Report from Dagstuhl Seminar 12321

Robust Query Processing

Edited by

Goetz Graefe[1], Wey Guy[2], Harumi A. Kuno[3], and Glenn Paulley[4]

1 **HP Labs, USA** goetz.graefe@hp.com
2 **USA** weyyuanguy@hotmail.com
3 **HP Labs, USA** harumi.kuno@hp.com
4 **Conestoga College, Kitchener, Ontario, Canada** gpaulley@acm.org

Abstract

The 2012 Dagstuhl 12321 Workshop on Robust Query Processing, held from 5–10 August 2012, brought together researchers from both academia and industry to discuss various aspects of robustness in database management systems and ideas for future research. The Workshop was designed as a sequel to an earlier Workshop, Dagstuhl Workshop 10381, that studied a similar set of topics. In this article we summarize some of the main discussion topics of the 12321 Workshop, the results to date, and some open problems that remain.

Seminar 09.–13. July, 2012 – www.dagstuhl.de/12321
1998 ACM Subject Classification H.2 Database Management, H.2.2 Physical Design, H.2.4 Systems, H.2.7 Database Administration
Keywords and phrases robust query processing, adaptive query optimization, query execution, indexing, workload management, reliability, application availability
Digital Object Identifier 10.4230/DagRep.2.8.1

1 Executive Summary

Goetz Graefe
Wey Guy
Harumi A. Kuno
Glenn Paulley

Introduction

In early August 2012 researchers from both academia and industry assembled in Dagstuhl at the 2012 Dagstuhl Workshop on Robust Query Processing, Workshop 12321. An earlier Workshop—Dagstuhl Workshop 10381—held in September 2010 [16] had supplied an opportunity to look at issues of Robust Query Processing but had failed to make significant progress in exploring the topic to any significant depth. In 2012, 12321 Workshop participants looked afresh at some of the issues surrounding Robust Query Processing with greater success and with the strong possibility of future publications in the area that would advance the state-of-the-art in query processing technology.

Background and related research

A considerable amount of query processing research over the past 20 years has focused on improving relational database system optimization and execution techniques for complex queries and complex, ever-changing workloads. Complex queries provide optimization challenges because selectivity and cardinality estimation errors multiply, and so there is a large body of work on improving cardinality estimation techniques and doing so in an autonomic fashion: from capturing histogram information at run time [1, 17], to mitigating the effects of correlation on the independence assumption [21], to utilizing constraints to bound estimation error [18, 15, 9, 10], to permitting various query rewritings to simplify the original statement [11, 23, 28, 27, 19, 26]. Studies of the feasibility of query re-optimization [8, 7], or deferring optimization to execution time [24], have until recently largely been based on the premise that the need for such techniques is due either to recovering from estimation errors at optimization time in the former case, or avoiding the problem entirely by performing all optimization on-the-fly, such as with Eddies [6] rather than in a staged, 'waterfall' kind of paradigm.

More recent work on adaptive query processing [13, 25, 14, 24] has considered techniques to handle the interaction of query workloads [3, 4, 5], coupled with the realization that changes to environmental conditions can significantly impact a query's chosen execution plan. These environmental conditions include:

- changes to the amount of memory available (buffer pool, heap memory);
- changes to I/O bandwidth due to concurrent disk activity;
- locking and other waits caused by concurrency control mechanisms;
- detected flaws in the currently executing plan;
- number of available CPU cores;
- changes to the server's multiprogramming level [2];
- changes to physical access paths, such as the availability of indexes, which could be created on the fly;
- congestion with the telecommunications network;
- contents of the server's buffer pool;
- inter-query interaction (contention on the server's transaction log, 'hot' rows, and so on.

Background – Dagstuhl seminar 10381

Self-managing database technology, which includes automatic index tuning, automatic database statistics, self-correcting cardinality estimation in query optimization, dynamic resource management, adaptive workload management, and many other approaches, while both interesting and promising, tends to be studied in isolation of other server components. At the 2010 Dagstuhl Workshop on Robust Query Processing (Dagstuhl seminar 10381) held on 19–24 September 2010, seminar attendees tried to unify the study of these technologies in three fundamental ways:

1. determine approaches for evaluating these technologies in the 'real' environment where these independently-developed components would interact;
2. establish a metric with which to measure the 'robustness' of a database server, making quantitative evaluations feasible so as to compare the worthiness of particular approaches. For example, is dynamic join reordering during query execution worth more than cardinality estimation feedback from query execution to query optimization?

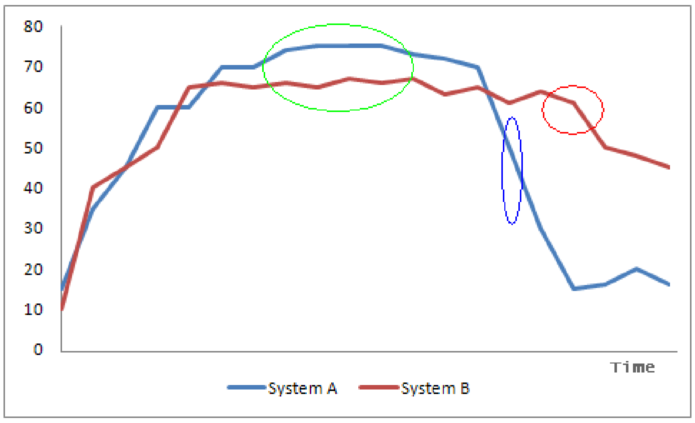

Figure 1 Comparison of Systems A and B in response to increasing workloads over time.

3. utilize a metric, or metrics, to permit the construction of regression tests for particular systems. The development of suitable metrics could lead to the development of a new, possibly industry-standard benchmark, that could be used to measure self-managing database systems by industry analysts, customers, vendors, and academic researchers and thus lead to better improvements in robust operation.

At the 2010 Dagstuhl seminar, attendees struggled somewhat with trying to define the notion of robustness, let alone trying to measure or quantify it. Robustness is, arguably, somewhat orthogonal to absolute performance; what we are trying to assess is a system's ability to continue to operate in the face of changing workloads, system parameters and environmental conditions.

An example of the sorts of problems encountered in trying to define robustness is illustrated in Figure 1. Figure 1 shows the throughput rates of two systems, System A (blue line) and System B (red line), over time, for the same workload. The Y-axis represents the throughput rate, and the X-axis is elapsed time. Over time, the workload steadily increases.

Three areas of the graph are highlighted in Figure 1. The first, in green, shows that as the workload is increased, System A outperforms System B by some margin. That peak performance cannot be maintained, however, as the load continues to be increased. The area in blue shows that once System A becomes overloaded, performance drops precipitously. On the other hand, System B shows a much more gradual degradation (circled in red), offering more robust behaviour than System A but with the tradeoff of not being able to match System A's peak performance.

One can argue that Figure 1 mixes the notions of query processing and workload management. In Figure 2 we simplify the problem further, and consider only simple range queries (using two columns) over a single table, where the (logarithmic) X-axis denotes the size of the result set.

In Figure 2, the yellow line illustrates a table scan: it is robust—it delivers identical performance over all result sets—but with relatively poor performance. The dashed red line is a traditional index-to-index lookup plan: that is, search in secondary index, row fetch out of the primary (clustered) index. This plan is considerably faster for very small selectivities, but becomes considerably poorer with only a marginal decrease in selectivity. The solid red line shows, in comparison, substantial-but-imperfect improvements over the index-to-index technique, due to asynchronous prefetch coupled with sorting batches of

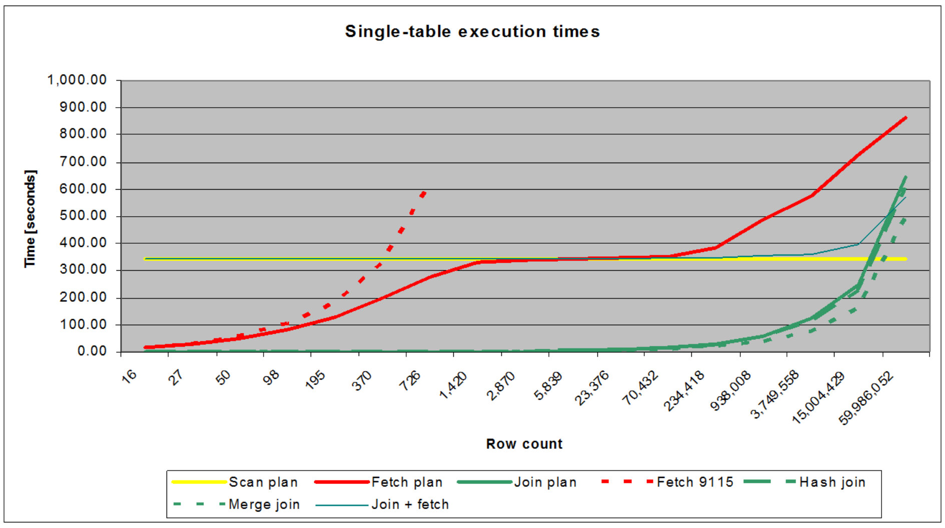

Figure 2 Comparison of access plans for a single table range query.

row pointers obtained from the secondary index. This query execution strategy is available in Microsoft SQL Server 2005. While Figure 2 is just one simple query—one table, range predicates on two columns—Figure 2 illustrates both the magnitude of the problem and the opportunity for improving the robustness of such plans.

At the 2010 Dagstuhl seminar, seminar attendees explored a number of different ways in which to define robustness. One idea was to define a metric for robustness as the accumulated variance in the wall clock times of workloads—or particular queries—or, alternatively, some measure of the distribution of that variance, a 2nd level effect. Since this working definition includes wall clock times, it implicitly includes factors such as optimizer quality, since a robustness metric such as this must include statement execution times. However, while the sophistication of query optimization, and the quality of access plans, is a component of a robust database management system, it is not the only component that impacts any notion of robustness.

This working definition of robustness raised as many questions as answers, and many of these were still unresolved by the end of the workshop. Those questions included:

- Sources of variance in query optimization include the statistics model, the cardinality model, and the cost model, with the latter usually being less critical in practice than the former two. One measure of 'robustness' is to assess the accuracy between estimates and actuals. What level of importance should the 'correctness' of a query optimizer have on a metric of robustness?
- Which offers more opportunity for disaster—and disaster prevention: robust query optimization or robust query execution?
- Is robustness a global property? Does it make sense to measure robustness at the component level? If so, which components, and what weight should be attached to each?
- Several of the attendees at the 2010 Dagstuhl workshop advocated a two-dimensional tradeoff between actual performance and 'consistency'. But what is 'consistency'? Is it merely plan quality, or something more?
- Robustness for who? Expectations are different between product engineers and end users; one should not try to define robustness unless one addresses whose expectations you are

trying to satisfy. Both rely on an idealized model of how a system should behave. Can we define that model? At the same time, what expectations can a user have of a really complex query?
- Is adaptivity the only way to achieve robustness?
- What would a benchmark for robustness attempt to measure?

During the workshop we analyzed these questions from various perspectives. Unfortunately we failed to reach consensus on a clear definition of robustness, how to measure it, and what sorts of tradeoffs to include. Our hope, in this, the second Dagstuhl workshop on Robust Query Processing, is to make additional progress towards clarifying the problems, and possibly make some progress towards defining some general—or specific—approaches to improve DBMS robustness.

2 Table of Contents

Executive Summary
Goetz Graefe, Wey Guy, Harumi A. Kuno, and Glenn Paulley 1

Dagstuhl seminar 12321 7
Approach 7
Scope 7
Qualitative and quantitative measures 7

Potential topics for exploration 9

Promising techniques 10

Achievements and further work 12

Participants 15

3 Dagstuhl seminar 12321

3.1 Approach

At the 2012 Dagstuhl seminar, attendees looked at the problem of robust query processing both more simply and more directly. Rather than attempt to define robustness *per se,* instead the attendees were asked to modify a classical model of query processing so that the result would be more 'robust', whatever that might mean. The basic idea was to model query processing as a simplistic, generic, cascading 'waterfall' system implementation, which included parsing, followed by query rewrite optimization, then join enumeration, and finally query execution. In detail, this set of cascading query processing functions included the following steps:

- logical database design;
- physical database design;
- SQL parsing;
- plan caching: per query, per parameter, and so on, including recompilation control and plan testing;
- query rewrite optimization;
- query optimization proper, including join enumeration;
- query execution;
- database server resource management: memory, threads, cores, and so on;
- database server storage layer, including the server's buffer pool and transactional support.

Workshop participants were then invited to propose approaches that would modify this 'strawman' query processing model by proposing the modification or transposition of query optimization or query execution phases that, it was hoped, would lead to more robust system behaviour.

3.2 Scope

The organizers purposefully chose to keep the focus of discussion relatively narrow and concentrated on more traditional components of relation database query processing. Hence the scope of our discussions included aspects such as Query optimization, query execution, physical database design, resource management, parallelism, data skew, database updates, and database utilities. Outside of the seminar's scope were arguably more esoteric query processing topics such as text and image retrieval, workload management, cloud-only issues, map-reduce issues, extract, transform, load processing, and stream query processing.

3.3 Qualitative and quantitative measures

To permit workshop participants to focus their ideas on the costs and benefits of their proposals, the workshop organizers developed a list of questions that each work group needed to address with their specific proposal. The questions were:

1. *What is the decision that's to be moved within the waterfall model?* Within the waterfall framework workshop participants could make several choices; for example, they could decide to introduce a new query processing layer, rather than (simply) move functionality

from one layer to another. As a concrete example, one might decide to create a new Proactive Physical Database Design layer within the model, thereby treating physical database design as a process and not as a static constraint imposed on server operation.

2. *Where is the decision made in the original waterfall model? What alternative or additional location do you suggest?* To keep the discussions simple, the organizers decided to avoid getting into complex situations such as 'parallel' waterfall implementations . For example, the physical database design phase, typically considered an offline task, could be 'pushed' into a periodic query processing phase, but we would stop at considering an 'online' physical database design task that could greatly perturb the overall model of the system.
3. *How do you know that this is a real problem?* Industry representatives at the Workshop stated that periodic workload fluctuations are the reality in commercial environments and that manual intervention to deal with issues as they arise is unrealistic. Even if performance analysis tools are available, the resources required to diagnose and solve database server performance problems are too great. Moreover, workload creation and/or modeling remains often too time-consuming for many database administrators, exacerbating diagnosis issues.
4. *What is the desired effect on robust performance or robust scalability in database query processing?* Not all proposals may lead to better absolute performance. Rather, the target metric of the proposals is to improve the system's robustness, however that may be defined. For example, a specific proposal may: (a) better anticipate the workload, (b) inform the DBA ahead of the system's peak workload, (c) permit the system to be better prepared for the system's expected workload, (d) support better energy efficiency, (e) improve better worst case expected performance of a given workload, or (f) provide additional insights for a DBA to permit better fault diagnosis.
5. *What are the conditions for moving this decision?* One of the tasks for each group looking at a specific proposal was to determine the parameters for moving a query processing task from one phase to another. In some cases, decision tasks would be designed to move to other phases only under some conditions—for example, due to the periodicity of the workload—whereas in other cases proposals included ideas to permanently move decisions from one query processing phase to another.
6. *How do the two decision points differ in terms of available useful information?* For example, in many cases it is feasible to consider the migration of a query processing task to another phase only when the metadata surrounding the execution context is known and complete. For example, the system may have a real, captured workload, captured performance indicators, and estimated and actual query execution costs. For other, the amount of metadata required may be significantly less.
7. *What are the limitations, i.e., when does the shift in decision time not work or not provide any benefit?* For example, is there a type of workload for which a shift in query processing execution will be counter-productive?
8. *How much effort (software development) would be required?* Is is feasible to consider implementing this proposed technique within a short-term engineering project? Are the risks of implementing the proposal quantifiable? Are the risks real? Can they be mitigated in some way?
9. *How can the potential benefit be proven for the first time, e.g., in a first publication or in a first white paper on a new feature?* Can the benefits of the new technique be described sufficiently well in a database systems conference paper? What workloads can be used to demonstrate the proposal's effectiveness, so that the proposal's benefit can be independently verified by others in the field?

10. *How can the benefit be protected against contradictory software changes, e.g., with a regression test?* This is a complex and difficult problem due in part to subtle changes in underlying assumptions that are made in virtually all complex software systems.
11. *Can regression tests check mechanisms and policies separately?*

To illustrate a potential re-ordering technique, seminar chair Goetz Graefe used an example of moving statistics collection from the physical database design step into the query optimization phase. This would mean that statistics collection would be performed on an on-demand basis during query optimization, rather than as part of a separate, offline 'performance tuning' phase. The positive impact of such a change would be to avoid blind cardinality estimation and access plan choices, at the expense of increasing the query's compile-time cost. Conditions that could impact the effectiveness of the proposal would be missing statistics relevant to the query, or stale statistics that would yield an inaccurate cost estimate. A proof of concept could include examples of poor index and join order choices using an industry standard benchmark, along with mechanisms to control the creation and refresh of these statistics at optimization time.

4 Potential topics for exploration

On the seminar's first day, Monday, seminar attendees brainstormed a variety of potential ideas to shift query processing functionality from one query processing phase to another. These included:

1. Admission control moved from workload management to query execution: 'pause and resume' functionality in queries and utilities.
2. Index creation moved from physical database design to query execution.
3. Materialized view and auxiliary data structures moved from physical database design to query optimization or query execution.
4. Physical database design (vertical partitioning, columnar storage, and column and/or table compression techniques) moved from physical database design to query execution.
5. Join ordering moved from query optimization to query execution.
6. Join and aggregation algorithm: moved from query optimization to query execution, along with set operations, DISTINCT, etc.
7. Memory allocation moved from resource management and query optimization to query execution.
8. Buffer pool policies: move policy control from the storage layer to query optimization and/or query execution.
9. Transaction control: move from the server kernel to the application.
10. Serial execution (1 query at a time): consider moving away from concurrent workload management to fixed, serial execution of all transactions.
11. Query optimization a week ago: move physical database design decisions to the query optimizer by pro-actively, iteratively modifying the database's physical database design through continuous workload analysis.
12. Consider a minimal (in terms of algorithms, implementation effort, and so on) query execution engine that is robust.
13. Query optimization: move from pre-execution to post-execution.

14. Consider the introduction of a robustness indicator during query execution; the idea is to analyze the robustness or efficacy of the query optimization phase—as a simple example, the standard deviation of query execution times.
15. Develop a comparator for pairs of 'similar' queries in order to explain what is different between them, both in terms of SQL semantics and in terms of access plans.
16. Holistic robustness.
17. Parallel plan generation, including degree of parallelism: move from query optimization to query execution.
18. Statistics collection and maintenance: move from physical database design to query optimization.
19. Storage formats: move from physical database design to another phase, possibly query optimization or query execution based on plan cache information.
20. Move refresh statistics, predicate normalization and reordering, multi-query awareness, access path selection, join order, join and aggregation algorithms, cardinality estimation, cost calculation, query transformation, common subexpression compilation, parallel planning, and Halloween protection to within the join sequence.
21. Pre-execution during query optimization; for cardinality estimation and for intermediate results, consider the interleaving of query optimization and query execution.
22. Data structure reorganization: move from query execution to query optimization.
23. Develop a measure of robustness and predictable performance as a cost calculation component for a query optimizer.
24. Statistics refresh: move from physical database design or query optimization to query execution
25. Plan cache management: consider augmenting mechanisms and policies set by the query optimization phase with outcomes from the query execution phase.
26. Plan caching decisions: move from the plan cache manager to within the query execution phase.

Over the remaining days of the 12321 seminar, attendees selected topics from the above list and met to consider these ideas in greater detail, and if possible develop a study approach for each that could take place once the Seminar had concluded.

5 Promising techniques

During the seminar's remaining days, attendees focused on studying the techniques above and developed concrete plans of study for a variety of approaches that each could improve a DBMS system's robustness. Of those discussed at Dagstuhl, attendees reached consensus that the following ideas were worthwhile exploring in greater depth once the seminar had concluded:

1. *Smooth operations.* The proposal is to implement a new query execution operator, called SmoothScan. At the query execution level, via the SmoothScan operator the system continuously refines the choices made initially by the query optimizer, being able to switch dynamically between index look-up techniques to table scans and vice-versa in a smooth manner.
2. *Opportunistic query processing.* Instead of executing only a single query plan (which may or may not be adaptable), the proposal takes an opportunistic approach that carefully

chooses and executes multiple query plans in parallel, where the fastest plan at any given stage of the query execution determines the overall execution time.

3. *Run-time join order selection.* Investigate techniques to not only re-order joins at execution time, but utilize the construction of small intermediate results to interleave optimization and execution and reduce the degree of errors in cardinality estimation.
4. *Robust plans.* The authors propose a new class of operators and plans that are 'smooth' in that their performance degrades gracefully if the expected and the actual characteristics of the underlying data differ. Smooth operators continuously refine the plan choices made by the query optimizer up-front, in order to provide robust performance throughout a wide range of query parameters.
5. *Assessing robustness of query execution plans.* Develop metrics and techniques for comparing two query execution plans with respect to their robustness.
6. *Testing adaptive execution.* Develop an experimental methodology to assess the impact of cardinality estimation errors on system performance and therefore evaluate, by comparison, the impact of adaptive execution methods.
7. *Pro-active physical design.* Using an underlying assumption of periodicity in most workloads, pro-actively, iteratively modify the database's physical design through continuous workload analysis.
8. *Adaptive partitioning.* In this proposal, the authors seek to continuously adapt physical design considerations to the current workload. Extending the ideas of *database cracking* [20], the authors propose additional 'cracking' techniques to both partition physical media (including SSDs) and to handle multi-dimensional queries over multiple attributes using KD-trees and a Z-ordering curve to track related data partitions.
9. *Adaptive resource allocation.* In this proposal the authors make the case for dynamic and adaptive resource allocation: dynamic memory allocation at run-time, and dynamic workload throttling, adaptively control concurrent query execution.
10. *Physical database design without workload knowledge.* In this proposal, the authors look at physical database design decisions during bulk loading. The potential set of design choices are page size/format, sorting, indexing, partitioning (horizontal, vertical), compression, distribution/replication within a distributed environment, and statistics.
11. *Weather prediction.* In an analogy to weather prediction, the authors propose a technique to manage user expectations of system performance through analysis of the current workload and prediction parameters analyzed using the current system state.
12. *Lazy parallelization.* Static optimization involving parallelization carries considerable risk, due to several root causes: (1) the exact amount of work to be done is not known at compile time, (2) data skew can have a significant impact on the benefits of parallelism, and (3) the amount of resources available at run time may not be known at compile time. Instead, the proponents of this approach intend to move parallelism choices from the query optimization phase to the query execution phase. In this scenario, the query optimizer would generate 'generic' access plans that could be modified on-the-fly during query execution to increase or decrease the degree of parallelism and alter the server's resource assignment accordingly.
13. *Pause and resume.* This proposal focused on mechanisms that permit stopping and resuming later with the least amount of work repeated or wasted—or even reverted, using some form of undo recovery in order to reach a point from which the server can resume the operation. While similar in intent to mechanisms that permit, for example, dynamic memory allocation, the policies and mechanisms with this proposal are quite different,

and rely on task scheduling and concurrency control mechanisms in order to permit the resumption of a paused SQL request.

14. *Physical database design in query optimization.* The idea behind this proposal is to move some physical database design decisions to the query optimization phase. For example, one idea is to defer index creation to query optimization time, so that indexes are created only when the optimizer can determine that they are beneficial. While this has the benefit of avoiding static physical database design and workload analysis, there are no guarantees that the system can detect cost and benefit bounds for all decisions and all inputs.

6 Achievements and further work

The 2012 Dagstuhl 12321 Workshop made considerable progress towards the goals of developing more robust query processing behaviour. That progress was made, primarily, by maintaining strict focus on the task of shifting a query processing decision from one phase to another, and then answering the questions listed in Section 3.3. The use of that framework enabled more concentrated discussion on the relative merits of the specific techniques, and at the same time reduced debate about the definition of 'robust' behaviour, or how to quantify it.

In 2010, Seminar 10381 dwelled on measurement and benchmarks, and subsequent to the seminar two benchmark papers were published. One, which combined elements of both the TPC-C and TPC-H industry-standard benchmarks and entitled the CH-Benchmark, was published in the 2011 DBTEST Workshop, held in Athens the following summer [12]. The other used the metaphor of an agricultural 'tractor pull contest' to create a benchmark to examine several measures related to robustness. Like the CH-Benchmark above, the complete tractor pulling benchmark is described in the Proceedings of the 2011 DBTEST Workshop [22].

In 2012, an achievement of the 2012 Dagstuhl seminar was in enumerating various approaches that could either (1) improve the robustness of a database management system under some criteria, or (2) developing metrics that could be used to measure aspects of a system's behaviour that could be used to optimize a system's set of decision points. The organizers are confident that this concrete progress will lead to several publications in the very near future.

In particular, two drafts of potential proposals have already been developed: the first for 'smooth' scans and 'smooth' operators, and the second regarding proactive physical database design for periodic workloads. The seminar's organizers are confident that several other proposals discussed at the 12321 Workshop will develop into research projects in the coming months.

However, despite the significant progress made at the 12321 Seminar in developing robust query processing techniques, it became clear during both the 10381 and 12321 seminars that there were no known ways of systematically, but efficiently, testing the robustness properties of a database management system in a holistic way. While there is a fairly obvious connection between robustness—however one might try to define it—and self-managing database management system techniques, testing these systems to ensure that they provide robust behaviour is typically limited in practice to individual unit tests of specific self-managing components. Testing the interaction of these various technologies together, with a production-like workload, remains an unsolved and difficult problem. Indeed, testing and metrics development remain an unknown factor for many, if not all, of the ideas generated at this latest Workshop.

References

1 Ashraf Aboulnaga and Surajit Chaudhuri. Self-tuning histograms: Building histograms without looking at data. In ACM SIGMOD *International Conference on Management of Data*, pages 181–192, Philadelphia, Pennsylvania, May 1999.

2 Mohammed Abouzour, Kenneth Salem, and Peter Bumbulis. Automatic tuning of the multiprogramming level in Sybase SQL Anywhere. In *ICDE Workshops*, pages 99–104. IEEE, 2010.

3 Mumtaz Ahmad, Ashraf Aboulnaga, Shivnath Babu, and Kamesh Munagala. QShuffler: Getting the query mix right. In *Proceedings of the* IEEE *International Conference on Data Engineering*, pages 1415–1417, 2008.

4 Mumtaz Ahmad, Ashraf Aboulnaga, Shivnath Babu, and Kamesh Munagala. Interaction-aware scheduling of report-generation workloads. *The* VLDB *Journal*, 20:589–615, August 2011.

5 Mumtaz Ahmad, Songyun Duan, Ashraf Aboulnaga, and Shivnath Babu. Predicting completion times of batch query workloads using interaction-aware models and simulation. In *Proceedings of the 14th International Conference on Extending Database Technology*, pages 449–460, New York, NY, USA, 2011. ACM.

6 Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously adaptive query processing. In ACM SIGMOD *International Conference on Management of Data*, pages 261–272, 2000.

7 Pedro Bizarro, Nicolas Bruno, and David J. DeWitt. Progressive parametric query optimization. IEEE *Transactions on Knowledge and Data Engineering*, 21:582–594, 2009.

8 Pedro G. Bizarro. *Adaptive query processing: dealing with incomplete and uncertain statistics*. PhD thesis, University of Wisconsin at Madison, Madison, Wisconsin, 2006.

9 Surajit Chaudhuri, Hongrae Lee, and Vivek R. Narasayya. Variance aware optimization of parameterized queries. In ACM SIGMOD *International Conference on Management of Data*, pages 531–542, 2010.

10 Surajit Chaudhuri, Vivek R. Narasayya, and Ravishankar Ramamurthy. A pay-as-you-go framework for query execution feedback. *Proceedings of the* VLDB *Endowment*, 1(1):1141–1152, 2008.

11 Mitch Cherniack. *Building Query Optimizers with Combinators*. PhD thesis, Brown University, Providence, Rhode Island, May 1999.

12 Richard Cole, Florian Funke, Leo Giakoumakis, Wey Guy, Alfons Kemper, Stefan Krompass, Harumi Kuno, Raghunath Nambiar, Thomas Neumann, Meikel Poess, Kai-Uwe Sattler, Michael Seibold, Eric Simon, and Florian Waas. The mixed workload CH-benCHmark. In *Proceedings of the Fourth International Workshop on Testing Database Systems*, New York, NY, USA, 2011. ACM.

13 Amol Deshpande, Zachary G. Ives, and Vijayshankar Raman. Adaptive query processing. *Foundations and Trends in Databases*, 1(1):1–140, 2007.

14 Kwanchai Eurviriyanukul, Norman W. Paton, Alvaro A. A. Fernandes, and Steven J. Lynden. Adaptive join processing in pipelined plans. In *13th International Conference on Extending Database Technology (*EDBT*)*, pages 183–194, 2010.

15 Parke Godfrey, Jarek Gryz, and Calisto Zuzarte. Exploiting constraint-like data characterizations in query optimization. In ACM SIGMOD *International Conference on Management of Data*, pages 582–592, Santa Barbara, California, May 2001. Association for Computing Machinery.

16 Goetz Graefe, Arnd Christian König, Harumi Kuno, Volker Markl, and Kai-Uwe Sattler. Robust query processing. Dagstuhl Workshop Summary 10381, Leibniz-Zentrum für Informatik, Wadern, Germany, September 2010.

17 Michael Greenwald. Practical algorithms for self-scaling histograms or better than average data collection. *Performance Evaluation*, 20(2):19–40, June 1996.

18 Jarek Gryz, Berni Schiefer, Jian Zheng, and Calisto Zuzarte. Discovery and application of check constraints in DB2. In *Proceedings, Seventeenth* IEEE *International Conference on Data Engineering*, pages 551–556, Heidelberg, Germany, April 2001. IEEE Computer Society Press.

19 Waqar Hasan and Hamid Pirahesh. Query rewrite optimization in STARBURST. Research Report RJ6367, IBM Corporation, Research Division, San Jose, California, August 1988.

20 Stratos Idreos, Martin L. Kersten, and Stefan Manegold. Database cracking. In *CIDR*, pages 68–78. www.cidrdb.org, 2007.

21 Ihab F. Ilyas, Volker Markl, Peter J. Haas, Paul Brown, and Ashraf Aboulnaga. CORDS: Automatic discovery of correlations and soft functional dependencies. In ACM SIGMOD *International Conference on Management of Data*, pages 647–658, Paris, France, June 2004.

22 Martin L. Kersten, Alfons Kemper, Volker Markl, Anisoara Nica, Meikel Poess, and Kai-Uwe Sattler. Tractor pulling on data warehouses. In *Proceedings of the Fourth International Workshop on Testing Database Systems*, pages 7:1–7:6, New York, NY, USA, 2011. ACM.

23 Jonathan J. King. QUIST–A system for semantic query optimization in relational databases. In *Proceedings of the 7th International Conference on Very Large Data Bases*, pages 510–517, Cannes, France, September 1981. IEEE Computer Society Press.

24 Volker Markl, Vijayshankar Raman, David E. Simmen, Guy M. Lohman, and Hamid Pirahesh. Robust query processing through progressive optimization. In ACM SIGMOD *International Conference on Management of Data*, pages 659–670, 2004.

25 Rimma V. Nehme, Elke A. Rundensteiner, and Elisa Bertino. Self-tuning query mesh for adaptive multi-route query processing. In *12th International Conference on Extending Database Technology (*EDBT*)*, pages 803–814, 2009.

26 G. N. Paulley and Per-Åke Larson. Exploiting uniqueness in query optimization. In *Proceedings, Tenth* IEEE *International Conference on Data Engineering*, pages 68–79, Houston, Texas, February 1994. IEEE Computer Society Press.

27 Hamid Pirahesh, Joseph M. Hellerstein, and Waqar Hasan. Extensible/rule based query rewrite optimization in STARBURST. In ACM SIGMOD *International Conference on Management of Data*, pages 39–48, San Diego, California, June 1992. Association for Computing Machinery.

28 H. J. A. van Kuijk. The application of constraints in query optimization. Memoranda Informatica 88–55, Universiteit Twente, Enschede, The Netherlands, 1988.

Participants

- Martina-Cezara Albutiu
TU München, DE
- Peter A. Boncz
CWI – Amsterdam, NL
- Renata Borovica
EPFL – Lausanne, CH
- Surajit Chaudhuri
Microsoft – Redmond, US
- Campbell Fraser
Microsoft – Redmond, US
- Johann Christoph Freytag
HU Berlin, DE
- Goetz Graefe
HP Labs – Madison, US
- Ralf Hartmut Güting
FernUniversität in Hagen, DE
- Wey Guy
Independent, US
- Theo Härder
TU Kaiserslautern, DE
- Fabian Hüske
TU Berlin, DE
- Stratos Idreos
CWI – Amsterdam, NL
- Ihab Francis Ilyas
University of Waterloo, CA
- Alekh Jindal
Universität des Saarlandes, DE
- Martin L. Kersten
CWI – Amsterdam, NL
- Harumi Anne Kuno
HP Labs – Palo Alto, US
- Andrew Lamb
Vertica Systems – Cambridge, US
- Allison Lee
Oracle Corporation – Redwood Shores, US
- Stefan Manegold
CWI – Amsterdam, NL
- Anisoara Nica
Sybase – Waterloo, CA
- Glenn Paulley
Conestoga College – Kitchener, CA
- Ilia Petrov
TU Darmstadt, DE
- Meikel Poess
Oracle Corp. – Redwood Shores, US
- Ken Salem
University of Waterloo, CA
- Bernhard Seeger
Universität Marburg, DE
- Krzysztof Stencel
University of Warsaw, PL
- Knut Stolze
IBM Deutschland – Böblingen, DE
- Florian M. Waas
EMC Greenplum Inc. – San Mateo, US
- Jianliang Xu
Hong Kong Baptist Univ., CN
- Marcin Zukowski
ACTIAN – Amsterdam, NL