

Ensuring the Reliability and Robustness of Database Management Systems

Edited by

Alexander Böhm¹, Maria Christakis², Eric Lo³, and Manuel Rigger⁴

- 1 SAP SE – Walldorf, DE, alexander@boehm.global
- 2 MPI-SWS – Kaiserslautern, DE, maria@mpi-sws.org
- 3 The Chinese University of Hong Kong, HK, ericlo@cse.cuhk.edu.hk
- 4 ETH Zürich, CH, manuel.rigger@inf.ethz.ch

Abstract

The goal of this seminar was to bring together researchers and practitioners from various domains such as of databases, automatic testing, and formal methods to build a common ground and to explore possibilities for systematically improving the state of the art in database management system engineering. The outcome of the seminar was a joint understanding of the specific intricacies of building stateful system software, as well as the identification of several areas of future work. In particular, we believe that database system engineering can both be significantly improved by adopting additional verification techniques and testing tools, and can provide important feedback and additional challenges (e.g. related to state management) to neighboring domains.

Seminar November 1–4, 2021 – <http://www.dagstuhl.de/21442>

2012 ACM Subject Classification Information systems → Data management systems

Keywords and phrases Databases, Reliability, Robustness, Testing, Dagstuhl Seminar

Digital Object Identifier 10.4230/DagRep.11.10.20

Edited in cooperation with Kulahcioglu Ozkan, Burcu

1 Executive Summary

Maria Christakis (MPI-SWS – Kaiserslautern, DE)

Alexander Böhm (SAP SE – Walldorf, DE)

Eric Lo (The Chinese University of Hong Kong, HK)

Manuel Rigger (ETH Zürich, CH)

License  Creative Commons BY 4.0 International license
© Maria Christakis, Alexander Böhm, Eric Lo, and Manuel Rigger

DataBase Management Systems (DBMSs) are used ubiquitously. Due to the ever-growing number and size of data sets, increasing performance demands, and the virtually unlimited hardware resources that are provided by public cloud infrastructure, sophisticated systems and optimizations are developed continuously. This dynamic and demanding environment is a major challenge for developers of DBMSs, which have to ensure that their systems are both correct and efficient.

Database management systems are a well-established field with several decades of research and engineering attention. These efforts have resulted in a multitude of both open-source and commercial systems that are widely deployed in production today and provide the backbone of a vast range of mission-critical applications. Still, surprisingly, recent work on automatic testing of DBMSs found a large number of bugs in widely-used DBMSs. This



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 4.0 International license

Ensuring the Reliability and Robustness of Database Management Systems, *Dagstuhl Reports*, Vol. 11, Issue 10, pp. 20–35

Editors: Alexander Böhm, Maria Christakis, Eric Lo, and Manuel Rigger



DAGSTUHL
REPORTS Dagstuhl Reports

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

clearly indicated that the topic of ensuring the reliability and robustness of DBMS deserves more attention, and that key insights from neighboring domains such as automatic testing and formal methods could potentially help to advance the state of the art in DBMS engineering.

Goals and Outcomes

One of the central goals and outcomes of the seminar was to build a common foundation and understanding for the key challenges of DBMS engineering, and how they can be potentially addressed. To this end, the seminar focused on

- Best practices and challenges in building open source and commercial database engines. Here, the key objectives include a high developer efficiency, mandating quick feedback by tests and verification tools already during feature development, as well as systematic (stress) testing of the software under high load and error conditions.
- The applicability of formal methods and verification tools to DBMS. Formal methods can be of great help to prove the correctness of key database system components such as query compilers, distributed consensus protocols, data replication components, or modules dealing with high availability. Still, an important question is how to systematically identify those components that can benefit from formal verification with reasonable implementation effort, and how to best integrate these methods into existing systems.
- Advanced testing techniques such as fuzzers, query synthesis, and workload generators. These methods allow to significantly increase the test coverage of a DBMS by systematically exploring uncovered code paths and putting stress on individual, important subsystems such as input verification and error handling that are a frequent source of software defects.
- Methods for the automatic generation of test data and testcase reduction. Occasionally, defects in database software are only found by customers running very complex queries operating on confidential data sets. Thus, to allow for problem reproduction, developers benefit from a minimal data set and a simplified query specification that does not disclose confidential data or exhibit unnecessary complexity.
- Security aspects such as ensuring confidentiality and data integrity in the presence of different classes of attackers.

Attendee Mix and Seminar Structure

The seminar lasted 2.5 days. Its format and attendee mix was significantly influenced by the ongoing pandemic. Of the 34 attendees, 13 attended in person and 21 remotely. All but one of the in-person attendees were based in Europe. Overall, we received the highest response rate from Europe (20 attendees), and a lower one from Asia (8 attendees) and the US (6 attendees). We are grateful to the two Video Conference Assistants (VCAs), Jack Clark and Mark Raasveldt, who managed the equipment to ensure a smooth experience for all attendees.

We started the seminar with an introduction round in which every attendee introduced themselves. We held another such session in the late afternoon, to accommodate the US attendees. Prior to the seminar, we contacted attendees to give overview talks to establish a common discussion basis, which was useful given that the attendees came from different

scientific communities. We had such overview talks on the first and second day. On the second and third day, we had in-depth talks. While we had planned breakout sessions, many of the talks were followed by fruitful and unplanned discussions. On the last day, we had a group discussion on the takeaways and future plans.

Future Plans

One major result from the seminar was to identify open problems and areas of future work that the group wants to address in an interdisciplinary manner. Among others, this includes the creation of a reference manual for database engineering groups to avoid redundant work and re-inventing techniques already established (or discarded) by other teams, the identification of database modules (e.g. the query compiler and transaction processing system) that can benefit from formal verification, designing new test oracles to test various data-centric systems for different kind of bugs, as well as the establishment of a common testcase specification format and a test corpus that can be shared between DBMS engineering teams. We discussed proposing another instance of the Dagstuhl seminar to utilize the established discussion basis and work on addressing these specific challenges.

2 Table of Contents

Executive Summary

Maria Christakis, Alexander Böhm, Eric Lo, and Manuel Rigger 20

Overview of Talks

Dynamic Symbolic Execution: An Introduction

Cristian Cadar 25

How to Make Serializable Concurrency Control Protocol Executions Verifiable

Jack Clark 25

Three Ways to Get Test Case Reduction Almost for Free

Alastair F. Donaldson 26

Formal Verification of Databases

Stefania Dumbrava 27

Database Security: Formalization, Verification, and Testing – Challenges and Open Questions

Marco Guarnieri 28

Robust Query Execution with Guarantees

Jayant R. Haritsa 28

Isolation Levels and Isolation Level Testing

Kyle Kingsbury 28

Hyper’s Reliability and Robustness Challenges

Marcel Kost 29

Testing Consensus Implementations

Burcu Kulahcioglu Ozkan 29

Metamorphic Testing of Datalog Engines

Muhammad Numair Mansur, Maria Christakis, and Valentin Wüstholtz 30

DuckDB Testing – Present and Future

Mark Raasveldt and Hannes Mühleisen 30

A Whirlwind Tour of Automated Database Management System Testing

Manuel Rigger 31

Fuzz Testing

Abhik Roychoudhury 31

Big Data, Small Testing

Anupam Sanghi 31

Working groups

Towards Bug-Free DBMS Ecosystems

Mai Zheng, Jack Clark, and Miryung Kim 32

Findings

General Conclusions 33

Open Problems and Future Work 34

24 21442 – Ensuring the Reliability and Robustness of Database Management Systems

Participants 35

Remote Participants 35

3 Overview of Talks

3.1 Dynamic Symbolic Execution: An Introduction

Cristian Cadar (Imperial College London, GB)

License © Creative Commons BY 4.0 International license
© Cristian Cadar

In this overview talk, I give an introduction to dynamic symbolic execution, discussing its key strengths, as well as its main scalability challenges. I also give an overview of some of the many applications of dynamic symbolic execution, and discuss the opportunities it offers for analysing database management systems.

3.2 How to Make Serializable Concurrency Control Protocol Executions Verifiable

Jack Clark (ETH Zürich, CH)

License © Creative Commons BY 4.0 International license
© Jack Clark

Main reference Jack Clark: “Verifying Serializability Protocols With Version Order Recovery”, ETH Zürich, 2021.
URL <https://doi.org/10.3929/ethz-b-000507577>

A core feature of many database systems is the ability to group operations into transactions. An isolation level defines the extent to which operations within a transaction interact with operations from other concurrent transactions. The serializable isolation level provides correctness guarantees that many programmers implicitly assume, since it provides the illusion of transactions running in some sequential order. However, implementing serializable transactions, particularly in a distributed setting, has proven to be a challenging task, with many systems failing to live up to their guarantees.

Unfortunately, verifying that an execution history is serializable is NP-complete. However, with access to a database system’s internal version order, serializability can be efficiently checked. Existing tools for checking the serializability of histories either have exponential running time or require specially crafted operations to be able to recover version order information, which significantly limits the amount of functionality that can be tested. Furthermore, existing tools cannot handle predicate operations which are a key feature of most database systems.

This talk demonstrates that it is possible to recover the version order directly from real database systems and that it can be used to efficiently verify execution histories. Additionally, it is shown that recovering additional object visibility information enables verification of histories that contain predicate operations, a condition which distinguishes the serializable isolation level from weaker levels. Building on these foundations, I demonstrate how we can move towards verifying histories generated by real-world workloads, something not achievable with existing tools and techniques.

3.3 Three Ways to Get Test Case Reduction Almost for Free

Alastair F. Donaldson (Imperial College London, GB)

License © Creative Commons BY 4.0 International license
© Alastair F. Donaldson

Joint work of Alastair F. Donaldson, Paul Thomson, Vasyl Teliman, Stefano Milizia, André Perez Maselco, Antoni Karpinski

Main reference Alastair F. Donaldson, Paul Thomson, Vasyl Teliman, Stefano Milizia, André Perez Maselco, Antoni Karpinski: “Test-case reduction and deduplication almost for free with transformation-based compiler testing”, in Proc. of the PLDI ’21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021, pp. 1017–1032, ACM, 2021.

URL <http://dx.doi.org/10.1145/3453483.3454092>

Randomised testing techniques are good at finding bugs, but tend to produce large bug-inducing test cases that are difficult to understand. Test case reduction is an essential technology for increasing the utility of randomised testing. A test case reduction tool takes a large bug-inducing test case and shrinks it to a smaller test case that still triggers the bug of interest, typically using a variant of the delta debugging algorithm [5].

Test case reduction is challenging when test cases have associated validity constraints. For example, a test case that triggers a miscompilation bug in a C compiler must be free from undefined behaviour. If a test case reducer introduces undefined behaviour when reducing a test case, the result of test case reduction may end up being a nonsensical C program that yields different results across multiple compilers, but due to undefined behaviour in the program rather than due to a compiler bug. The C-Reduce test case reducer for C programs [4] relies on a plethora of external tools to ensure test case validity.

In this talk I give an overview of three ways that test case reduction can be obtained almost for free in a manner that preserves test case validity automatically.

The first is based on a technique called transformation-based testing, used in the spirv-fuzz tool [1]. In this approach a test case comprises an original input and a mutated input where the mutated input is obtained by applying a sequence of semantics-preserving transformations to the original input. A bug is identified when the system under tests treats these equivalent-by-construction inputs differently. Test case reduction then involves using delta debugging to search for a minimal sub-sequence of transformations such that the original and minimally-transformed inputs yield different results. Test case reduction is thus applied on the transformations, rather than on the input; as a result, the approach relies on fuzzing using relatively small seed inputs.

The second is the test case reduction approach employed by the Hypothesis property-based testing tool for Python [3]. This involves performing test case reduction on the sequence of bits that was used to generate a bug-inducing input, searching for a shorter, simpler sequence of bits that, when fed to the generator, yields a smaller, simpler input that still triggers the bug. Because reduced test cases are emitted by the same generator that emitted the original bug-inducing test case, reduced test cases automatically enjoy any validity guarantees that the generator provides.

The final approach, specific to compiler testing, involves using “program reconditioning”. With this approach, undefined behaviours are eliminated from a program after the program is generated, rather than being avoided during generation. Treating the removal of undefined behaviour as a separate “reconditioning” step means that reconditioning can also be used during test case reduction: the test case reducer need not worry about introducing undefined behaviour because before feeding a reduced input to the compilers under test, the input will be reconditioned. This idea was the subject of a recent MSc thesis [2] and is the subject of ongoing work.

My hope is that these ideas may have relevance in the field of database testing, if similar problems of test case validity apply.

References

- 1 Alastair F. Donaldson, Paul Thomson, Vasyl Teliman, Stefano Milizia, André Perez Maselco, and Antoni Karpinski. Test-case reduction and deduplication almost for free with transformation-based compiler testing. In Stephen N. Freund and Eran Yahav, editors, *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, pages 1017–1032. ACM, 2021.
- 2 Bastien Lecoecur. GLSLsmith: A random generator of OpenGL shader programs. Master’s thesis, Imperial College London, 2021.
- 3 David Maciver and Alastair F. Donaldson. Test-case reduction via test-case generation: Insights from the hypothesis reducer (tool insights paper). In Robert Hirschfeld and Tobias Pape, editors, *34th European Conference on Object-Oriented Programming, ECOOP 2020, November 15-17, 2020, Berlin, Germany (Virtual Conference)*, volume 166 of *LIPICs*, pages 13:1–13:27. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.
- 4 John Regehr, Yang Chen, Pascal Cuoq, Eric Eide, Chucky Ellison, and Xuejun Yang. Test-case reduction for C compiler bugs. In Jan Vitek, Haibo Lin, and Frank Tip, editors, *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '12, Beijing, China – June 11 – 16, 2012*, pages 335–346. ACM, 2012.
- 5 Andreas Zeller and Ralf Hildebrandt. Simplifying and isolating failure-inducing input. *IEEE Trans. Software Eng.*, 28(2):183–200, 2002.

3.4 Formal Verification of Databases

Stefania Dumbrava (ENSIIE – Paris & SAMOVAR – Evry, FR)

License © Creative Commons BY 4.0 International license
© Stefania Dumbrava

We highlight three applications of formal methods to building formally verified specifications of relational, deductive, and graph database query engines. We focus, in particular, on theorem proving with the Coq proof assistant. First, in the relational setting, we give an overview of the state of the art and focus on the formal executable specification of the relational database model [1]. Second, in the deductive setting, we present our experience on developing a modular, reusable library of certified engines for different Datalog dialects (stratified and regular), using the SSReflect extension of Coq [2, 3]. Finally, we discuss the proof engineering aspects related to building an incremental engine, capable of performing view maintenance in the graph database setting. We conclude by outlining future perspectives on using such correct-by-construction specifications as trusted oracles against which one can test commercial implementations.

References

- 1 Véronique Benzaken, Evelyne Contejean, Stefania Dumbrava. A Coq Formalization of the Relational Data Model. In Zhong Shao, editor, *Programming Languages and Systems – 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings, volume 8410 of Lecture Notes in Computer Science*, pages 189–208. Springer, 2014.
- 2 Véronique Benzaken, Evelyne Contejean, Stefania Dumbrava. Certifying Standard and Stratified Datalog Inference Engines in SSReflect. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *Interactive Theorem Proving – 8th International Conference, ITP 2017*,

Brasília, Brazil, September 26-29, 2017, Proceedings, volume 10499 of Lecture Notes in Computer Science, pages 171–188. Springer, 2017.

- 3 Angela Bonifati, Stefania Dumbrava, Emilio Jesús Gallego Arias. Certified Graph View Maintenance with Regular Datalog. *Theory Pract. Log. Program.*, 18(3-4):372–389, 2018.

3.5 Database Security: Formalization, Verification, and Testing – Challenges and Open Questions

Marco Guarnieri (IMDEA Software – Madrid, ES)

License  Creative Commons BY 4.0 International license
© Marco Guarnieri

Securing database systems is critical to protect the confidentiality and integrity of the data stored in databases as well as to ensure the security of applications built on top of databases. In this talk, I present an overview of different classes of attacker models for database systems ranging from honest-but-curious attackers that interact with a database following the SQL standard to attackers compromising database-backed applications. For each attacker, I overview existing security mechanisms, attacks bypassing them, and I discuss challenges in testing and verification of security mechanisms.

3.6 Robust Query Execution with Guarantees

Jayant R. Haritsa (Indian Institute of Science – Bangalore, IN)

License  Creative Commons BY 4.0 International license
© Jayant R. Haritsa

Robust query processing with strong performance guarantees is an extremely desirable objective in the design of industrial-strength database engines. However, it has proved to be a largely intractable and elusive challenge despite sustained efforts spanning several decades. In this talk, we show how the use of a radically different approach involving application of geometric techniques on execution-time profiles can provide provable guarantees on worst-case performance. Further, these guarantees are independent of data distributions and query structures.

3.7 Isolation Levels and Isolation Level Testing

Kyle Kingsbury (San Francisco, US)

License  Creative Commons BY 4.0 International license
© Kyle Kingsbury

Users who care about their data store it in databases, which (at least in principle) guarantee some form of transactional isolation. However, experience shows that many databases do not provide the isolation guarantees they claim. With the recent proliferation of new distributed databases, demand has grown for checkers that can, by generating client workloads and injecting faults, produce anomalies that witness a violation of a stated guarantee. An ideal checker would be sound (no false positives), efficient (polynomial in history length

and concurrency), effective (finding violations in real databases), general (analyzing many patterns of transactions), and informative (justifying the presence of an anomaly with understandable counterexamples). Sadly, we are aware of no checkers that satisfy these goals. We present Elle: a novel checker which infers an Adya-style dependency graph between client-observed transactions. It does so by carefully selecting database objects and operations when generating histories, so as to ensure that the results of database reads reveal information about their version history. Elle can detect every anomaly in Adya et al’s formalism (except for predicates), discriminate between them, and provide concise explanations of each.

3.8 Hyper’s Reliability and Robustness Challenges

Marcel Kost (Salesforce – München, DE)

License © Creative Commons BY 4.0 International license
© Marcel Kost

Hyper is the data engine of Tableau, an interactive visual analytics application. It transparently aggregates the user’s data using the SQL queries generated by the visual frontend, which can become arbitrarily complex.

Hyper is known for its code generation, where SQL queries are compiled to machine code for faster execution. Making this technique robust is a major challenge, since this makes the DBMS prone to hard crashes, and many existing tools like sanitizers can’t be applied to run-time generated code.

In addition Hyper is used as part of Tableau’s cloud offering, which introduces a large number of cloud-related reliability challenges. The biggest challenge besides general cloud-native problems like deployment and monitoring is efficiently handling multitenancy while still guaranteeing high availability. To master this challenge, a lot of work in resource governance and load balancing is required, probably leading to distributed query processing.

3.9 Testing Consensus Implementations

Burcu Kulahcioglu Ozkan (TU Delft, NL)

License © Creative Commons BY 4.0 International license
© Burcu Kulahcioglu Ozkan

Joint work of Cezara Dragoi, Constantin Enea, Burcu Kulahcioglu Ozkan, Rupak Majumdar, Filip Niksic
Main reference Cezara Dragoi, Constantin Enea, Burcu Kulahcioglu Ozkan, Rupak Majumdar, Filip Niksic:
“Testing consensus implementations using communication closure”, Proc. ACM Program. Lang.,
Vol. 4(OOPSLA), pp. 210:1–210:29, 2020.
URL <http://dx.doi.org/10.1145/3428278>

Distributed database systems rely on the coordination and agreement of distributed nodes to provide certain guarantees to the application developers. However, it is difficult to design and implement distributed systems correctly to meet their guarantees. They are error-prone since developers have to reason about a large number of possible event interleaving’s due to asynchronous message-based communication, arbitrary message delays, message losses, network partitions, and node failures.

Current testing techniques focus on systematic or randomized exploration of all executions of an implementation while treating the implemented algorithms as black boxes. On the other hand, proofs of correctness of many of the underlying algorithms often exploit

semantic properties that reduce reasoning about correctness to a subset of behaviors. For example, the communication-closure property, used in many proofs of distributed consensus algorithms, shows that every asynchronous execution of the algorithm is equivalent to a lossy synchronous execution, thus reducing the burden of proof to only that subset. We formulate the communication-closure hypothesis, which states that bugs in implementations of distributed consensus algorithms will already manifest in lossy synchronous executions, and present a testing algorithm based on this hypothesis. We show that a random testing algorithm based on sampling lossy synchronous executions can empirically find a number of bugs, including previously unknown ones.

In this talk, I introduce the key ideas in our algorithm for testing consensus implementations that are fundamental in many distributed database systems.

3.10 Metamorphic Testing of Datalog Engines

Muhammad Numair Mansur (MPI-SWS – Kaiserslautern, DE), Maria Christakis (MPI-SWS – Kaiserslautern, DE), and Valentin Wüstholtz

License  Creative Commons BY 4.0 International license
© Muhammad Numair Mansur, Maria Christakis, and Valentin Wüstholtz

Datalog is a popular query language with applications in several domains. Like any complex piece of software, Datalog engines may contain bugs. The most critical ones manifest as incorrect results when evaluating queries—we refer to these as query bugs. Given the wide applicability of the language, query bugs may have detrimental consequences, for instance, by compromising the soundness of a program analysis that is implemented and formalized in Datalog. In this talk, I present the first metamorphic-testing approach for detecting query bugs in Datalog engines. We ran our tool on three mature engines and found 13 previously unknown query bugs, some of which are deep and revealed critical semantic issues.

3.11 DuckDB Testing – Present and Future

Mark Raasveldt (CWI – Amsterdam, NL) and Hannes Mühleisen (CWI – Amsterdam, NL)

License  Creative Commons BY 4.0 International license
© Mark Raasveldt and Hannes Mühleisen

DuckDB is a fast analytical embedded database system. For users it is crucial that the system is both correct and fast. In this talk we discuss how the different components of the extensive test suite for DuckDB works, and how we use it to ensure the reliability of the system. In addition, we also talk about the fuzzers that we run and the way in which we do performance regression testing.

3.12 A Whirlwind Tour of Automated Database Management System Testing

Manuel Rigger (ETH Zürich, CH)

License © Creative Commons BY 4.0 International license
© Manuel Rigger

One prime approach to ensuring the reliability of Database Management Systems (DBMSs) is automated testing. Automated testing can find bugs without user interaction, but not guarantee their absence. Generally speaking, such approaches involve generating a test case, validating the test case's result (which is known as a test oracle), and then reducing the bug to a minimal version. Testing DBMSs typically involves generating a database, a query, and then validating the query's result. In this talk, I will give an overview of the automated testing landscape focusing on the available test oracles. My goal is to convey a simplified overview of the ideas that have been tried and point out works by the seminar's attendees and how they connect. Furthermore, I will do a deep dive into the Ternary Logic Partitioning approach that we designed.

3.13 Fuzz Testing

Abhik Roychoudhury (National University of Singapore, SG)

License © Creative Commons BY 4.0 International license
© Abhik Roychoudhury
URL <https://www.comp.nus.edu.sg/~abhik/projects/Fuzz/>

Fuzz testing, proposed by Barton Miller, is a popular technology for finding bugs in software systems via (biased) random search over the domain of inputs. In this talk, we will review the technology of fuzz testing – specifically its variants in the form of blackbox, greybox and whitebox fuzzing. We will discuss coverage based greybox fuzzing, which conducts a biased random search over inputs guided by a fitness function to approximate code coverage. Recent developments in fuzzing such as directed fuzzing to reach specific code locations, and smart fuzzing to test applications processing structured data will be discussed. Overall, we will discuss the pro-s and con-s of fuzzing technology and take a forward looking view where it can be tuned to find deeper bugs such as violations of non-trivial temporal properties.

3.14 Big Data, Small Testing

Anupam Sanghi (Indian Institute of Science – Bangalore, IN)

License © Creative Commons BY 4.0 International license
© Anupam Sanghi
Joint work of Anupam Sanghi, Raghav Sood, Jayant R. Haritsa, Srikanta Tirthapura, Shadab Ahmed
Main reference Anupam Sanghi, Raghav Sood, Jayant R. Haritsa, Srikanta Tirthapura: “Scalable and Dynamic Regeneration of Big Data Volumes”, in Proc. of the 21st International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018, pp. 301–312, OpenProceedings.org, 2018.
URL <http://dx.doi.org/10.5441/002/edbt.2018.27>

Synthetic databases are required in a variety of industrial use-cases, ranging from testing and tuning database engines and applications to system benchmarking. In the past decade, several frameworks have advocated modeling data synthesis using a set of cardinality constraints.

Specifically, a cardinality constraint dictates that the output of a given relational expression over the generated database should feature a specified number of rows. We begin this talk with an overview of these frameworks and their current limitations. Then, we present in detail, Hydra, our proposed data generation framework. Hydra constructs a minuscule database summary from the input cardinality constraints, and leverages this summary to dynamically generate data during query execution without explicitly instantiating, storing, and loading the entire database. Adopting this online approach helps to eliminate the time and space overheads typically associated with data synthesis. Finally, to complement dynamic generation, Hydra ensures that the summary generation algorithm is data-scale-free, making it suitable for Big Data environments.

4 Working groups

4.1 Towards Bug-Free DBMS Ecosystems

Mai Zheng (Iowa State University – Ames, US), Jack Clark (ETH Zürich, CH), and Miryung Kim (UCLA, US)

License  Creative Commons BY 4.0 International license
© Mai Zheng, Jack Clark, and Miryung Kim

This abstract summarizes the results of a virtual group discussion during the Dagstuhl Seminar 21442 “Ensuring the Reliability and Robustness of Database Management Systems”. Participants include Jack Clark (ETH Zürich, CH), Miryung Kim (University of California, Los Angeles, US), and Mai Zheng (Iowa State University, US).

Database Management Systems (DBMS) do not work in isolation. They typically rely on the underlying operating systems (OS) to provide file, networking, and other services. For example, Lightning DB stores a B+ tree as a .mdb file in the file system and uses the mmap syscall to map the .mdb file into memory; moreover, it relies on the fsync and other syscalls to achieve ACID (i.e., atomicity, consistency, isolation, durability) guarantees. Therefore, it is important to take the execution environment of DBMS into account when testing DBMS in practice (i.e., the entire DBMS ecosystem).

Unfortunately, the interactions between DBMS and the surrounding system is complicated. For example, SQLite maintains both data and log files in the file system (FS) and uses a combination of unlink, fsync and other syscalls to implement transactions. But the ACID properties of the transactions may be violated unexpectedly for a number of reasons including the ambiguity in POSIX specifications and DBMS configurations [1, 2]. Similar issues have also been observed on other widely used DBMS ecosystems [1, 2]. How to address the inherent dependency and ambiguity in DBMS ecosystems remains an open question.

The two automatic testing approaches above [1, 2] show the preliminary effectiveness, but they fall short of coverage. Recent efforts have applied formal methods to address the challenge. For example, FSCQ shows that it is possible to formally verify the crash consistency property of a complete file system [3]. However, the verified FS is still much simpler than the POSIX file systems (e.g., Ext4) supporting various DBMS in practice. Also, follow-up research has exposed a bug in the verified FS via fuzzing [4], which implies the difficulty in achieving bug-free DBMS ecosystems. Most recently, Amazon [5] applies lightweight formal methods to validate a key-value (KV) store node in Amazon S3 service, which demonstrates the feasibility of verifying the correctness of an entire storage node.

Unfortunately, while promising, the approach cannot be easily extended to general DBMS ecosystems due to a number of constraints (e.g., it is highly customized for Rust-based S3 KV store).

Addressing the challenges of testing DBMS ecosystems will likely require the expertise and collaboration across different communities including databases, formal methods, file systems, etc. Given the prime importance of DBMS ecosystems, we call for communities' collective efforts in examining the cross-layer challenges and coming up with practical solutions.

References

- 1 Thanumalayan Sankaranarayana Pillai, Vijay Chidambaram, Ramnatthan Alagappan, Samer Al-Kiswany, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. All file systems are not created equal: On the complexity of crafting crash-consistent applications. In Jason Flinn and Hank Levy, editors, *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014*, pages 433-448. USENIX Association, 2014.
- 2 Mai Zheng, Joseph Tucek, Dachuan Huang, Feng Qin, Mark Lillibridge, Elizabeth S. Yang, Bill W. Zhao, and Shashank Singh. Torturing databases for fun and profit. In Jason Flinn and Hank Levy, editors, *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014*, pages 449-464. USENIX Association, 2014.
- 3 Haogang Chen, Daniel Ziegler, Tej Chajed, Adam Chlipala, M. Frans Kaashoek, and Nikolai Zeldovich. Using Crash Hoare logic for certifying the FSCQ file system. In Ajay Gulati and Hakim Weatherspoon, editors, *2016 USENIX Annual Technical Conference, USENIX ATC 2016, Denver, CO, USA, June 22-24, 2016*. USENIX Association, 2016.
- 4 Seulbae Kim, Meng Xu, Sanidhya Kashyap, Jungyeon Yoon, Wen Xu, and Taesoo Kim. Finding semantic bugs in file systems with an extensible fuzzing framework. In Tim Brecht and Carey Williamson, editors, *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*, pages 147-161. ACM, 2019.
- 5 James Bornholt, Rajeev Joshi, Vytautas Astrauskas, Brendan Cully, Bernhard Kragl, Seth Markle, Kyle Sauri, Drew Schleit, Grant Slatton, Serdar Tasiran, Jacob Van Geffen, and Andrew Warfield. Using lightweight formal methods to validate a key-value storage node in Amazon S3. In Robbert van Renesse and Nikolai Zeldovich, editors, *SOSP'21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021*, pages 836-850. ACM, 2021.

5 Findings

In this section, we highlight some of the general conclusions the participants derived from the seminar, and discuss open problems and future work to follow up on.

5.1 General Conclusions

During the discussion, various database developers from both industry and academia shared their quality assurance and testing strategies. In particular, they highlighted both the importance of sophisticated testing strategies for finding defects early, as well as the enormous effort (considering both hardware resources and manual labor) that is put into this part of the system development process. Still, regressions and defects that are only found late in the

process or even by customers continue to be a huge challenge even for very mature systems that are in active development for several decades. Despite this obvious importance of the topic, there is **almost no sharing of best practices between database development teams** and only a few publications on the topic. For commercial vendors, their quality assurance strategy is usually seen as a business secret, whereas software quality topics are often not considered interesting enough to publish by open-source software teams and academic development groups.

While **formal methods and verification techniques** have shown their practical benefits in many domains such as aviation, embedded, and real-time system (to only name a few), their **practical application in the database system space is still surprisingly limited**.

Automated testing including sophisticated techniques for defect finding such as semantic fuzzing and error injection plays a key role in most database development teams already today. Still, a **major challenge is to find good oracles that can reliably distinguish between an expected and faulty outcome of a randomly generated, complex testcase**. While the current state of the art is helpful to uncover low-level defects in the system implementations (i.e. software crashes that continue to be an important defect class in many systems), more work is needed in order to uncover more high-level, semantic problems such as wrong query results. In particular, such semantical tests would be of significant help to improve the reliability and robustness of multi-node deployments of databases: These distributed DBMS suffer from additional system complexity by allowing for additional classes of errors such as faulty network communication or requiring distributed coordination protocols between the nodes, to only name two prominent examples.

5.2 Open Problems and Future Work

As mentioned above, the seminar helped the participants to uncover open problems and challenges for the development of reliable and robust database software. Below, we list several topics that participants of the seminar expressed their interest in working on.

There is a clear need to **establish a handbook on best practises for DBMS development teams** that provides comprising guidance of the current state of the art of engineering reliable and robust database systems. We hope that such a reference will allow teams to pick from a bouquet of available techniques depending on their requirements and software complexity. This also includes examples of techniques that were evaluated and did not work out, so that teams can learn from the negative experiences made by other engineering groups.

To address the limited use of formal verification in the DBMS context, we plan to **identify key components in the architecture of DBMS that lend themselves to the use of formal methods**. This potentially includes the query optimizer, where formal methods can help to provide semantic equivalence between optimized and non-optimized query plans, as well as for verifying key characteristics of distributed transaction management, consensus protocols, or replication mechanisms. By providing practical examples from the DBMS domain, we hope to lower the adoption hurdle for system development teams.

Over time, each database management system tends to accumulate a large amount of performance, scalability, and correctness tests. Usually, these tests are in a proprietary format and not shared between systems. By proposing a **standardized format for a high-level test specification** and providing a centralized repository, we hope to foster sharing and re-use between different software development teams.

Participants

- Alexander Böhm
SAP SE – Walldorf, DE
- Cristian Cadar
Imperial College London, GB
- Alastair F. Donaldson
Imperial College London, GB
- Stefania Dumbrava
ENSIIE – Paris & SAMOVAR –
Evry, FR
- Marco Guarnieri
IMDEA Software – Madrid, ES
- Marcel Kost
Salesforce – München, DE
- Burcu Kulahcioglu Ozkan
TU Delft, NL
- Hannes Mühleisen
CWI – Amsterdam, NL
- Danica Porobic
Oracle Labs –
6Redwood Shores, US
- Mark Raasveldt
CWI – Amsterdam, NL
- Manuel Rigger
ETH Zürich, CH
- Anupam Sanghi
Indian Institute of Science –
Bangalore, IN



Remote Participants

- Artur Andrzejak
Universität Heidelberg, DE
- Chee-Yong Chan
National University of
Singapore, SG
- Yongheng Chen
Georgia Institute of Technology –
Atlanta, US
- Maria Christakis
MPI-SWS – Kaiserslautern, DE
- Jack Clark
ETH Zürich, CH
- Jens Dittrich
Universität des Saarlandes –
Saarbrücken, DE
- Paolo Guagliardo
University of Edinburgh, GB
- Jayant R. Haritsa
Indian Institute of Science –
Bangalore, IN
- Miryung Kim
UCLA, US
- Kyle Kingsbury
San Francisco, US
- Greg Law
Undo – Cambridge, GB
- Si Liu
ETH Zürich, CH
- Eric Lo
The Chinese University of
Hong Kong, HK
- Muhammad Numair Mansur
MPI-SWS – Kaiserslautern, DE
- Zhou Qiang
PingCAP – Hangzhou, CN
- Tilmann Rabl
Hasso-Plattner-Institut,
Universität Potsdam, DE
- Abhik Roychoudhury
National University of
Singapore, SG
- Zhendong Su
ETH Zürich, CH
- S. Sudarshan
Indian Institute of Technology –
Mumbai, IN
- Tao Xie
Peking University, CN
- Tianyin Xu
University of Illinois –
Urbana-Champaign, US
- Mai Zheng
Iowa State University –
Ames, US