Report from Dagstuhl Seminar 17222

Robust Performance in Database Query Processing

Edited by
Renata Borovica-Gajic[1], Goetz Graefe[2], and Allison Lee[3]

1 **The University of Melbourne, AU,** `renata.borovica@unimelb.edu.au`
2 **Google – Madison, US,** `goetzg@google.com`
3 **Snowflake Computing Inc. – San Mateo, US,** `allison.lee@snowflake.net`

Abstract

The Dagstuhl Seminar 17222 on "Robust performance in database query processing", held from 28/May until 02/June 2017, brought together researchers from academia and industry to discuss aspects of robustness in database management systems that have not been addressed by the previous instances of the seminar. This article summarizes the main discussion topics, and presents the summary of the outputs of four work groups that discussed: i) updates and database utilities, ii) parallelism, partitioning and skew, iii) dynamic join sequences, and iv) machine learning techniques used to explain unexpected performance observations.

Seminar May 28–2, 2017 – http://www.dagstuhl.de/17222
1998 ACM Subject Classification H.2 DATABASE MANAGEMENT; H.2.2 Physical Design: Access Methods, H.2.4 Systems: Parallel databases, Query processing, Relational databases
Keywords and phrases Robust Query Performance, Database Management Systems, Adaptive Query Processing, Query Optimization, Query Execution, Updates, Parallelism, Data Skew
Digital Object Identifier 10.4230/DagRep.7.5.169

1 Executive Summary

Renata Borovica-Gajic
Goetz Graefe
Allison Lee
Glenn Paulley

The Dagstuhl Seminar 17222 on "Robust performance in database query processing" assembled researchers from industry and academia for the third time to discuss robustness issues in database query performance. The seminar gathered 24 researchers around the world working on plan generation and plan execution in database query processing and in cloud-based massively parallel systems with the purpose to address the open research challenges with respect to the robustness of database management systems.

Delivering robust query performance is well known to be a difficult problem for database management systems. All experienced DBAs and database users are familiar with sudden disruptions in data centers due to poor performance of queries that have performed perfectly well in the past. The goal of the seminar is to discuss the current state-of-the-art, to identify specific research opportunities in order to improve the state-of-affairs in query processing, and to develop new approaches or even solutions for these opportunities.

Unlike the previous seminars, the organizers (Renata Borovica-Gajic, Goetz Graefe and Allison Lee) this time attempted to have a focused subset of topics that the participants discussed and analyzed in more depth. From the proposed topics on algorithm choices, join sequences, updates, database utilities, parallelism and skew, column stores, physical database design, and explainability of non-robust query performance, the participants chose four topics and formed four work groups: i) one discussing updates and database utilities, ii) one discussing parallelism and skew, iii) one discussing join sequences, and iv) one focusing on the explanations of the sources of non-robust performance.

Upon choosing the topics of interest, the organizers then guided the participants to approach the topic through a set of steps: by first considering related work in the area; then introducing metrics and tests that will be used for testing the validity and robustness of the solution; after metrics, the focus was on proposing specific mechanisms for the proposed approaches; and finally the last step focused on the implementation policies.

The seminar thus spent its first day on reviewing prior related work, with a special emphasis on the pieces of work that appeared following the previous instances of the seminar: benchmarks (Dagstuhl 12321 [4, 6, 7]), Smooth Scan [2], and Generalized join [3]. Tuesday was spent on defining metrics and tests. On Wednesday, the participants discussed possible alternative approaches and hiked together in the woods. Thursday was focused on driving one chosen approach to specific mechanisms. Finally, we spent Friday on discussing the policies and presented the overall progress.

At the end of the week, each group was hoping to continue their work towards a research publication. The group on parallelism and skew was hoping to publish first a survey on forms of skew and existing remedies for skew. The work group on dynamic join sequences even had a working prototype by the end of the seminar. The reports of work groups are presented next.

2 Table of Contents

3 Working groups

3.1 Updates and database utilities

Ilia Petrov, Jiaqi Yan, Thanh Do, Knut Stolze, John Cieslewicz, and Goetz Graefe

Within Dagstuhl Seminar 17222, the work group on updates and database utilities explored mechanisms and policies for database updates including very large updates as common in load utilities. The principal mechanisms for improving performance and scalability focus on avoiding write amplification, e.g., reading and writing an entire database page for an update-in-place modifying just a few bytes, doing the same in multiple indexes and materialized views, logging updates of a few bytes in log records with large headers, etc. Instead of updates-in-place, many systems use variations of very traditional master tapes and change tapes, often called deltas and sometimes forced by append-only file systems. For ordered indexes, the resulting storage structures are log-structured merge trees (really forests with many trees!) and partitioned b-trees (a single b-tree with many internal partitions).

These storage structures may improve not only efficiency and scalability but also robustness of performance, which we understand to mean linear or near-linear (eg N log N) execution costs when expressed as function of size of database or table, size of input or change set, and size of transactions (bulk load vs trickle load). The opposite of robust performance are cost functions with cliffs. Our focus here is on measured performance rather than theoretical performance of algorithms or data structures; we expect that the measured performance correlate very highly with a linear or near-linear regression line.

We hope to capture and extend our results in a publication, e.g., to a workshop on performance benchmarking or on database query processing. Rather than focus on any one proprietary system or on a specific set of techniques, this publication will be about benchmarking and metrics for updates and for database utilities; and rather than focus on single-thread efficiency or multi-node scalability, it will be about measuring robustness of performance.

3.2 Parallel Join Processing with Skew

Peter A. Boncz, Alfons Kemper, Angelos Anadiotis, J. Christoph Freytag, Kai-Uwe Sattler, and G. N. Paulley

The Dagstuhl Workshop 17222 workgroup on Parallelism and Skew explored a range of topics surrounding the ability for a database system to mitigate the problems of data skew in processing join queries in both tightly- and loosely-coupled database systems. While the problems surrounding data skew have been studied over the past thirty years, and some work has documented well some of the fundamental aspects of data skew in join processing, the workgroup felt that an in-depth look at the state of the art was necessary, for a number of reasons. First, much of the work on joins with skewed data was published between 25 and 30 years ago when system performance tradeoffs – for example, the relative speed of disk to main memory – was considerably different than it is today. Second, our workgroup focus was not merely on raw performance but on characterizing robust behaviour, a viewpoint largely missing from prior work. Third, it quickly became desirable to define a robustness benchmark for joins over skewed data so that we could measure performance degradation and compare systems using a common, agreed-upon metric. Fourth, we were aware of some recent research in the literature, notably Flow-Join from this year's ICDE conference in Helsinki and co- authored by two of the members of seminar 17222, that offered advantages over existing, more well-known approaches [10].

During the seminar the workgroup explored three closely-related ideas: a robustness benchmark for join processing in the face of data skew; a survey of existing techniques from the literature; and improvements to current techniques to further mitigate the presence of skew by either avoiding known 'performance cliffs' at query optimization time, or adapting to circumstances at query execution time to avoid such cliffs on one or more threads or processes that will causes overall query latency to increase. During the seminar, the group concentrated on the first two topics. Our primary goal during the seminar was to define a robustness benchmark for join processing in the face of skew, as our feeling was that customers and users have expectations of linear or near-linear (e.g. N lg N) execution costs when expressed as a function of the size of a database or table, and as well expect near-linear performance with respect to increases in memory size and the relative number of CPU cores. The group considered several possible functions, but eventually decided upon a metric R that exponentially degrades as query execution times become less predictable. We then considered options about to construct such a benchmark, and looked at several promising and existing options: the Social Network benchmark, the Star Schema benchmark, TPC-DS, and a modified version of TPC-H. We decided upon the latter option, and discussed the different types of skew that could be placed into the benchmark. We also discussed modifying the benchmark's existing queries, or changing them outright, to determine how best to illustrate the effects of skew.

In the case of skewed column distributions, it makes sense to have different parameter equivalence classes; for example, in the TPC-DS benchmark, these are values from the same step in the step functions that it uses. These are parameters from a skewed distribution, that despite being skewed, still has multiple individual values with the same frequency. The goal is to be able to run a particular query variant with different parameters but mostly identical behavior; that is, values with the same frequency imply similar intermediate result cardinalities.

However, in a skewed distribution, there may be multiple such equivalence classes, e.g. some highly frequent values (all the same frequency) and some low frequency values (all the same frequency). This leads to the concepts of query variants, e.g. Q2 of the benchmark may now have two variants, Q2a and Q2b, where variant 2a binds with the frequent parameter values, and 2b with the infrequent ones. Subsequent to the Workshop our focus group plans to modify the DBGEN utility of the TPC-H benchmark to determine the feasibility of our ideas and their usefulness in deriving a robustness metric for a DBMS [11].

Our second thrust during our meetings was to examine the state-of-the-art in parallel join processing to determine how query processing performance could be improved. We identified a collection of seminal papers from the existing literature as a starting point for our survey. We plan to augment these papers to develop a much more comprehensive survey that we hope will be of value to the research community.

3.3 Explaining unexpected performance in database query execution

Surajit Chaudhuri, Anisoara Nica, Marcin Zukowski, Fabian Hueske, Immanuel Trummer, Tahir Azim, and Renata Borovica-Gajic

Within Dagstuhl Seminar 17222, the work group on "Explaining unexpected performance in database query execution" explored mechanisms for informing users about the source of performance deviation (typically degradation) of a query with respect to prior query executions. The goal of this effort is to increase the user satisfaction and develop trust toward the database management systems by providing explanation for poor query performance. The existence of such a tool would also be highly beneficial for database vendors as a way to mitigate costs of database support, since the tool would reduce the number of angry calls from users requesting justification for query performance change.

In general, the work group divided the source of deviation into two broad categories: i) expected deviation because of larger query inputs or reduced resources (e.g. lower memory budget), ii) unexpected deviation as a typical consequence of a plan change or a performance cliff in the cost model that triggered higher resource usage (e.g. intermediate results spilling to disk). The first category encompasses "robust", i.e. expected and justified query performance given the query's inputs and outputs, while the second group displays "non-robust" behaviour where the query input is not the reason for observed performance degradation.

Within this particular problem, the group especially focused on a somewhat easier problem of detecting and explaining deviation of a single query template over a sequence of observed query invocations. Templatized queries are typical in modern decision support systems characterized by a frequent repeat of the same plan template with different parameter values (e.g. reports or user-defined functions). Given this context, our problem can be formulated as: Given the past X executions of query Q, detect whether and explain why the invocation X+1 deviates from the expected performance of query Q given the input X+1.

A crucial question to answer is what we consider as "expected" given the new input characteristics and prior query executions. An approach that the group discussed during the seminar is the approach of employing machine learning techniques to train a model of the behavior of the query given its past executions. The group discussed in detail what should constitute a model, which features should be extracted, etc. Alternatively, the group discussed whether techniques such as learning cost models or statistical models could be sufficient to explain the source of performance deviation.

At a high-level, the group settled on an operator-level approach for modeling query performance. For each operator in each query template, the system collects statistics on a variety of metrics, including input and output cardinalities and physical resource usage (CPU, memory, IO). A machine learning model trained for each metric then characterizes its behavior over the query history seen thus far. If a subsequent query is unexpectedly slow, consulting these models provides clues on which metrics were anomalous and could provide a possible explanation for the slowdown. Furthermore, the operator-level approach prevents anomalies from getting compounded or merged up the query tree. Thus, any unexpectedly slow operator can be detected as close to the leaves as possible.

As part of the discussion into this approach, the group reviewed a number of recent papers related to the use of machine learning for modeling and predicting query performance. The goal of this review was two-fold: first, to familiarize the group with the most relevant work in the domain and, second, to check if existing work had previously tackled the problem of explainable query performance. In summary, we found a substantial amount of work on using machine learning for query performance modeling and prediction, but almost no work on explaining the causes when query performance deviates from the trained models.

Malik et al [8] parse out features from the query text of SkyServer queries and combine them with previously seen output cardinalities to train a model for future cardinality estimation. Tzoumas et al [13, 12] use graph theory and machine learning to detect correlations in the input data and thus improve cardinality estimation. Akdere et al [1] demonstrate the effectiveness of machine learning to predict query execution time by using plan-level and operator-level models based on optimizer cost estimates, query parameters and actual runtime statistics. Finally, Wu et al [5] avoid machine learning approaches and instead use additional statistics to predict query execution time by building on PostgreSQL's optimizer cost models. They use offline profiling to estimate the unit cost of reading pages and tuples using different access paths. They then use online sampling for each query to estimate how many pages and tuples will be read by a query. In a series of follow-up papers [14, 15], they incorporate more sophisticated statistical models to improve query performance prediction in the presence of uncertainty and concurrency. Microsoft's Adaptive Query Processing feature [9] adds the ability to test a query's performance with different plans and visually inspect operators for estimated and actual statistics along with possible performance warnings. However, it does not correlate the current query performance with what the user has seen in the past. While each of these techniques improves the state-of-the-art in query performance estimation, the task of explaining why many queries deviate from the estimates remains out of their scope.

The group also gave considerable attention towards proposing benchmarks that will model realistic use cases, while at the same time would catch all sources of "unexpected" performance behavior. As a first step to this goal the work group proposed a benchmark based on significant data correlation and workload skew. As a future step, we plan to consider the impact of resources as well and incorporate them into the test benchmark.

The overall work group plan is to pursue a publication on this topic followed by a supporting software tool whose main focus would be on explaining unexpected performance to the user either graphically or in the form of natural language text.

3.4 Dynamic join sequences

Campbell Fraser, Bart Samwel, Thomas Neumann, Srinivas Karthik, and Allison Lee

The workgroup on operator sequencing in Dagstuhl Seminar 17222 explored techniques to avoid catastrophically bad performance due to poor join order selection by the query optimizer. In a conventional SQL Query Processing engine, the Query Optimizer generates a single execution plan for each query. For a host of reasons, this often results in query performance that is orders of magnitude worse than that of the fastest possible performance for the query. The models used to estimate the cardinalities at intermediate stages of query execution are often too simple to provide accurate estimates, and the statistics on which they rely can be out of date. Parameterised queries are particularly vulnerable because, even with perfect statistics and a rich estimation model, it may be true that no single plan is capable of providing acceptable performance for all possible parameters, for a given query.

The workgroup reviewed existing adaptive execution techniques that have been proposed to overcome this architectural limitation, including re-optimizing fragments of the plan during execution, tuple routing, dynamically rearranging the order of joins, and parameter sensitive plans. Based on previous work and initial ideas for solutions, we generated a list of limitations to the scope of solutions, that we could use to evaluate the usefulness of a solution for a given use case. For instance, previous work on tuple routing was limited to certain join methods (e.g. symmetric hash join), and work on dynamically rearranging the order of joins was limited to linear plans of indexed nested-loops joins.

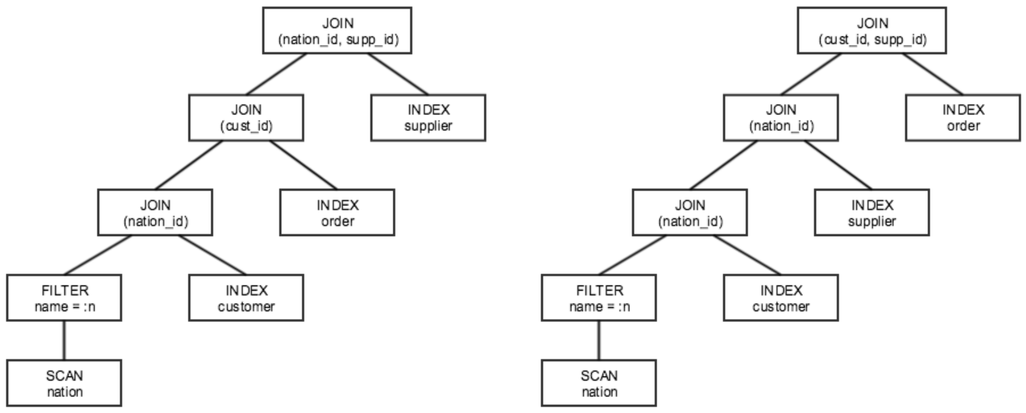
The group proposed a simple test query and data set that could be used to evaluate the performance of proposed solutions.

```
SELECT *
FROM nations n
     join customers c using (n_id)
     join orders o using (c_id)
     join suppliers s on s.n_id = c.n_id and s.s_id = o_sid
WHERE n.name = :n
```

The query contains a join which can substantially blow up the size of intermediate results, depending on the value of a query parameter (: n). For most (less frequent) parameter values, any join order performs adequately, while for more frequent values, the wrong join order will result in performance that is orders of magnitude worse than other join orders. Throughout our discussions, this query was used to evaluate the potential of various ideas that were proposed. We also proposed a more general technique to evaluate a workload against a proposed solution, which involved generating a large set of random join orders for each query, detecting the "cliff" in the distribution of runtimes of those join orders, and then evaluating the impact of a solution on join orders falling on either side of the cliff.

After brainstorming several possible solutions, we focused on a novel tuple-routing approach. The bookkeeping for the routing is simple, does not increase in size or complexity with the number of joins in the query, and requires only inexpensive telemetry that is already gathered by some query processing engines. An intuitive system of back pressure and coordination is used to route tuples away from inefficient join orders and through efficient orders. The method is primarily aimed at linear indexed nested-loop joins but can also work with bushy plans and with hash tables functioning as "indexes on the fly". Unlike previous adaptive techniques, this method is not index restrictive; a different index can be used for the same table when it appears in different join orders. The algorithm is always making forward progress; it never backtracks or throws away partial results. Figures 1a and 1b show two plan options for our simple test query above. For a frequently occurring value of nation (e.g. 'US'), the plan in Figure 1a performs well, while the plan in Figure 1b performs very badly. Figure 2 shows a tuple-routing plan for this query, with the "good" and "bad" paths annotated. Back pressure will cause most tuples to be routed through the left-most path.

During the seminar, we implemented a prototype inside of a teaching database, which handled linear indexed nested-loops joins only. We compared performance (measured as the total number of tuples produced as intermediate results) of all possible join orders for our test query against our tuple routing scheme. As expected, we found a worst case 2X degradation in intermediate tuples produced versus a good plan, and orders of magnitude improvement in the worst plans.



(a) Good plan **(b)** Bad plan

Figure 1 An example of a good and bad plan

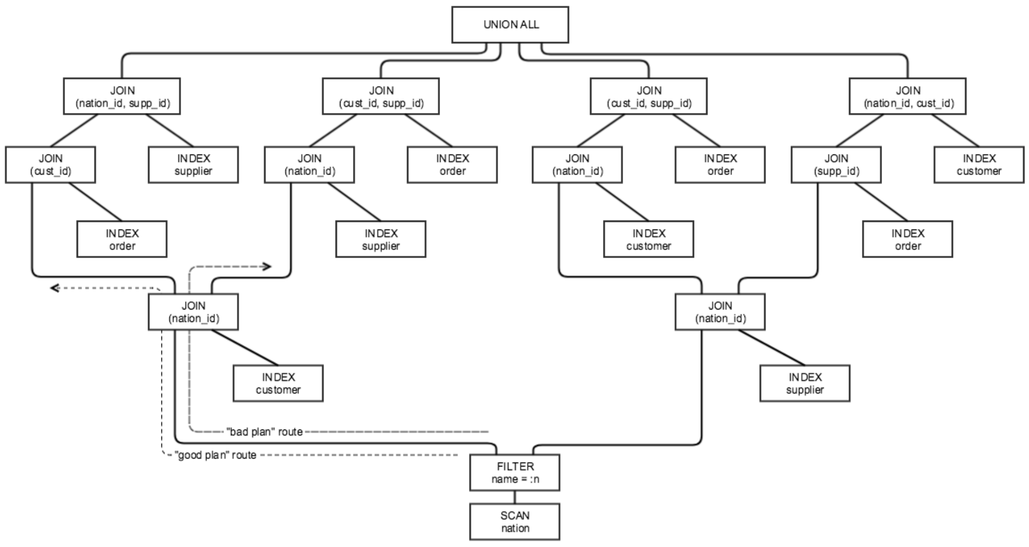


Figure 2 A tuple routing plan

There are several open issues which the group plans to continue work on after the seminar, including:

- policies for when to start to route tuples, to decrease the performance degradation when the optimizer chooses a good plan, and when to stop or decrease tuple routing once a good plan has been found
- implementing back-pressure for non-linear plans,
- allowing hash joins in the plan,
- techniques for swapping the driving table,
- techniques for choosing a subset of join orders in the plan,
- prototyping the solution in a real database system.

4 Summary

State-of-the-art approaches in query processing are oriented toward achieving high performance while robustness of these approaches is often neglected. This may result in queries whose run time fluctuates severely as a result of marginal changes in the underlying data. When talking about robust performance, stable and expected or close to expected performance has to be provided every single time, even in the presence of changes in the selectivity of operators or changes in the system state between compile time and run time. This report presents four approaches toward improving the robustness of database management systems by looking at the impact of updates, parallelism and skew, join orders and explainability of non-robust query performance.

References

1. Mert Akdere, Ugur Çetintemel, Matteo Riondato, Eli Upfal, and Stanley B. Zdonik. Learning-based query performance modeling and prediction. pages 390–401, 2012.
2. Renata Borovica-Gajic, Stratos Idreos, Anastasia Ailamaki, Marcin Zukowski, and Campbell Fraser. Smooth Scan: Statistics-oblivious access paths. In *ICDE*, 2015.
3. Goetz Graefe. New algorithms for join and grouping operations. *Computer Science - R&D*, 27(1):3–27, 2012.
4. Goetz Graefe, Wey Guy, Harumi A. Kuno, and Glenn N. Paulley. Robust query processing (dagstuhl seminar 12321). *Dagstuhl Reports*, 2(8):1–15, 2012.
5. Hakan Hacigumus, Yun Chi, Wentao Wu, Shenghuo Zhu, Junichi Tatemura, and Jeffrey F. Naughton. Predicting query execution time: Are optimizer cost models really unusable? pages 1081–1092, 2013.
6. 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*, DBTest '11, pages 7:1–7:6, New York, NY, USA, 2011. ACM.
7. 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*, DBTest '11, pages 7:1–7:6, New York, NY, USA, 2011. ACM.
8. Tanu Malik, Randal C. Burns, and Nitesh V. Chawla. A black-box approach to query cardinality estimation. pages 56–67, 2007.
9. Microsoft. Microsoft adaptive query processing and diagnostics. `https://www.youtube.com/watch?v=szTmo6rTUjM`.

10 Wolf Rödiger, Sam Idicula, Alfons Kemper, and Thomas Neumann. Flow-join: Adaptive skew handling for distributed joins over high-speed networks. In *ICDE*, pages 1194–1205, 2016.

11 TPC. Tpc-h benchmark. `http://www.tpc.org/tpch/`.

12 Kostas Tzoumas, Amol Deshpande, and Christian S. Jensen. Lightweight graphical models for selectivity estimation without independence assumptions. *PVLDB*, page 2011, 2011.

13 Kostas Tzoumas, Amol Deshpande, and Christian S. Jensen. Efficiently adapting graphical models for selectivity estimation. *The VLDB Journal*, 22(1):3–27, 2013.

14 Wentao Wu, Yun Chi, Hakan Hacígümüş, and Jeffrey F. Naughton. Towards predicting query execution time for concurrent and dynamic database workloads. *PVLDB*, 6(10):925–936, 2013.

15 Wentao Wu, Xi Wu, Hakan Hacigümüş, and Jeffrey F. Naughton. Uncertainty aware query execution time prediction. *PVLDB*, 7(14):1857–1868, 2014.

Participants

- Angelos-Christos Anadiotis
EPFL – Lausanne, CH
- Tahir Azim
EPFL – Lausanne, CH
- Peter A. Boncz
CWI – Amsterdam, NL
- Renata Borovica-Gajic
The University of Melbourne, AU
- Surajit Chaudhuri
Microsoft Research – Redmond, US
- John Cieslewicz
Google Mountain View, US
- Thanh Do
Google – Madison, US
- Campbell Fraser
Google – Kirkland, US
- Johann-Christoph Freytag
HU Berlin, DE
- Goetz Graefe
Google – Madison, US
- Fabian Hüske
data Artisans – Berlin, DE
- Alfons Kemper
TU München, DE
- Allison Lee
Snowflake Computing Inc. – San Mateo, US
- Thomas Neumann
TU München, DE
- Anisoara Nica
SAP SE – Waterloo, CA
- Glenn Paulley
SAP SE – Waterloo, CA
- Ilia Petrov
Hochschule Reutlingen, DE
- Bart Samwel
Google – Amsterdam, NL
- Kai-Uwe Sattler
TU Ilmenau, DE
- Knut Stolze
IBM Deutschland – Böblingen, DE
- Immanuel Trummer
Cornell University – Ithaca, US
- Srinivas Karthik Venkatesh
Indian Institute of Science – Bangalore, IN
- Jiaqi Yan
Snowflake Computing Inc. – San Mateo, US
- Marcin Zukowski
Snowflake Computing Inc. – San Mateo, US