Report from Dagstuhl Seminar 18091

Data Consistency in Distributed Systems: Algorithms, Programs, and Databases

Edited by

Annette Bieniusa[1], Alexey Gotsman[2], Bettina Kemme[3], and Marc Shapiro[4]

1 **TU Kaiserslautern, DE,** bieniusa@cs.uni-kl.de
2 **IMDEA Software – Madrid, ES,** alexey.gotsman@imdea.org
3 **McGill University – Montreal, CA,** kemme@cs.mcgill.ca
4 **Sorbonne-Université – LIP6 & Inria – Paris, FR,** marc.shapiro@acm.org

Abstract

For decades distributed computing has been mainly an academic subject. Today, it has become mainstream: our connected world demands applications that are inherently distributed, and the usage of shared, distributed, peer-to-peer or cloud-computing infrastructures are increasingly common. However, writing distributed applications that are both correct and well distributed (e.g., highly available) is extremely challenging.

In fact, there exists a fundamental trade-off between data consistency, availability, and the ability to tolerate failures. This trade-off has implications on the design of the entire distributed computing infrastructure, including storage systems, compilers and runtimes, application development frameworks and programming languages. Unfortunately, this also has significant implications on the programming model exposed to the designers and developers of applications. We need to enable programmers who are not experts in these subtle aspects to build distributed applications that remain correct in the presence of concurrency, failures, churn, replication, dynamically-changing and partial information, high load, absence of a single line of time, etc.

This Dagstuhl Seminar proposes to bring together researchers and practitioners in the areas of distributed systems, programming languages, verifications, and databases. We would like to understand the lessons learnt in building scalable and correct distributed systems, the design patterns that have emerged, and explore opportunities for distilling these into programming methodologies, programming tools, and languages to make distributed computing easier and more accessible.

Main issues in discussion:

Application writers are constantly making trade-offs between consistency and availability. What kinds of tools and methodologies can we provide to simplify this decision making? How does one understand the implications of a design choice? Available systems are hard to design, test and debug. Do existing testing and debugging tools suffice for identifying and isolating bugs due to weak consistency? How can these problems be identified in production using live monitoring? Can we formalize commonly desired (generic) correctness (or performance) properties? How can we teach programmers about these formalisms and make them accessible to a wide audience? Can we build verification or testing tools to check that systems have these desired correctness properties? How do applications achieve the required properties, while ensuring adequate performance, in practice? What design patterns and idioms work well? To what degree can these properties be guaranteed by the platform (programming language, libraries, and runtime system)? What are the responsibilities of the application developer, and what tools and information does she have?

Seminar February 25–March 2, 2018 – https://www.dagstuhl.de/18091
2012 ACM Subject Classification Information systems → Database design and models, Information systems → Data structures, Information systems → Storage replication, Computer

Except where otherwise noted, content of this report is licensed
under a Creative Commons BY 3.0 Unported license

Data Consistency in Distributed Systems: Algorithms, Programs, and Databases, *Dagstuhl Reports*, Vol. 8, Issue 02, pp. 101–121
Editors: Annette Bieniusa, Alexey Gotsman, Bettina Kemme, and Marc Shapiro
DAGSTUHL REPORTS Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

systems organization → Parallel architectures, Computer systems organization → Distributed architectures

Keywords and phrases consistency, CRDTs, Distributed Algorithms, distributed computing, Distributed Systems, partitioning, replication, Strong Consistency, transactions, Weak Consistency

Digital Object Identifier 10.4230/DagRep.8.2.101

Edited in cooperation with Manuel Bravo

1 Executive Summary

Annette Bieniusa (TU Kaiserslautern, DE)
Alexey Gotsman (IMDEA Software – Madrid, ES)
Bettina Kemme (McGill University – Montreal, CA)
Marc Shapiro (Sorbonne-Université – LIP6 & Inria – Paris, FR)

License Ⓒ Creative Commons BY 3.0 Unported license
© Annette Bieniusa, Alexey Gotsman, Bettina Kemme, and Marc Shapiro

Large-scale distributed systems have become ubiquitous, and there are a variety of options to develop, deploy, and operate such applications. Typically, this type of application is data-centric: it retrieves, stores, modifies, forwards, and processes data from different sources. However, guaranteeing availability, preventing data loss, and providing efficient storage solutions are still major challenges that a growing number of programmers are facing when developing large-scale distributed systems. In our seminar, we brought together academic and industrial researchers and practitioners to discuss the status quo of data consistency in distributed systems. As result of talks and discussions, we identified several topics of interest that can be grouped into the following four areas.

Theoretical foundations: The seminar included a tutorial on specification of consistency guarantees provided by distributed systems and talks on comparing different styles of specification and expressing replicated data type semantics in Datalog. Different specification styles are suitable for different purposes and more work is needed to identify the most appropriate ones. The seminar also included talks on formally reasoning about which consistency levels are enough to satisfy correctness properties of applications. The talks demonstrated that formal verification is a promising approach to cope with the challenge of selecting appropriate consistency levels.

Distributed systems and database technologies: With the growing number of replicated data stores, the two fields of distributed systems and databases are moving closer together. The communities should be made more aware of each others results. A common concern in agreement, i.e., ensuring that database copies are updated correctly. Traditionally, the distributed systems community has based many of their approaches on classical consensus algorithms or looked at weaker consistency models. In contrast, database systems focused most work on 2-phase commit protocols and eager update protocols. At the same time, the database community also considered other ACID aspects that required to combine commit protocols with concurrency control protocols and recovery schemes. In the last decade however, and in particular with practical implementations of the Paxos consensus algorithms, and the use of file replication in storage systems for availability, work of the two communities has come closer together. A challenge in this context is that work that emerges from the different communities still makes slightly different assumptions about failure and correctness models. They can often be quite subtle so that the differences are not obvious, even to the experts. And they can lead to very different approaches to find solutions. Bridging this gap in terms of understanding each other, and the implications of correctness and failure models remains a challenging task. As an example, the separation of the concepts of atomicity, isolation and durability in the database world offers many opportunities for optimization, but includes extra complexity when analyzing which algorithms are appropriate in which situations.

Conflict-handling in highly-scalable systems: In the last years, conflict-free replicated data types (CRDTs) have been adopted by an ever-growing number of products and companies to deal with high-availability requirements under concurrent modifications of data. Recent advances in related techniques for collaborative editing might make it possible that hundreds of people work together on a shared document or data item with limited performance impact. Several talks presented programming guidelines, static analyses, and related tools for safe usage of CRDTs in situations where eventual consistency is not enough to maintain application invariants.

Programming models for distributed systems: Micro-services have become a standard approach for constructing large-scale distributed systems, though microservice composition and scalability raises a lot of questions. Some presentations discussed current work on actor-based and data-flow programming. Design for testability and test frameworks are crucial for providing reliable services, but they currently require a lot of experience as of today. We believe that future progress on programming models and new results in theoretical foundations will help to simplify this challenging task and support programmers in building safe systems.

2 Table of Contents

Executive Summary

Overview of Talks

Working groups

3 Overview of Talks

3.1 Does your fault-tolerant distributed system tolerate faults?

Peter Alvaro (University of California – Santa Cruz, US)

License Ⓒ Creative Commons BY 3.0 Unported license
© Peter Alvaro

Joint work of Peter Alvaro, Josh Rosen, Joseph M. Hellerstein, Kolton Andrus
Main reference Peter Alvaro, Kolton Andrus, Chris Sanden, Casey Rosenthal, Ali Basiri, Lorin Hochstein: "Automating Failure Testing Research at Internet Scale", in Proc. of the Seventh ACM Symposium on Cloud Computing, Santa Clara, CA, USA, October 5-7, 2016, pp. 17–28, ACM, 2016.
URL http://dx.doi.org/10.1145/2987550.2987555

Large-scale distributed systems must be built to anticipate and mitigate a variety of hardware and software failures. In order to build confidence that fault-tolerant systems are correctly implemented, an increasing number of large-scale sites regularly run failure drills in which faults are deliberately injected in production or staging systems. While fault injection infrastructures are becoming relatively mature, existing approaches either explore the combinatorial space of potential failures randomly or exploit the "hunches" of domain experts to guide the search. Random strategies waste resources testing "uninteresting" faults, while programmer-guided approaches are only as good as the intuition of a programmer and only scale with human effort.

In this talk, I will present intuition, experience and research directions related to lineage-driven fault injection (LDFI), a novel approach to automating failure testing. LDFI utilizes existing tracing or logging infrastructures to work backwards from good outcomes, identifying redundant computations that allow it to aggressively prune the space of faults that must be explored via fault injection. I will describe LDFI's theoretical roots in the database research notion of provenance, present early results from the field, and present opportunities for near- and long-term future research.

3.2 The FuzzyLog Approach to Building Distributed Services

Mahesh Balakrishnan (Yale University – New Haven, US)

License Ⓒ Creative Commons BY 3.0 Unported license
© Mahesh Balakrishnan

Control plane applications such as coordination services, SDN controllers, filesystem namespaces, and big data schedulers have strong requirements for consistency as well as performance. Building such applications is currently a black art, requiring a slew of complex distributed protocols that are inefficient when layered and difficult to combine. The shared log approach achieves simplicity for distributed applications by replacing complex protocols with a single shared log; however, it does so by introducing a global ordering over all updates in the system, which can be expensive, unnecessary, and sometimes impossible. We propose the FuzzyLog abstraction, which provides applications the simplicity of a shared log without its drawbacks. The FuzzyLog allows applications to construct and access a durable, iterable partial order of updates in the system. FuzzyLog applications retain the simplicity of their shared log counterparts while extracting parallelism, providing a range of consistency guarantees and tolerating network partitions. In effect, the FuzzyLog is a democratizing abstraction for building scalable, robust distributed systems.

3.3 Highly available applications done correctly

Annette Bieniusa (TU Kaiserslautern, DE)

License ⓒ Creative Commons BY 3.0 Unported license
© Annette Bieniusa
Joint work of Mathias Weber, Peter Zeller, Arnd Poetzsch-Heffter
Main reference Mathias Weber, Annette Bieniusa, Arnd Poetzsch-Heffter: "EPTL – A Temporal Logic for Weakly Consistent Systems (Short Paper)", in Proc. of the Formal Techniques for Distributed Objects, Components, and Systems – 37th IFIP WG 6.1 International Conference, FORTE 2017, Held as Part of the 12th International Federated Conference on Distributed Computing Techniques, DisCoTec 2017, Neuchâtel, Switzerland, June 19-22, 2017, Proceedings, Lecture Notes in Computer Science, Vol. 10321, pp. 236–242, Springer, 2017.
URL http://dx.doi.org/10.1007/978-3-319-60225-7_17

The construction of highly available applications poses challenges to developers and software architects. Reasoning about the correctness of such systems requires special care when it comes to security aspects.

In our talk, we discuss different aspects that arise in the practise of developing highly available systems in the context of the AntidoteDB, a highly-available transactional CRDT data store. We will show how programmers can use the Repliss tool for specifying the semantics of their programs and check what type of consistency is required for maintaining invariants in their application. Further, we introduce a novel temporal, event-based parallel temporal logic (EPTL) that allows to specify weakly-consistent systems. In contrast to temporal logics like LTL or CTL, EPTL can model semantics of components that are truly concurrent while abstracting from implementation and communication details such as causality tracking mechanisms. As a third contribution, we present an access control mechanism for providing secure access to data items under causal consistency together with its specification EPTL.

3.4 Towards Affordable Externally Consistent Guarantees for Geo-Replicated Systems

Manuel Bravo (INESC-ID – Lisbon, PT) and Luis Rodrigues (INESC-ID – Lisbon, PT)

License ⓒ Creative Commons BY 3.0 Unported license
© Manuel Bravo and Luis Rodrigues

Cloud services's designers are faced with a dilemma: either favor low latency adopting weaker consistency models such as eventual and causal consistency; or favor strong consistency imposing higher latency responses. A promising approach to alleviate the tension between semantics and performance consists in allowing multiple consistency levels to coexist.

We propose a novel consistency model, namely external causality, that takes causal consistency and spice it up with affordable externally consistent guarantees. The idea behind external causality is that most operations, namely internal operations, are executed locally (in a single site) and asynchronously replicated. Nevertheless, stronger operations called external operations, which provide externally consistent guarantees, coexist with internal operations. An external operation is ordered after any other operation—both internal and externals–already successfully installed in the system as of the time the external operation began. External operations allow developers to make stronger assumptions. Our hope is that external causality can potentially simplify the development of applications without compromising performance.

3.5 A Tutorial on Specifications for Distributed Services

Sebastian Burckhardt (Microsoft Research – Redmond, US)

License Ⓒ Creative Commons BY 3.0 Unported license
© Sebastian Burckhardt
URL https://1drv.ms/f/s!AgoBH3oDy8d-iIh86SHxOpzKAbRdfg

Applications are increasingly developed as a composition of services, with advanced distributed protocols hidden beneath simple service APIs. Any service (whether it is cloud storage, an advanced CRDTs, or an application-defined microservice) must however somehow specify a semantics under concurrent and/or distributed accesses, which is nontrivial in the presence of consistency relaxations, such as lazy replication and asynchronous update propagation. In this tutorial, I give an introduction to an advanced specification methodology for service semantics, how it can be used to specify the behavior of typical CRDTs and collaborative editing, and how it has helped us to clarify the terminology and prove correctness and optimality of implementations.

3.6 Building Elastic Micro-Services with Orleans, now Geo-Distributed

Sebastian Burckhardt (Microsoft Research – Redmond, US)

License Ⓒ Creative Commons BY 3.0 Unported license
© Sebastian Burckhardt
Joint work of Philip A. Bernstein, Sebastian Burckhardt, Sergey Bykov, Natacha Crooks, Jose M. Faleiro, Gabriel Kliot, Alok Kumbhare, Muntasir Raihan Rahman
Main reference Philip A. Bernstein, Sebastian Burckhardt, Sergey Bykov, Natacha Crooks, Jose M. Faleiro, Gabriel Kliot, Alok Kumbhare, Muntasir Raihan Rahman, Vivek Shah, Adriana Szekeres, Jorgen Thelin: "Geo-distribution of actor-based services", PACMPL, Vol. 1(OOPSLA), pp. 107:1–107:26, 2017.
URL http://dx.doi.org/10.1145/3133931

Virtual actor frameworks, such as the Orleans system, have proven quite useful to build elastically scalable micro-services. However, it is not a priori clear how to use them in a geo-distributed setting with high communication latency. To this end, we have developed 2 extensions to the model, one with and one without actor replication. The replicated version, which supports reading and updating with a choice of linearizable and eventual consistency. Our evaluation on several workloads shows the advantage of offering varying configuration choices: for example, replication can provide fast, always-available reads and updates globally, while batching of linearizable storage accesses at a single location can boost the throughput of an order processing workload by 7x.

3.7 Comparing Specification Styles for Transactional Consistency Models

Andrea Cerone (Imperial College London, GB)

License Ⓒ Creative Commons BY 3.0 Unported license
© Andrea Cerone

We compare three different frameworks for specifying weak consistency models of databases whose transactions enjoy atomic visibility.

The first framework allows for declarative specifications of consistency models by placing constraints, or axioms, over abstract executions. Abstract executions were originally introduced by Burckhardt et al. [1].

The second framework is based on Adya's dependency graphs [3]: consistency models are specified by considering only those dependency graphs that forbid cycles of a certain form. I show that, for a particular class of specifications of consistency models given in terms of abstract executions, it is possible to automatically infer an acyclicity condition that captures the same consistency models using dependency graphs; such an acyclicity condition is encoded as an irreflexive relation in a recursive variant of Tarki's calculus of binary relations, with transaction dependencies as ground terms. Complete details of this result are given in [2]. I also conjecture that, in the general case, Tarki's calculus of binary relations is not expressive enough to capture consistency models that can be specified using axioms over abstract executions.

The third framework is based on a novel notion of history heaps, which I recently developed together with P. Gardner and S. Xiong. History heaps record, for each object in the database, the whole list of versions that have been written by transactions for such an object; versions also contain the meta-data corresponding to the transactions that accessed such a version. Consistency models are specified in an operational way: history heaps are used to encode an abstract view of the state of the database accessed by transactions, while a transition relation between history heaps describes how the system may evolve when executing a transaction. I show that specifications of consistency models using dependency graphs can be easily converted into equivalent specifications given in terms of history heaps.

References

1 S. Burckhardt, D. Leijen, M. Fähndrich, M. Sagiv. *Eventually Consistent Transactions.* ESOP, 2012.

2 A. Cerone, A. Gotsman, H. Yang. *Algebraic Laws for Weak Consistency.* CONCUR, 2017.

3 A. Adya. *Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions.* Ph.D. Thesis, MIT 1999.

3.8 Low Latency vs Strong Semantics in Causal Consistency: Protocols and trade-offs

Diego Didona (EPFL – Lausanne, CH)

License 🅭 Creative Commons BY 3.0 Unported license
© Diego Didona
Joint work of Willy Zwaenepoel, Jingjing Wang, Rachid Guerraoui
Main reference Diego Didona, Rachid Guerraoui, Jingjing Wang, Willy Zwaenepoel: "Causal Consistency and Latency Optimality: Friend or Foe?", CoRR, Vol. abs/1803.04237, 2018.
URL http://arxiv.org/abs/1803.04237

Causal consistency is appealing because it is among the strongest consistency levels compatible with availability, and avoids the performance penalties incurred by strongly consistent systems. Yet existing causal consistency designs either sacrifice scalability or low latency to support stronger semantics, e.g., generic read-write transactions or read-only transactions.

In this talk we will present scalable approaches to achieve low latency by means of nonblocking read operations. These approaches apply to systems that support generic and read-only transactions. Then, we will analyze so called "latency optimal" read-only transaction designs. We find that, surprisingly, latency-optimal designs can perform worse than non-optimal ones. To explain this result, we will present a theorem that shows that latency-optimal read-only transactions incur an unavoidable overhead that grows with the number of clients, thus reducing the overall system efficiency.

3.9 Paxos on the Edge

Amr El Abbadi (University of California – Santa Barbara, US), Divyakant Agrawal, and Faisal Nawab

License 🅭 Creative Commons BY 3.0 Unported license
© Amr El Abbadi, Divyakant Agrawal, and Faisal Nawab

Joint work of Faisal Nawab, Divyakant Agrawal, Amr El Abbadi
Main reference Faisal Nawab, Divyakant Agrawal, Amr El Abbadi: "DPaxos: Managing Data Closer to Users for Low-Latency and Mobile Applications", in Proc. of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018, pp. 1221–1236, ACM, 2018.
URL http://dx.doi.org/10.1145/3183713.3196928

The utilization of edge nodes is inevitable for the success and growth of many emerging low latency and mobile applications. In this talk, we will explore a novel Paxos-based consensus protocol that manages access to partitioned data across globally-distributed datacenters and edge nodes. The main objective is to reduce the latency of serving user requests, while ensuring fault-tolerance and adapting gracefully to mobility. These goals are achieved by proposing changes to the traditional Paxos protocol that reduce the size of quorums needed to serve requests and to react to failures and mobility.

3.10 Compositional Reasoning and Inference for Weak Isolation

Suresh Jagannathan (Purdue University – West Lafayette, US)

License 🅭 Creative Commons BY 3.0 Unported license
© Suresh Jagannathan

Joint work of Gowtham Kaki, Kartik Nagar; Mahsa Najafzadeh
Main reference Gowtham Kaki, Kartik Nagar, Mahsa Najafzadeh, Suresh Jagannathan: "Alone together: compositional reasoning and inference for weak isolation", PACMPL, Vol. 2(POPL), pp. 27:1–27:34, 2018.
URL http://dx.doi.org/10.1145/3158115

Serializability is a desirable correctness property that simplifies reasoning about concurrently executing transactions. But, on weakly consistent distributed stores, serializability cannot be achieved without sacrificing availability, an unpalatable trade-off for many applications. Consequently, applications typically choose to weaken the strong isolation guarantees afforded by serializability in favour of weaker, albeit more available, variants. In this talk, I'll present some recent work on a verification methodology for reasoning about weakly-isolated transactions, and an inference procedure that determines the weakest isolation level that can be ascribed to transactions without violating an application's high-level invariants. The key to effective inference is the observation that weakly-isolated transactions can be viewed as functional (monadic) computations over an abstract database state, allowing us to treat their operations as state transformers over the database. This interpretation enables automated verification using off-the-shelf SMT solvers.

References

1 Gowtham Kaki, Kartik Nagar, Mahsa Najafzadeh, Suresh Jagannathan, "Alone Together: Compositional Reasoning and Inference for Weak Isolation". PACMPL 2(POPL): 27:1-27:34 (2018).

3.11 Consistency Compromises at the Coalface

Brad King (Scality – Paris, FR)

License 🅭 Creative Commons BY 3.0 Unported license
© Brad King

Consistency is always desirable but comes at a cost. The path taken to find acceptable consistency compromises for a multi-petabyte scale storage platform will be discussed. The Scality storage platform uses an appealing shared-nothing architecture which has excellent scaling characteristics, but cannot reliably handle many workloads without some form of coordination to provide consistency guarantees. The basic architecture, the challenges faced, the tools chosen and ongoing work will be presented. The current platform has several different approaches used in combination including: flat group quorums, Paxos, Raft and Totem based protocols. The constraints of working in production environments with mission critical applications presents challenges in combining performance, correctness and reliability while continuing to evolve the technology will be considered. Among other challenges, sufficient testing of the possible degraded and partitioned situations can become an intractable problem.

References

1 Bradley King *Consistency Compromises at the Coalface*. Scality, 11 rue Tronchet, 75008 Paris France

3.12 Jepsen 9: A Fsyncing Feeling

Kyle Kingsbury (San Francisco, US)

License 🅭 Creative Commons BY 3.0 Unported license
© Kyle Kingsbury

Distributed systems often claim to save our data durably, to provide isolated transactions, to make writes visible to reads. Jepsen is a distributed systems testing harness, which applies property-based testing to databases to verify their correctness claims during common failure modes: network partitions, process crashes, and clock skew. In this talk, we discuss anomalies in Tendermint, Hazelcast, and Aerospike.

3.13 Data structures as queries: Expressing CRDTs using Datalog

Martin Kleppmann (University of Cambridge, GB)

License 🅭 Creative Commons BY 3.0 Unported license
© Martin Kleppmann

Joint work of Martin Kleppmann, Victor B. F. Gomes, Dominic P. Mulligan, Alastair R. Beresford

Currently there are two conventional formulations of CRDTs: state-based (where we prove that our merge function is commutative, associative, and idempotent) or operation-based (where we prove that the functions that apply operations to the local state are commutative). I propose a third formulation in which the CRDT is expressed as a query over a monotonically growing set of operations. The merge function for the set of operations is just the set union, which is trivially commutative, associative, and idempotent. By expressing the desired data structure as a deterministic query over that set we get convergence automatically. I will discuss how we can use the Datalog language to express such queries, how this query-based approach can help us better understand existing CRDTs, and how it facilitates designing new ones.

3.14 Staying in Sync: From Transactions to Streams

Martin Kleppmann (University of Cambridge, GB)

License 🅭 Creative Commons BY 3.0 Unported license
© Martin Kleppmann
Main reference Martin Kleppmann: "Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems", O'Reilly, 2016.
URL http://shop.oreilly.com/product/0636920032175.do

For the very simplest applications, a single database is sufficient, and then life is pretty good. But as your application needs to do more, you often find that no single technology can do everything you need to do with your data. And so you end up having to combine several databases, caches, search indexes, message queues, analytics tools, machine learning systems, and so on, into a heterogeneous infrastructure…

Now you have a new problem: your data is stored in several different places, and if it changes in one place, you have to keep it in sync in the other places, too. It's not too bad if all your systems are up and running smoothly, but what if some parts of your systems have failed, some are running slow, and some are running buggy code that was deployed by accident?

It's not easy to keep data in sync across different systems in the face of failure. Distributed transactions and 2-phase commit have long been seen as the "correct" solution, but they are slow and have operational problems, and so many systems can't afford to use them.

In this talk we'll explore using event streams and Kafka for keeping data in sync across heterogeneous systems, and compare this approach to distributed transactions: what consistency guarantees can it offer, and how does it fare in the face of failure?

3.15 Homomorphic Computation for Distributed Computing

Christopher Meiklejohn (UC Louvain, BE)

License 🅭 Creative Commons BY 3.0 Unported license
© Christopher Meiklejohn

State-of-the-art programming models for building coordination-free distributed applications typically rely on a combination of lattice-based programming with monotonic application logic. As the CALM result has demonstrated, these programs guarantee convergence in the face of various network anomalies such as message reordering and message duplication. However, two of the systems that represent the state-of-the-art, Lasp [1] and Bloom$\hat{L}$ [2], each place the onus on the developer of a.) modeling their application state as join-semilattices, and b.) ensuring that computations in application code are monotone. Furthermore, these programming models can take advantage of homomorphisms, a special case of monotone programming where function application distributes over the join, to provide incremental computing: key to applications that are geographically distributed.

In this talk, we present a work-in-progress result on writing correct monotone programs with join-semilattices. This framework generalizes the reprensentation for lattice-based data types, provides a type system approach to ensuring monotonicity, and provides a mechanism for automatically lifting monotone functions to homomorphic functions between lattices. We present a operational semantics for an incremental evaluation model, that generalizes the execution models of both Lasp and Bloom^L and demonstrate how the existing systems fit into our framework.

References

1 Meiklejohn, C., and Van Roy, P. Lasp: A language for distributed, coordination-free programming. In *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming* (2015), ACM, pp. 184–195.

2 Logic and lattices for distributed programming. In *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM.

3.16 Massive Collaboratie Editing

Pascal Molli (University of Nantes, FR)

License 🅭 Creative Commons BY 3.0 Unported license
© Pascal Molli

Distributed real-time editors made real-time editing easy for millions of users. However, main stream editors rely on Cloud services to mediate sessions raising privacy and scalability issues. Decentralized editors tackle privacy issues, but scalability issues remain. We aim to build a decentralized editor that allows real-time editing anytime, anywhere, whatever is the number of participants. In this study, we propose an approach based on a massively replicated sequence data structure that represents the shared document. We establish an original trade-off on communication, time, and space complexity to maintain this sequence over a network of browsers. We prove a sublinear upper bound on communication complexity while preserving an affordable time and space complexity. To validate this trade-off, we built a full working editor and measured its performance on large-scale experiments involving up till 600 participants. As expected, the results show a traffic increasing as $O((logI)^2 lnR)$ where I is the number of insertions in the document, and R the number of participants.

3.17 External Consistency in Partial Replication without TrueTime API

Roberto Palmieri (Lehigh University – Bethlehem, US), Masoomeh Javidi Kishi, and Sebastiano Peluso

License 🅭 Creative Commons BY 3.0 Unported license
© Roberto Palmieri, Masoomeh Javidi Kishi, and Sebastiano Peluso

This paper speaks about challenges of guaranteeing external consistency in a partially replicated system without any centralized synchronization component and where read-only transactions are never abort. Google Spanner establishes external consistency by leveraging the TrueTime API; in this work we replace it with a combination of vector and scalar clocks to achieve similar guarantees.

In our system, which we name SSS, write transactions commit by leveraging two-phase commit. Read-only transactions implement non-blocking execution by leaving a trace of their execution on accessed replicas so that write transactions can detect the presence of a write-after-read conflict, which forces write transaction to hold their response to client until the read-only is completed. Internally in the system, although write transaction waits for read-only transactions, their written values are already exposed to other concurrent transactions, therefore system throughout is not affected by the above delay.

Interestingly, read-only transactions notify concurrent and conflicting write transactions upon completion so that they can proceed providing the response (put on hold previously) to clients. This notification sent by read-only transactions also serves as garbage collection message to discard any left trace by a read-only transaction in the system.

3.18 Programming Scalable Cloud Services

Gustavo Petri (University Paris-Diderot, FR), Patrick Eugster, Srivatsan Ravi, Masoud Saeida Ardekani (Samsung Research – Mountain View, US), and Bo Sang

License 🅭 Creative Commons BY 3.0 Unported license
© Gustavo Petri, Patrick Eugster, Srivatsan Ravi, Masoud Saeida Ardekani, and Bo Sang

In this talk we will introduce a programming model for elastic cloud applications based on actors. Our model leverages a native notion of ownership to structure the actors at runtime. By means of this ownership, we can deliver atomic cross-actor transactions, while retaining scalability and elasticity. After presenting the programming model, we will conclude with open problems and some future directions.

3.19 Enforcing SQL constrains in Weakly Consistent Databases

Nuno Preguica (New University of Lisbon, PT)

License 🅭 Creative Commons BY 3.0 Unported license
© Nuno Preguica

Joint work of João Sousa, Valter Balegas, Sérgio Duarte, Carla Ferreira, Rodrigo Rodrigues, Subhajit Sidhanta

Weak consistency is popular in the design of geo-replicated databases. When compared with strong consistency, this approach has the advantage of allowing low latency and high availability, as operations can execute in any replica without the need to coordinate with other replicas. For working correctly, some applications need to enforce application-specific constraints, which is challenging in weakly consistent databases. In this talk, we discuss to which extent it is possible to enforce SQL constraints in such settings.

3.20 Isolation Level Analysis

Sebastian Schweizer (TU Braunschweig, DE), Annette Bieniusa (TU Kaiserslautern, DE), Keijo Heljanko, Roland Meyer, and Arnd Poetzsch-Heffter

License 🅭 Creative Commons BY 3.0 Unported license
© Sebastian Schweizer, Annette Bieniusa, Keijo Heljanko, Roland Meyer, and Arnd Poetzsch-Heffter

Modern database systems offer different isolation levels. The isolation level defines what synchronization guarantees a programmer can rely on. The choice is a trade-off between performance (weak isolation) and strong guarantees (strong isolation).

Isolation Level Analysis is an approach to compare the behavior of a database program in different isolation levels. It allows to automatically find the isolation level that is best for a specific application, i.e. it is strong enough to avoid synchronization issues but weak enough to provide good throughput. Our technique takes as input a database program and two isolation levels. It then checks whether there is an execution that is possible in the weak but not in the strong isolation level. If no such execution is found, then the database operator can safely switch to the weaker level without adding additional behavior.

3.21 Just-Right Consistency: As available as possible, consistent when necessary, correct by design

Marc Shapiro (Sorbonne-Université – LIP6 & Inria – Paris, FR), Annette Bieniusa (TU Kaiserslautern, DE), Christopher Meiklejohn (UC Louvain, BE), Nuno Preguica (New University of Lisbon, PT), and Valter Balegas

License 🅭 Creative Commons BY 3.0 Unported license
© Marc Shapiro, Annette Bieniusa, Christopher Meiklejohn, Nuno Preguica, and Valter Balegas

Main reference Marc Shapiro, Annette Bieniusa, Nuno M. Preguiça, Valter Balegas, Christopher Meiklejohn: "Just-Right Consistency: reconciling availability and safety", CoRR, Vol. abs/1801.06340, 2018.
URL http://arxiv.org/abs/1801.06340

In a distributed data store, the CAP theorem forces a choice between strong consistency (CP) and availability and responsiveness (AP) when the network can partition. To address this issue, we take an application-driven approach, Just-Right Consistency (JRC). JRC defines a consistency model that is sufficient to maintain the application invariants, and otherwise remaining as available as possible.

JRC leverages knowledge of the application. Two invariant-maintaining patterns, ordered updates and atomic grouping, are compatible with concurrent and asynchronous updates, orthogonally to CAP. In contrast, checking a data precondition on partitioned state is CAP-sensitive. However, if two updates do not negate each other's precondition, they may legally execute concurrently. Updates must synchronise only if one negates the precondition of the other.

The JRC approach is supported: by the CRDT data model that ensures that concurrent updates converge; by Antidote, a cloud-scale CRDT data store that guarantees transactional causal consistency; and by developer tools (static analysers and domain-specific languages) that help guarantee invariants. This research is supported in part by FP7 SyncFree, H2020 LightKone, and by ANR project RainbowFS.

3.22 Fast State-Machine Replication via Monotonic Generic Broadcast

Pierre Sutra (Télécom SudParis – Évry, FR)

License Creative Commons BY 3.0 Unported license
© Pierre Sutra
Joint work of Vitor Enes, Tuanir França Rezende, Alexey Gotsman, Matthieu Perrin, Pierre Sutra

This talk introduces Monotonic Generic Broadcast (MG-broadcast), a new group communication primitive enabling efficient state-machine replication. Like generic broadcast, MG-broadcast does not require ordering commutative state-machine commands. In addition, it allows a replicated state machine to serve reads from a local replica while preserving sequential consistency.

We present a protocol implementing MG-broadcast that is leaderless: commands do not have to be ordered by a single leader node, which results in better scalability and availability. Furthermore, the latency of our protocol is optimal when one failure may occur at a time, in which case an update command contacts a simple majority of processes and always completes in one round trip. This makes our protocol especially appropriate for geo-distribution.

We close this talk by presenting several empirical results that evaluate MG-broadcast in a geo-distributed setting, using 3 to 11 geographical locations. We show that, under a range of workloads, our protocol outperforms prior replicated state machine solutions.

3.23 Robust (Parallel) Snapshot Isolation

Viktor Vafeiadis (MPI-SWS – Kaiserslautern, DE)

License Creative Commons BY 3.0 Unported license
© Viktor Vafeiadis
Joint work of Azalea Raad, Ori Lahav, Viktor Vafeiadis
Main reference Azalea Raad, Ori Lahav, Viktor Vafeiadis: "On Parallel Snapshot Isolation and Release/Acquire Consistency", in Proc. of the Programming Languages and Systems – 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Lecture Notes in Computer Science, Vol. 10801, pp. 940–967, Springer, 2018.
URL http://dx.doi.org/10.1007/978-3-319-89884-1_33

Snapshot isolation (SI) and *parallel snapshot isolation* (PSI) are two standard transactional consistency models that is used in databases and distributed systems. Since they provide much better performance than serializability, it makes sense to adopt them as a transactional models for STMs in programming languages.

In the programming language setting, however, one must crucially allow the interaction of transactional and non-transactional code. In a recent paper [1], we constructed RPSI, a robust version of PSI that is better suited for the setting. We have built a simple lock-based reference implementation of RPSI over the release-acquire fragment of the C/C++ concurrency model [2], and have proved that our implementation can exhibit exactly the same behaviour as allowed by RPSI's declarative specification.

In ongoing work, we are looking to achieve a similar result for SI.

References

1 Azalea Raad, Ori Lahav, and Viktor Vafeiadis. On parallel snapshot isolation and release/acquire consistency. In ESOP 2018: 27th European Symposium on Programming, Springer, 2018.

2 Ori Lahav, Nick Giannarakis, and Viktor Vafeiadis. Taming release-acquire consistency. In POPL 2016: 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 649-662, ACM, 2016.

3.24 Elements of a unified semantics for synchronization-free programming based on Lasp and Antidote

Peter Van Roy (UC Louvain, BE)

License 🅭 Creative Commons BY 3.0 Unported license
© Peter Van Roy

Joint work of Peter Zeller, Annette Bieniusa, Mathias Weber, Christopher Meiklejohn, Peter Van Roy, Nuno Preguiça, Carla Ferreira
Main reference LightKone: Lightweight Computations at the Edge. H2020 Project, Jan. 2017 – Dec. 2019, see lightkone.eu.
URL https://www.lightkone.eu

We outline a unified semantics for Lasp and Antidote based on Burckhardt's abstract execution formalism. This semantics is the foundation for an edge computing platform that we are building in the LightKone H2020 project. The platform provides a functional programming style with an efficient implementation on large dynamic networks with unreliable nodes and communication. Lasp and Antidote are both synchronization-free programming systems, i.e., they are based on CRDTs, which are distributed data structures that provide consistency between replicas using a weak synchronization model, namely eventual replica-to-replica communication. Lasp and Antidote are situated in two very different areas of the synchronization-free design space. Lasp is a distributed dataflow system implemented on large dynamic networks. Antidote is a causally consistent transactional database implemented on georeplicated data centers. The unified semantics brings together two communities and will help us make progress in synchronization-free programming.

4 Working groups

4.1 "Theory and Practice" working group report

Carlos Baquero (University of Minho – Braga, PT) and Carla Ferreira (New University of Lisbon, PT)

License 🅭 Creative Commons BY 3.0 Unported license
© Carlos Baquero and Carla Ferreira

On February 27 we split into two groups to discuss the interplay among theory and practice in distributed systems. Our group had twelve participants and the initial discussion was sparked by examples on highly available editable sequences, were there is a contrast between known lower bounds on metadata size and, in contrast, the practical need to continue research on efficient solutions for average case scenarios. We looked for more examples that reflected this tension, and discussed how the FLP results lead, for a while, to a decrease in research on asynchronous consensus algorithms, until Paxos finally re-surfaced. The discussion then evolved into how difficult it can be for practitioners to navigate the spectrum of system models and algorithms for solving specific tasks. Finally we came up with the following two take home messages, resulting from the discussion:

- Lower bounds and impossibility results are important navigation tools, but if taken too generally they can limit research on contexts that might seem covered by those results. E.g. multi-master sequence editing, FLP vs Paxos algorithms, vector clocks size lower bounds and scalable causality research.
- Consistency models have a complex taxonomy, it is hard to expect users to navigate that correctly. A flowchart or guided navigation, a wiki taxonomy, could help users choose the best tools/algorithms for the aimed setting. The end effect of uncharted complexity can lead practitioners to chose stronger consistency than needed or move to the other end of the spectrum and choose basic key-value stores.

4.2 Theory vs Practice: are we developing the right models or systems?

Khuzaima Daudjee (University of Waterloo, CA)

License 🅭 Creative Commons BY 3.0 Unported license
© Khuzaima Daudjee

The group of participants ranged from researchers working on aspects related to verification and formalizing notions of consistency to managing data to building industrial systems.

Several questions/issues were raised including how is the theory relevant to people developing systems. Some views that were shared:

- systems people need to build systems that work
- practitioners are quite interested in understanding how things work and correctness of protocols; in fact there is a demand for these

Researchers expressed concern about what stops theoretical notions from working that included:

- strong assumptions
- relevant papers are badly written
- terminology is inconsistent
- protocols are not efficient when implemented
- need to mathematically verify guarantees that papers propose

There was concern over jargon and imprecision in the use of technical terms from researchers from different communities. Some expressed hope that "equivalence classes" will develop and terms will solidify and converge. Often, programmers do not understand the differences between the different terms and the consistency levels they offer.

It can really help when someone implements a theory to make its use and understanding concrete. Oracle delivered SI as serializable isolation; this is a good example of how practice accelerated acceptance into theory. Well-specified protocols help gain acceptance, e.g., Lamport's Paxos.

It takes time for adoption of ideas and concepts into tools. Maybe what we need are a few consistency models that are used as gold standards.

4.3 Where Do We Go Next

Kyle Kingsbury (San Francisco, US)

License 🅭 Creative Commons BY 3.0 Unported license
© Kyle Kingsbury

We discussed the future of distributed consistency–what research directions appear most promising? What has failed, or gone under-explored? And more importantly, where do we *need* to be?

We start by recalling that the *last* Next Big Thing was going to be dataflow. This did not quite come to pass–we believe, in part, because our field has abandoned monolithic approaches in favor of heterogenous systems. However, we remain *unhappy* with system building. The systems approach has come to dominate industry, but the compositional *rules* for those system components are poorly understood.

In addition, layering has sometimes proven *weaker* than integrated wholes. TAPIR, for instance, suggests that we can build faster transactional systems by giving *up* ordering at lower layers–in exchange for a more complex transaction system on top. That said, it's nice to know what the layers *are* before we start breaking them.

We suffer from a lack of uniform abstractions for consistency problems, both in describing safety models, but also the *performance* of systems, and their compositional rules. We would like to see a language with which one could describe a given system's abstract behavior, and prove how composing it with a second system would provide, or fail to provide, important invariants such as isolation and fault tolerance. This specification language would need to be usable by (at least some) engineers.

What would be the semantics of the interfaces between systems, in such a language?

Perhaps a focus on request-response patterns and their relationships would be useful? This cannot be the whole story, because many problems, like streaming or materialized view maintenance, cannot be represented easily in terms of RPC. Perhaps (temporally qualified) relations between system states might prove useful. A cache, for instance, should reflect, at some time, a state of the database it draws from. Perhaps a process algebra would be more useful?

Moreover, we don't just want to compose. We want to impose *restrictions*, like access control. We want to *hide* things behind abstraction boundaries. Can a language encode these things?

Given the success of weakly safe systems glued together from heterogenous components, we suspect that building "cathedral-style" languages or databases, mean to encompass everything programmers might need to do in a distributed system, is likely doomed; there are a host of technical and social reasons that drive adoption, and monolithic designs are resistant to change and difficult to adapt to new contexts. Perhaps what we need are *models* or *patterns* for building distributed computation: MapReduce, for example, has been a successful model implemented in several ways across industry.

Another such model which has *not* been broadly adopted might be metadata for causality tracking. Vector clocks, causal tokens, idempotence tokens, request IDs in distributed tracing like Zipkin, and causally consistent timestamps: these seem like patterns that could be formalized and shared between components. It would be nice if, say, a Riak vector clock could be passed as a causal token into some other database to ensure a query includes data reflective of that vclock.

There are, of course, lots of ways to implement causality. We could use explicit vector clocks vs implicit session or global orders. We could leave this unspecified, or offer extension points. As new types of causal tokens are identified, the language should grow to accommodate them.

Some may claim that we have never reaped the promised benefits of model or standards re-use. However, some successful ideas *have* proven remarkably successful. Libraries and programming languages are widely re-used. Unix pipes and the concept of files remain ubiquitous. Proxies, caches, and load balancers are well-understood patterns now, and all cooperate beautifully in the case of HTTP. HTTP (and REST) itself has proven successful through its use of standardized *and* extensible headers, providing a language for heterogenous components to cooperate. Perhaps we can learn from their example.

Participants

- Peter Alvaro
University of California – Santa Cruz, US
- Mahesh Balakrishnan
Yale University – New Haven, US
- Carlos Baquero
University of Minho – Braga, PT
- Annette Bieniusa
TU Kaiserslautern, DE
- Ahmed Bouajjani
University Paris-Diderot, FR
- Manuel Bravo
INESC-ID – Lisbon, PT
- Sebastian Burckhardt
Microsoft Research – Redmond, US
- Andrea Cerone
Imperial College London, GB
- Gregory Chockler
Royal Holloway, University of London, GB
- Khuzaima Daudjee
University of Waterloo, CA
- Diego Didona
EPFL – Lausanne, CH
- Amr El Abbadi
University of California – Santa Barbara, US
- Carla Ferreira
New University of Lisbon, PT
- Alexey Gotsman
IMDEA Software – Madrid, ES
- Suresh Jagannathan
Purdue University – West Lafayette, US
- Bettina Kemme
McGill University – Montreal, CA
- Brad King
Scality – Paris, FR
- Kyle Kingsbury
San Francisco, US
- Martin Kleppmann
University of Cambridge, GB
- Christopher Meiklejohn
UC Louvain, BE
- Roland Meyer
TU Braunschweig, DE
- Maged M. Michael
Facebook – New York, US
- Pascal Molli
University of Nantes, FR
- Roberto Palmieri
Lehigh University – Bethlehem, US
- Matthieu Perrin
University of Nantes, FR
- Gustavo Petri
University Paris-Diderot, FR
- Nuno Preguica
New University of Lisbon, PT
- Luis Rodrigues
INESC-ID – Lisbon, PT
- Rodrigo Rodrigues
INESC-ID – Lisbon, PT
- Masoud Saeida Ardekani
Samsung Research – Mountain View, US
- Sebastian Schweizer
TU Braunschweig, DE
- Marc Shapiro
University Pierre & Marie Curie – Paris, FR
- Pierre Sutra
Télécom SudParis – Évry, FR
- Viktor Vafeiadis
MPI-SWS – Kaiserslautern, DE
- Peter Van Roy
UC Louvain, BE