Research Statement

Gowtham Kaki

Computing has come a long way since Von Neumann introduced his stored-program model of a computer in 1945. Von Neumann's architecture is fundamentally *compute-centric* – a Central Processing Unit (CPU) performs all the computation by periodically fetching data and programs from a passive main memory or a secondary storage. Implicit in this model is the assumption that it is possible to tread a computation's data in its entirety through a "central" processor to a compute a single value that is the canonical result of a computation. This view of computing stands in stark contrast to the reality of modern applications, which are fundamentally *data-centric* as opposed to compute-centric – they operate on large amounts of data organized into multiple data banks, e.g., *databases*, *shards*, and *replicas*, and perform computations that are essentially distributed in nature to compute results for which no canonical answer exists. Despite such fundamental differences in the nature of computing, applications today are nonetheless programmed using the same language abstractions and programming techniques that trace their origins to the Von Neumann model. The resultant *impedance mismatch* between the computation performed and the programming model employed has been a consistent source of negative impact on the security, reliability, and performance of data-centric applications. For instance, coercing database transactions, which are concurrent by their very nature, into Von Neumann model of computing, which emphasizes an orderly execution of instructions, has resulted in an impedance mismatch that effectively exposed several database-backed commercial applications to concurrency-related attacks [11]. One such attack has been particularly notorious for it has led to the theft of over half-a-million dollars, and subsequent closure of an otherwise well-performing Bitcoin exchange named Flexcoin [1]. More such examples abound in literature and in popular press.

As evident from above, there is a need for novel computational models, language abstractions and programming techniques that better suit the nature of modern data-centric applications. Developing such programming infrastructure is indeed one of the focal points of my research, in particular my most recent work on distributed programming models and replicated data structures [8, 9, 3]. The more pressing need however is for automated reasoning techniques, program analyses, and compilation tools that work with the current generation of data-centric applications, helping them meet the ever so strict goals of scalability and availability *without* compromising security and robustness. Developing such automated reasoning infrastructure is another focal point of my research, and a unifying theme behind most of my work [6, 7, 5, 4, 10]. **At a high level, my research falls at the intersection of programming languages, software engineering, databases, and distributed systems. In particular, I aim to solve the most pressing problems in the latter two domains by developing novel approaches that build on top of the conventional wisdom and technical know-how of the former two**. The relevance of my work to the theory and practice of contemporary data-centric computing has been recognized through a generous research fellowship from Google. In what follows, I describe my research at a high-level categorizing it into three broad themes, and conclude by briefly describing my ideas for future research.

Theoretical Foundations of Data Storage and Processing Systems

The theoretical and mental models of data storage systems prevalent among the programming community are often too simple to match the reality of such systems. For instance, developers commonly assume relational databases, such as MySQL and PostgreSQL, to be ACID-compliant with Serializable transactions, whereas in reality they are not [2]. To accentuate performance, database systems routinely drop Serializability in favor of weaker *isolation* guarantees that do *not* mask the concurrent nature of database transactions. This mismatch between expectation and reality results in subtle bugs that manifest as serious violations of safety [1, 11]. The problem is aggravated in case of distributed "NoSQL" data stores, which, by construction, admit *weakly-consistent* executions that defy the commonsense of programming. If programmers were to effectively navigate the complex reality of modern data storage systems, novel reasoning techniques and automated tool support are essential. Such tools and techniques should necessarily be based on sound proof theory and accurate formal models of modern data storage systems, which are among the major contribution of my research. In [6], I present the first ever high-level formal semantics of *weak isolation* as it occurs in the real world. Based on this semantics, I develop a deductive proof system that admits rigorous proofs of correctness for database-backed applications composed of weakly-isolated transactions. In a follow-up to this work [7], I extend the formal treatment to distributed "NoSQL" data stores and their applications composed of weakly-consistent operations. A distinguishing feature of both the formalisms is their remarkable simplicity notwithstanding their comprehensive treatment of complex real-world data systems. Such simplicity is indeed what makes the formal reasoning to be automated and applied at scale to identify concurrency-related vulnerabilities and verify correctness of realistic (distributed) database applications, as shall be explained in the next subsection.

My work in theoretical foundations extends to data processing systems as well. In [5], I demonstrate how stream processing systems (Microsoft's Naiad in particular) can take advantage of GC-free memory regions to significantly increase overall throughput of the system without compromising memory safety. The region type system I present in the paper, albeit grounded in sound theory, is an exercise in pragmatism for it strikes a fine balance between static type checking and lightweight run-time checks to ensure that the type system does not "get in the way" of Naiad's expert developers. Such productive collaborations with systems and database researchers is, I believe, how my research can have the most impact going forward.

Automatic Verification and Synthesis for Data-Centric Applications

My theoretical work in [6, 7, 5] gave me an excellent framework within which I could develop novel techniques in automated verification, modelchecking, type inference, and program synthesis for data-centric applications. Building on the proof system in [6], I built a tool called ACIDIFIER that can automatically analyze MySQL and PostgreSQL applications with weakly-isolated transactions to either detect concurrency-related vulnerabilities or certify them correct. ACIDIFIER can also infer the weakest (hence most scalable) isolation configurations at which it is safe to run these applications. For instance, for the TPC-C OLTP benchmark, ACIDIFIER discovered that none of the transactions need to be run at Serializable isolation in order to guarantee the correct behavior, which turned out to be accurate. Another tool I built, called Q9 is based on a novel operational semantics of weakly-consistent replicated data stores I present in [7], and extends automated (bounded) verification to distributed "NoSQL" applications. Q9 analyzes distributed applications under a finite

concurrency bound by reducing the application logic and data store semantics into set-theoretic constraints that can be very efficiently solved using off-the-shelf SMT solvers. This approach helped Q9 discover a range of weak-consistency bugs in large-scale applications, including the distributed variants of TPC-C and TPC-E OLTP benchmarks in just under two minutes, and subsequently repair the applications through selective consistency strengthening.

Beyond verification, my work also spans synthesis of replicated data types (RDTs), such as replicated queues and document trees, which are the building blocks of an emerging class of peer-to-peer and interactive distributed applications, including blockchains and collaborative editors. The theoretical foundations were laid in [4], where I show that any functional data structure can be reduced to a collection of *structural relations* for which automatic reasoning is decidable. In [4], I built a dependent type system called Catalyst that performs the aforementioned relational reasoning to automatically verify the correctness of data structure transformations, such as AST transformations in a compiler. The ideas were further developed in [8] to automatically *synthesize* data structure transformations, in particular a *merge* transformation that merges concurrent versions of a given data type in the context of their common ancestor. This crucial result made it possible to automatically *derive* replicated data types from first principles, thus providing a simpler and more principled alternative to the currently-popular variants of replicated data types that insist on commutativity of operations (i.e., CRDTs). The synthesis approach was used in practice to automatically derive a standard library of *mergeable* replicated data types (MRDTs) from ordinary OCaml data types, which were then used to build decentralized (peer-to-peer) variants of large OLTP benchmarks [8].

Novel Programming Models for Data-Centric Computation

Another focal point of my research work is the development of novel language abstractions and programming models that better suit the nature of data-centric computing. I made two major contributions towards this end. In [10], my collaborators and I present a Haskell domain-specific language (DSL) called Quelea that liberates application development on weakly-consistent "NoSQL" stores from the Von Neumann style. Quelea starts with a clean-slate design where, unlike the Von Neumann model, *no* order is guaranteed between any two operations, i.e., no consistency is assumed whatsoever. Instead, applications are required to *declare* their consistency requirements as a series of high-level ordering constraints on their operations, which will then be then be compiled down to efficient low-level code that handles systems and networking concerns (analogous to how declarative Haskell programs are compiled to efficient low-level imperative code by GHC). As we demonstrate in [10], Quelea drastically reduces the effort required to build highly-scalable distributed applications on top of off-the-shelf NoSQL data stores, e.g., Cassandra.

The second contribution is an OCaml DSL, named Carmot, for building peer-to-peer decentralized applications [9, 8]. Key to Carmot is the observation that a distributed computation (on data) is not fundamentally different from the collaborative development (of source code), hence familiar version control protocols, e.g, Git, which have been successfully used to manage the latter, can be profitably employed in the context of the former. Indeed, Carmot lets programmers orchestrate a distributed computation very much like a Git workflow, namely by forking branches, committing concurrent versions, and periodically merging versions to keep the branches in sync. As we demonstrate in [9, 8], the novel version control-inspired programming model of Carmot is indispensable in bringing the benefits of Mergeable Replicated Data Types (MRDTs), in particular

automatic synthesis, to fully-decentralized data-centric applications, taking us one step closer to the goal of automatically deriving such applications from their sequential counterparts.

Future Work

I see many opportunities for future research in data-centric computing, which I would like to pursue in collaboration with researchers in Systems, Databases, Cryptography, and Machine Learning. I sketch a few possibilities below.

Formal models of complex software systems In [6] and [7], I demonstrate the utility of employing simple formal models of (distributed) database systems in lieu of their complex implementations to reason about application safety. I see this approach being useful in the context of file systems, lock-free data structures, data processing libraries – basically any real-world software system that accrues code and optimizations from decades of engineering effort, and becomes too complex to be analyzed directly by the verification tools tasked with establishing the safety of their clients. Building high-fidelity formal models of such systems, however, requires considerable domain knowledge, which is often scattered through API and informal documentation, unit tests, configuration files, logs, and commit messages. Mining structured knowledge from these sources, and (mostly) automating the task of synthesizing formal models that act as proxies for these systems during verification is a challenging-yet-worthwhile exercise, and one that I plan to take up in near future.

Programming models for fully-decentralized and privacy-conscious computing Users of internet services are becoming increasingly vocal about their (justifiable) privacy concerns on the internet. Ambitious initiatives have sought to address this issue by promoting full decentralization in application and network protocol design (e.g., Blockchains and IPFS), and by putting people in charge of their data in their own premises (e.g., TRVE Data). Well-intended as they may be, such initiatives nonetheless have systemic issues for there are currently no languages and tools that can bridge the wide chasm between the limited human cognition of the programmer, and planet-scale workings of the underlying system. In [9], I show that a distributed programming model inspired by version control workflow can significantly reduce the cognitive burden involved in building highly decentralized applications. However, the questions about performance (at internet-scale) and privacy are yet unanswered, and addressing them may require a significant re-think of how computation should be performed over the internet. I would like to collaborate with Systems, Database, and Cryptography researchers towards answering this question.

Distributed vector representations of data-centric computations Following the runaway success of "word2vec" in natural language processing, there have been attempts to build distributed vector representations of code snippets ("code2vec") with the aim of comparing and predicting the semantic properties of code. While the approach is promising, the success so far has been limited owing to the high semantic complexity of general-purpose computations in a Turing-complete language compared to words in a natural language. I believe more success can be had by restricting our attention to data-centric computations of which most are expressible in a Turing-*incomplete* language such as Datalog (A case in the point is SQL, which is a subset of Datalog. Another example is the TPC-C benchmark we used in evaluating ACIDIFIER [6], which was completely written in a Turing-incomplete OCaml DSL). I am therefore interested in evaluating the possibility of building distributed vector representations of data-centric computations expressed in (a variant of) Datalog – "Datalog2Vec", in collaboration with researchers in Databases, Machine Learning, and NLP.

References

[1] BTC Stolen from Poloniex, 2014. URL https://bitcointalk.org/index.php?topic=499580.

[2] Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. HAT, Not CAP: Towards Highly Available Transactions. In *Proceedings of the 14th USENIX Conference on Hot Topics in Operating Systems*, HotOS'13, pages 24–24, Berkeley, CA, USA, 2013. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2490483.2490507.

[3] Gowtham Kaki. *Automatic Reasoning Techniques for Non-Serializable Data-Intensive Applications*. PhD thesis, 8 2019. URL https://hammer.figshare.com/articles/Automatic_Reasoning_Techniques_for_Non-Serializable_Data-Intensive_Applications/8977562.

[4] Gowtham Kaki and Suresh Jagannathan. A Relational Framework for Higher-order Shape Analysis. In *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming*, ICFP '14, pages 311–324, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2873-9. doi: 10.1145/2628136.2628159. URL http://doi.acm.org/10.1145/2628136.2628159.

[5] Gowtham Kaki and G. Ramalingam. Safe Transferable Regions. In Todd Millstein, editor, *32nd European Conference on Object-Oriented Programming (ECOOP 2018)*, volume 109 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:31, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-079-8. doi: 10.4230/LIPIcs.ECOOP.2018.11. URL http://drops.dagstuhl.de/opus/volltexte/2018/9216.

[6] Gowtham Kaki, Kartik Nagar, Mahsa Najafzadeh, and Suresh Jagannathan. Alone together: Compositional reasoning and inference for weak isolation. *Proc. ACM Program. Lang.*, 2(POPL):27:1–27:34, December 2017. ISSN 2475-1421. doi: 10.1145/3158115. URL http://doi.acm.org/10.1145/3158115.

[7] Gowtham Kaki, Kapil Earanky, KC Sivaramakrishnan, and Suresh Jagannathan. Safe replication through bounded concurrency verification. *Proc. ACM Program. Lang.*, 2(OOPSLA):164:1–164:27, October 2018. ISSN 2475-1421. doi: 10.1145/3276534. URL http://doi.acm.org/10.1145/3276534.

[8] Gowtham Kaki, Swarn Priya, KC Sivaramakrishnan, and Suresh Jagannathan. Mergeable replicated data types. *Proc. ACM Program. Lang.*, (OOPSLA), October 2019.

[9] Gowtham Kaki, KC Sivaramakrishnan, and Suresh Jagannathan. Version Control Is for Your Data Too. In Benjamin S. Lerner, Rastislav Bodík, and Shriram Krishnamurthi, editors, *3rd Summit on Advances in Programming Languages (SNAPL 2019)*, volume 136 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:18, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-113-9. doi: 10.4230/LIPIcs.SNAPL.2019.8. URL http://drops.dagstuhl.de/opus/volltexte/2019/10551.

[10] KC Sivaramakrishnan, Gowtham Kaki, and Suresh Jagannathan. Declarative Programming over Eventually Consistent Data Stores. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2015, pages 413–424, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3468-6. doi: 10.1145/2737924.2737981. URL http://doi.acm.org/10.1145/2737924.2737981.

[11] Todd Warszawski and Peter Bailis. ACIDRain: Concurrency-Related Attacks on Database-Backed Web Applications. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 5–20, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4197-4. doi: 10.1145/3035918.3064037. URL http://doi.acm.org/10.1145/3035918.3064037.