Replacement**

vides "immunity" to the error since the performance of P_{re} is close to that of the *locally optimal plan* P_{oa} throughout the space – note that is in spite of the endo-optimal region of P_{re} constituting only a very small fraction of this space.

Replacement Problems

While performance improvements are usually the order of the day, occasional situations are also encountered wherein a replacement plan performs much worse in its exo-optimal region than the original optimizer choice, that is, where P_{re} performs worse than P_{oe} at q_a. A particularly egregious example, arising from the *same* plan diagram described above, is shown in Figure 6(b) for $q_e = (0.03, 0.14)$ – notice here that it is now the replacement plan P_{re} (P34), which is *orders-of-magnitude* worse than P_{oe} (P26) in the presence of selectivity errors.

The above example highlights the need to establish an efficient criterion of when a specific swallowing is *globally safe*, that is, within the λ-threshold throughout the space. To achieve this objective, a generalized mathematical model of the behavior of plan cost functions over the selectivity space was designed in [9]. The model, although simple, is sufficient to accurately capture the cost behavior of all plans that have arisen from the experimental query templates, and is the first such characterization for industrial-strength optimizers.

Using this model, the following powerful result was proved: Safety checks on only the *perimeter* of the selectivity space are sufficient to decide the safety of reduction over the *entire* space. These checks involve the costing of *"foreign plans"*, that is, of costing plans in their exo-optimal regions, a feature that has become available in the current versions of several industrial-strength optimizers. Apart from providing reduction safety, foreign-plan costing can be additionally leveraged to both (a) enhance the degree of reduction of the plan diagram, and (b) improve the complexity characteristics of the reduction process, as compared to the earlier CostGreedy algorithm.

Overall, the new approach called **SEER** (Selectivity-Estimate-Error-Resistance), provides an effective and safe mechanism for identifying robust plans that are resistant, as compared to the optimizer's original choices, to errors in the base relation selectivity estimates. Further, LiteSEER, an optimally-efficient light-weight heuristic version of SEER that very cheaply provides a high degree of safety by restricting its attention to only the *corners* of the selectivity space, has also been developed.

A particularly noteworthy aspect of these techniques is that their performance guarantees apply at the level of *individual queries*. This is in marked contrast to the *aggregate* basis of prior proposals in the literature, which made them difficult to use in practice. Further, since the optimizer is treated as a black-box, the SEER approach is inherently (a) completely non-

intrusive, and (b) capable of handling whatever SQL is supported by the system. Equally importantly, no additional information beyond that provided by the engine's API interface is expected.

Viewed in toto, the results presented in [9] indicate that a large percentage of optimizer choices over the parameter space can be improved through robust replacements.

3.5 Run-time Applications

Apart from aiding optimizer design, plan diagrams can also be used in *operational* settings. Specifically, since they identify the optimal set of compile-time plans, they can be used at run-time to immediately identify the best plan for the current query without going through the time-consuming optimization exercise. Further, they can prove useful to *adaptive* plan selection techniques [4], which, based on run-time observations, may dynamically choose to re-optimize the query and switch plans midway through the processing. In this context, plan diagrams can help to eliminate the re-optimization overheads incurred in determining the substitute plan choices. The reduced plan diagrams, on the other hand, help to both minimize the number of invocations of the re-optimization process, as well as the likelihood of requiring a plan switch after the re-optimization.

3.6 Online Robust Plans

The SEER algorithm is an *off-line* approach in that it uses prior knowledge of the POSP set of plans in order to make the replacements. In practice, however, we would ideally like to have the robustness feature to be organically integrated *within* the optimizer, rather than generated as a post-facto exercise, and by virtue of this integration, making it directly applicable to ad-hoc individual queries. This goal was achieved in [1] through an algorithm called **EXPAND**, which judiciously expands the candidate set of sub-plans that are retained at each node of the plan enumeration lattice during the core dynamic-programming exercise. That is, instead of merely forwarding the cheapest sub-plan from each node in the lattice, a *train* of sub-plans is sent, with the cheapest being the "engine", and stabler alternative choices being the "wagons". To ensure that the overheads of maintaining trains instead of engines are not impractically large, a four-stage pruning process that incorporates both cost and robustness aspects is used to ensure that only wagons that are plausible replacements for the engine are retained. The final plan selection is made at the root of the dynamic-programming lattice from amongst the set of complete plans available at this terminal node, subject to user-specified cost and stability criteria.

The Expand scheme has been incorporated in the kernel of the public-domain PostgreSQL database engine, and a variety of plan selection algorithms that cover a spectrum of design tradeoffs have been been implemented and evaluated on benchmark environments. The results have shown that a significant degree of robustness can be obtained with relatively minor conceptual changes to current optimizers, especially those supporting a foreign-plan-costing feature. Expand often delivers plan choices that eliminate more than *two-thirds of the performance gap* (between P_{oe} and P_{oa}) for a non-trivial number of error instances. Equally importantly, the replacement is almost never materially worse than the optimizer's original choice. In a nutshell, the replacement plans *"often help substantially, but never seriously hurt"* the query performance. Therefore, the Expand approach results in an intrinsically improved optimizer that directly and efficiently produces high-quality plan diagrams in a completely online fashion.

Summary

Overall, the primary message of this work is that, although it may appear unlikely at first glance, it is indeed feasible to *efficiently produce plan diagrams that simultaneously possess the highly desirable properties of being online, anorexic, safe and robust*. We expect that this result will have a significant impact on the design of next-generation database query optimizers.

4 Closing Remarks

Our objective here was to make the case to readers that a rich source of both conceptually challenging and practically relevant technical problems exists in the world of database engines. Notwithstanding this encouraging environment, the attention of the research community at large is focussed primarily on middleware and application issues. It is our fond hope that the current article may serve to catalyze attention towards database systems topics in the Indian research community, especially among the graduate students, and help engine study reclaim its former position of eminence in the research mainstream.

References

[1] M. Abhirama, S. Bhaumik, A. Dey, H. Shrimal and J. Haritsa, "On the Stability of Plan Costs and the Costs of Plan Stability", *PVLDB Journal*, 3(1), 2010.

[2] S. Chaudhuri, "An Overview of Query Optimization in Relational Systems", *Proc. of ACM Symp. on Principles of Database Systems (PODS)*, 1998.

[3] S. Chen, P. Gibbons, and S. Nath, "Rethinking Database Algorithms for Phase Change Memory", *Proc. of 5th Biennial Conf. on Innovative Data Systems Research (CIDR)*, 2011.

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

[5] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W H Freeman & Co, 1979.

[6] J. Haritsa, "The Picasso Database Query Optimizer Visualizer", *PVLDB Journal*, 3(2), 2010.

[7] J. Haritsa, "Plan Diagrams: Visualizing Database Query Optimizers", *Annals of Indian National Academy of Engineering (INAE)*, Volume VIII, 2011.

[8] D. Harish, P. Darera and J. Haritsa, "On the Production of Anorexic Plan Diagrams", *Proc. of 31st Intl. Conf. on Very Large Data Bases (VLDB)*, 2007.

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

[10] A. Hulgeri and S. Sudarshan, "Parametric Query Optimization for Linear and Piecewise Linear Cost Functions", *Proc. of 28th Intl. Conf. on Very Large Data Bases (VLDB)*, 2002.

[11] A. Hulgeri and S. Sudarshan, "AniPQO: Almost Non-intrusive Parametric Query Optimization for Nonlinear Cost Functions", *Proc. of 29th Intl. Conf. on Very Large Data Bases (VLDB)*, 2003.

[12] L. Mackert and G. Lohman, *R** Optimizer Validation and Performance Evaluation for Local Queries, *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, 1986.

[13] N. Reddy and J. Haritsa, "Analyzing Plan Diagrams of Database Query Optimizers", *Proc. of 31st Intl. Conf. on Very Large Data Bases (VLDB)*, 2005.

[14] M. Stillger, G. Lohman, V. Markl and M. Kandil, "LEO – DB2's LEarning Optimizer", *Proc. of 27th Intl. Conf. on Very Large Data Bases (VLDB)*, 2001.

[15] www.artlex.com.h/ArtLex/c/cubism.html

[16] en.wikipedia.org/wiki/Phase-change_memory

[17] dsl.serc.iisc.ernet.in/projects/PICASSO/picasso.html

[18] en.wikipedia.org/wiki/SQL:2008

[19] www.tpc.org/tpch

[20] www.tpc.org/tpcds

Jayant R. Haritsa is a professor of database systems in the Supercomputer Education & Research Centre and the Department of Computer Science & Automation at the Indian Institute of Science, Bangalore. He received the BTech degree in Electronics and Communications Engineering from the Indian Institute of Technology (Madras), and the MS and PhD degrees in Computer Science from the University of Wisconsin (Madison). He is a Fellow of IEEE, INAE, NASI and IASc, and is also a Distinguished Scientist of ACM. He is a recipient of the Swarnajayanti Fellowship, the Vikram Sarabhai Research Award, the Shanti Swarup Bhatnagar Prize, and the Distinguished Alumnus Award of IIT Madras.