My goal is to make modern software systems easier to build, more reliable, and more secure. To this end I pursue research in the fields of software engineering, distributed systems, and security.

My software engineering research focuses on software analysis and formal methods, with an emphasis on developing new tools that empower developers to harness the power of these techniques. I also design and prototype novel systems. This provides me with first-hand experience of the difficulties that systems developers face, which makes me more effective in my research to support them. In developing systems I am especially interested in developing new designs of cloud computing infrastructure and in building systems that provide users with better privacy and security guarantees.

My research has appeared in top Computer Science conferences and journals[1]. As a sample of my research output, since starting at UBC my research has appeared in ICSE (3), ASE (3), ESEC/FSE (1), ESEM(1), NSDI (1), ANCS (1), TPDS (1), CHI (1), and IJCAI (1). Our paper from ESEM 2015 received the Best Full Paper Award. According to Google scholar, six of my most impactful papers have been cited over 100 times each.

Although my research spans several sub-fields of Computer Science, **my primary focus area is software engineering (SE)**. Most of my published work appears in top SE venues and most of my academic service is in support of the SE community.

Besides publications, my research is also characterized by releases of open source software that anyone can build on. For example, my specification mining tools[2] are frequently extended by other researchers and my log analysis tools[3] are actively used by practitioners.

Because of my broad research focus, I actively seek out collaborators and frequently work with colleagues in the UBC CS and ECE departments. For example, with both local and remote collaborators I have co-supervised 3 MSc and 1 postdoc, and am currently co-supervising 3 PhD students.

I also actively pursue collaborations with industry. I have worked with IBM, Shopify, Tasktop, and multiple research and product groups inside of Huawei. Each of these companies contributed funding to support my work. These industrial collaborations also enabled our group to deploy our prototypes in the wild, gain access to professional developers and access unique industrial datasets (e.g., data center topologies).

Since joining UBC I have supervised or am currently supervising 32 undergraduates, 17 MSc students, 6 PhD students, and 1 postdoctoral researcher. Five students have received awards for research under my supervision. Graduates from my group have gone on to top positions in industry (Google, Facebook, Arista Networks), PhD programs (Berkeley, UCSD, U. Waterloo, McGill), and the postdoctoral researcher is now an Assistant Professor at the University of Colorado, Colorado Springs. My research agenda is informed by the interests of the students in my group: I encourage students to pursue projects that they find motivating and that leverage my expertise. For example, our recent work on securing distributed machine learning systems is a student-led initiative.

My broad research program connects me with several different research communities, including systems and software engineering. I contribute in service roles to as many of these communities as I can. For example, I have served on top conference program committees, such as ICSE, ICDCS, ISSRE, and ESEM. I have also taken on organization roles. I was a general co-chair of a top workshop on operating systems (HotOS 2017). And, I have served as a program committee co-chair and now a steering committee member of VISSOFT (a working conference bridging software engineering and visualization communities). Most recently I am serving as a student volunteers co-chair for ICSE 2019.

In the rest of the document I will discuss my contributions to my three sub-fields of research since joining UBC. I will start with my **software engineering** research, in which I focus on developing new program analyses and carrying out empirical studies. I will then discuss my **distributed systems** research on improving the construction of systems and on developing new abstractions. Finally, I will overview my research focused on improving the **privacy and security** of software systems.

[1] The most read and cited Computer Science papers in my research areas appear in prestigious conferences with low acceptance rates.
[2] https://github.com/ModelInference
[3] https://bestchai.bitbucket.io/shiviz/, https://github.com/DistributedClocks

Software engineering (SE) research

My SE research has two strands: program analysis and empirical studies. The goal of my program analysis research is to support developers in their work, usually by embedding advanced program analyses into tools that they can use. The goal of my empirical studies is to precisely characterize the needs of and challenges facing software engineers. For example, in a recent study we considered developers' needs and processes in performance comprehension [ESEC/FSE 2018.P1].

Program analysis for specification mining

My expertise lies in an area of program analysis called *specification mining*, also known as *process mining*. My PhD was in this area, and I continue research in this domain. Many of the specification mining techniques that I have developed rely on dynamic program analyses, which are applicable to existing codebases of complex systems. Since specifications mined through dynamic program analysis are by nature incomplete, and may be inaccurate, they are therefore often referred to as *likely* specifications.

Specifications of software systems are critically important. Numerous program analyses, such as test generation, model checking, and runtime verification depend on their availability. However, program specifications are frequently missing. Specifications may be missing because developers lack the expertise to formally specify their systems, because the specification process is a manual and time-consuming process, or because specifications may quickly fall out of date as the software changes. This challenge inspired the field of *specification mining* – automatically generating specifications from software artifacts.

Since starting at UBC, my initial efforts were directed at publishing the last two pieces of my PhD work: inferring specifications of networked systems [ICSE 2014.P2], and generalizing the description of specification mining algorithms [TSE 2015.P3].

Subsequently, I have worked on a number of ways to generalize existing specification mining approaches to new data and new domains. In collaboration with Yuriy Brun (UMass Amherst), I have developed techniques for inferring performance models as specifications [ASE 2014.P4][4]. With two undergraduate students working under my supervision I have also generalized the mining of temporal patterns, the first technique to work for linear temporal logic properties of arbitrary complexity [ASE 2015.P5] (Figure 1)[5]. In collaboration with David Lo (Singapore Management University), I designed the first meta-algorithm for specification mining, which combines existing algorithms and yields higher performance than any of the underlying algorithms [ASE 2015.P6].

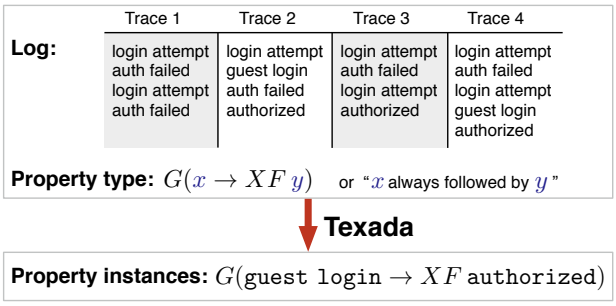

Figure 1: Example of specification mining with Texada [ASE 2015.P5], which takes a log, a property type, and outputs mined property instances (formulas based on the property type that evaluate to true on all traces in the input log).

In a more recent effort, I turned my focus to specifications of distributed systems, which frequently refer to the state of multiple nodes. For example, a statement of the form "$\forall$ node $i, groupLeader_i == groupLeader_j$" captures consistent global knowledge of the leader in a distributed system. I, my student Stewart Grant (MSc), and a visiting undergraduate student developed Dinv[6], the first tool to infer distributed data specifications from complex distributed systems [ICSE 2018.P7]. We have used Dinv to infer correctness invariants of the Raft consensus protocol in etcd, a widely adopted key-value store, and in other complex systems.

Ongoing work: data-temporal specification mining I am continuing to work with my students to generalize the mining of specifications to properties that can better capture human requirements. Specifically, we are attempting to unify the mining of data specifications with the mining of temporal logic specifications. For

[4] http://perfume.cs.umass.edu/
[5] https://bitbucket.org/bestchai/texada/
[6] https://bitbucket.org/bestchai/dinv/

example, our approach can detect properties like *"whenever numAuthTries == 3, eventually alertAdministrator() is invoked"*.

Studies and analyses of complex concurrent systems

In collaboration with Ali Mesbah (UBC ECE) and our (graduated) MASc student Keheliya Gallaba, we began a line of work on studying the use of callbacks in JavaScript applications. JavaScript is an event-based language that lacks concurrency abstractions. It makes up for this with events, that can be scheduled and reacted to with callbacks. Today, it is difficult to imagine a useful JavaScript application that does not contain callbacks – it is a core language feature. However, callbacks are complex to understand and to use: some refer to JavaScript's "callback hell"[7]. In many ways, callbacks are an obstacle to learning JavaScript and to writing correct programs in this language. We performed a large-scale empirical study to characterize the use of callbacks in popular JavaScript systems [ESEM 2015.P8]. This is the first study of its kind and it alerted the community to the complexity of callback usage in existing systems. For example, half of all callbacks are invoked asynchronously and callbacks are frequently deeply nested. This work received the best full paper award at ESEM 2015.

Building on this study we developed program analysis techniques to mitigate callback complexity. Specifically, we designed analyses to refactor JavaScript callbacks into a language construct called *promises*. A promise provides an explicit method for nesting callbacks, and a means to explicitly denote the error and the success paths of an event handler. We developed a static analysis approach to perform this refactoring. In a highly dynamic language like JavaScript, the major challenge is refactoring the code without changing the semantics of the program. Our analysis and corresponding tool [ICSME 2017.P9] refactors callbacks with many more advanced usage patterns and refactors 235% more callbacks than prior work.

In a more recent empirical case study with Alexandra Fedorova (UBC ECE), Julia Rubin (UBC ECE), and several of our students, we considered bug reports in the WiredTiger projects, the default database engine behind MongoDB, a popular key-value store. The focus of our study was *performance comprehension*: the process by which developers understand the underlying cause of a performance issue. This topic had not been studied prior to our work. Our study employs several qualitative techniques and the resulting paper [ESEC/FSE 2018.P1] reports salient observations. For example, WiredTiger developers often use tools without knowing in advance whether the obtained information will be relevant to their problem.

I am continuing this work by developing tools to support developers of complex concurrent systems like WiredTiger. For example, we recently built a tool, called TSViz[8], for visualizing thread activities in complex multi-threaded software [ICSE Tools 2017.P10].

Software visualization In designing tools I frequently draw on visualization techniques to build artifacts that are simple to understand and enjoyable to use. For example, my work with Gail Murphy and Daniel Rozenberg (MSc) on visualizing repository metrics over time to enable comparison/contrast of multiple repositories [MSR 2016.P11] builds heavily on visualization principles such as Shneiderman's *overview first, zoom and filter, then details-on-demand*. Another example is the TSViz tool mentioned above and the ShiViz tool described below.

Distributed systems research

We are increasingly relying on distributed systems for our everyday activities. For example, the rapid adoption of machine learning techniques is made possible by huge computational and storage resources. These resources are typically accessed in a data center with numerous distributed software systems coordinating the usage, accounting, and fault tolerance of these resources. My research aims to make distributed systems easier to construct and to explore new distributed system designs and abstractions to support emerging applications.

[7] http://callbackhell.com
[8] http://bestchai.bitbucket.io/tsviz/

Improving the construction of distributed systems

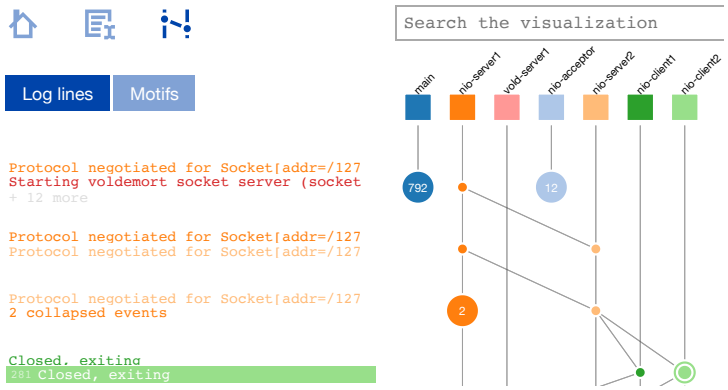

Figure 2: ShiViz interface for a log generated by the Voldemort key-value store. Time flows top to bottom, boxes denote thread timelines with circular events corresponding to log entries.

Distributed systems are notoriously challenging to build since they must overcome many concurrency, failure, and scalability issues. I am interested in developing new abstractions and tools to make distributed systems easier to design and implement. In collaboration with Michael Ernst (University of Washington), Yuriy Brun (UMass Amherst), and several undergraduate students under my supervision, I have developed ShiViz (Figure 2)[9], a tool to help developers understand their systems by visualizing traces of distributed system executions [CACM 2016.P12]. ShiViz helps developers to understand the sequence of interactions in the system, to search a trace for notable events or specific communication topologies, to compare several distributed executions, and to perform a variety of other actions on the traces. ShiViz is deployed and used by developers within Huawei. It is also being further extended by other researchers[10].

Users of ShiViz must instrument their systems to generate complete execution traces. In collaboration with Justin Cappos (NYU) and our co-supervised postdoc (Yanyan Zhuang, now an Assistant Professor at the University of Colorado, Colorado Springs), I worked on techniques to help debug distributed systems without instrumentation (e.g., Skype) [NSDI 2014.P13]. We developed an approach to reconstruct the sequence of events in a distributed system from totally ordered (local) traces of socket API calls from several machines. These calls can be collected using black-box tracing tools like `strace`. After collecting these socket traces our tool orders the events in the traces using the semantics of the networking API and uses the recovered information to classify and diagnose anomalous behavior.

With the above team I also helped to create Tsumiki, a meta-platform for quickly developing distributed testbeds to support CS education and research [TPDS 2018.P14]. Testbeds are critical to evaluating and deploying new distributed systems. However, no single testbed can satisfy the requirements of every project, prompting continual efforts to develop new testbeds. Tsumiki introduced a set of ready-to-use software components and interfaces to help system developers rapidly prototype custom networked testbeds without substantial effort. Tsumiki has been used in production as part of multiple testbeds, resulting in installations on tens of thousands of devices and use by thousands of researchers worldwide.

Ongoing work: compiling correct distributed systems There is a growing interest in specification languages for distributed systems, which can be checked exhaustively or proved to satisfy certain properties. For example, Amazon uses TLA+ and PlusCal in building its web services[11]. PlusCal is a formal specification language which has simple constructs for synchronization, nondeterminism, and specifying safety and liveness, which makes it an ideal choice to specify distributed algorithms that can then be model-checked. However, there is no tool to match a PlusCal specification with the implementation. Towards this end, I am working with three MSc and several undergraduate students on building a compiler that converts a PlusCal specification into Go language source code while preserving the semantics of the specification. When our work is complete, a verified PlusCal design will compile to a correct and efficient Go implementation that can be immediately deployed.

Ongoing work: efficient model checking of distributed systems A model checker (MC) is a program analysis tool that exhaustively tests a system. That is, a MC explores many program behaviors automatically, without the developer needing to compose any test scenarios. An abstract MC require developers to manually model the system. This abstract model is fast to check, but may lead to false positives/negatives. By contrast, a concrete MC executes the underlying system and is immune to false positives/negatives, but it

[9] http://bestchai.bitbucket.io/shiviz/
[10] For example, it is used within Microsoft for visualizing P and P# traces, https://github.com/p-org/TraceVisualizer
[11] http://lamport.azurewebsites.net/tla/amazon.html

is slow since it has to run the actual system and deal with state explosion. I am developing a *hybrid model checker* that combines abstract and concrete MCs into a single, efficient, pipeline. There are three key ideas in this project: (1) use a concrete MC to generate traces to bootstrap the abstract model (through model learning), (2) use abstract trace replay in the concrete system to identify abstract false positives, and (3) refine and improve the abstract model with concrete counter-examples from step (2). By doing the bulk of the checking with the efficient abstract checker, we hope to get the best of both worlds: precise and efficient model checking without requiring the developer to compose a model of their system.

New abstractions for data center systems

Today's data centers provide a rudimentary API for tenants who primarily allocate and deallocate containers and virtual machines (VMs) to access data center resources. New abstractions have been proposed, but they are challenging to provide at data-center scale. In my recent work with collaborators in formal methods and AI (Alan Hu and Holger Hoos) and our PhD students (Nodir Kodirov and Sam Bayless), I have pursued new abstractions that better align data center resources with application demands.

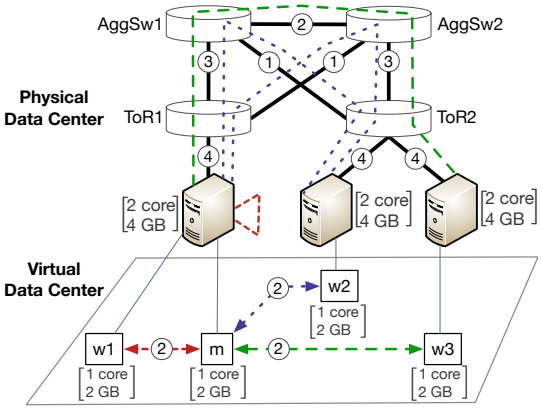

Figure 3: (Top) Example physical data center topology. Circled numbers on links denote available bandwidth in Gbps. (Bottom) Example Hadoop VDC with one master (m) and three worker VMs ($w1 - w3$).

VDC abstraction An alternative to allocating a single VM is to allocate a virtual data center (VDC): a collection of VMs with a network topology that specifies their interconnections (Figure 3). The VDC abstraction matches today's popular use-cases of big data processing. For example, a Hadoop job requires at least a master node and several worker nodes. A VDC can easily encode a Hadoop topology, which simplifies resource management for the tenant and also provides richer semantics to the data center operator (who knows that the VMs in a VDC belong to the same tenant, should be allocated close to one another, etc). Unlike the much simpler problem of allocating a single VM to the cloud, a VDC topology with several VMs must be allocated together, with guaranteed bandwidth and latency between some or all of the VMs. In our recent work we developed algorithms to support VDC allocation and placement in a physical data center topology [IJCAI 2017.P15]. Our algorithms are able to allocate 150% – 300% as many VDCs into the same physical data center as previous methods. We are currently working with Huawei Canada to deploy our approach in Huawei data centers.

Virtualized network functions chain abstraction Another data center trend is the outsourcing of middlebox functionality from enterprise networks into the data center. Outsourced (and traditionally hardware-based) middleboxes are today packaged into virtualized *network functions* (VNFs) that run in software. However, these VNFs, unlike VMs, require strict service level agreements (SLAs), particularly those pertaining to network bandwidth/latency/loss/jitter, and are typically connected in a chain topology.

VNF is another domain where the existing VM-based interface offered by data centers is insufficient. In recent work, with the same collaborators as the VDC project described above, I developed an abstract-concrete chain abstraction for VNFs [ANCS 2018.P16]. This work proposes an API, and also develops several data-center scale algorithms to implement this API. For example, our algorithms are able to achieve data center utilization within 96% of optimum, while scaling significantly past previously published work.

Ongoing work: abstractions for disaggregated data centers A new data center design trend is resource disaggregation: individual resource types like disk storage, RAM, and CPU are packaged into resource-specific units (blades) rather than into a server machine that aggregates these together. Disaggregated designs have better efficiency since the operator can upgrade specific hardware blades without impacting other resources. Users can provision the exact resources they need and expand/shrink this set at runtime. Both factors also improve data center utilization.

Disaggregation raises numerous questions about the abstraction that should be presented to tenants: should they still see a VM, or something else? The answer will influence the design of the entire software stack in the data center. Our recent work considered implications of disaggregation on fault tolerance [HotNets 2017.P17]. We are continuing by building systems that provide legacy applications with a transparent migration path

to disaggregated data centers. Our specific focus is on implementing local procedure calls in a distributed fashion. For this we are using efficient SDN-based distributed shared memory coupled with a fault tolerant and exactly-once remote procedure call abstraction.

Privacy and security research

My final line of work is in privacy and security. I use empirical methods to understand emerging threats and design novel systems to improve users' privacy and security.

Investigating the social insider threat

With collaborators in HCI (Tiago Guerreiro from U. Lisbon and his PhD student), and usable security (Konstantin Beznosov, UBC ECE), and a co-supervised MSc student (Wali Usmani), I qualitatively studied what we term *social insider attacks*. These are instances when someone you know intrudes on your device, your social network account, or other digital resources. Our initial study, funded by a grant from the Officer of Privacy Commissioner of Canada, looked at intrusions on Facebook accounts [CHI 2017.P18]. We found that these attacks are surprisingly common. In a survey we found that 24% of 1000+ respondents carried out these attacks and 21% had been victims of these attacks. As well, we identified common motives behind these attacks and documented some of the consequences for both the attacker and the victim. This work received extensive news coverage, with several radio interviews and articles in the Daily Mail, CBC Canada, Business Tech, and other periodicals.

I am pursuing this research further by considering social insider attacks on mobile phones and studying the circumstances surrounding these attacks. My ultimate goal is to develop defensive measures once we have a sufficient understanding of these attack variants. For example, we have prototyped several continuous authentication approaches to this problem.

Ongoing work: new abstractions for secure data access Today a data owner has control of their data's access policies only as far as the boundaries of their own systems. For example, once you email someone a file, you have no control over what the recipient might do with the file: where they might access it, how many times they might copy it, or who they might send it to.

In a collaboration with Andrew Warfield (previously at UBC, now at Amazon) and three MSc students, I am investigating the use of trusted hardware to implement a novel data-centric abstraction called *trusted capsules*. A trusted capsule bundles data and policy in a way that extends a data owner's policies across system boundaries. Trusted capsules are mobile, programmable, and have dynamically-resolving policies for access control. For example, a company can include a policy for confidential files that only allows access to these files at the GPS coordinates of the company's office.

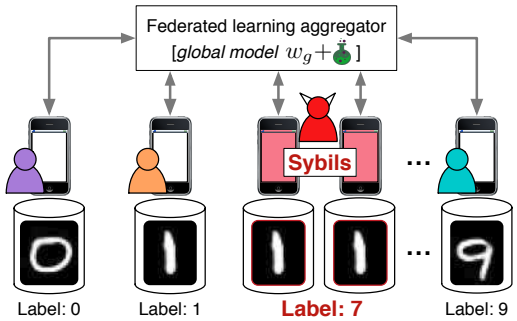

Figure 4: Federated learning with colluding sybils mounting a sybil-based poisoning attack. In the attack two sybils poison the model by computing over images of 1s with the (incorrect) class label 7.

This project has the potential to reshape the guarantees that one can make about data access across devices. We are developing our prototype on ARM development boards with TrustZone as the trusted hardware. But, in the future we plan to transition this design to Android smartphones.

Ongoing work: privacy-preserving machine learning Modern machine learning (ML) implementations use distributed designs to scale to massive datasets and, in some cases, to offload computation to participating clients. Federated learning is a state of the art design deployed by Google that trains on data from mobile devices while keeping the data on the client devices: only model parameters are transferred to a central aggregator to construct the model. This provides a basic level of privacy, and allows clients to compute their model updates locally and independently. However, federated learning makes a risky design tradeoff: clients, who were previously passive data providers, can now contribute arbitrary intermediate values as part of the decentralized training process.

I have been investigating techniques to make distributed ML more private and secure. I have done initial work in this domain with a system for private ML on medical image data [MLMI 2017.P19] in collaboration with Rafeef Abugharbieh (UBC ECE). I am pursuing follow-on topics in this area with three MSc students. We are developing novel defensive strategies that deter poisoning attacks in distributed ML contexts (Figure 4)[12], and are developing a new ML framework that uses a peer-to-peer design for private training that does not depend on a trusted central coordinator.

Representative papers

[1] Stewart Grant, Hendrik Cech, and Ivan Beschastnikh. Inferring and Asserting Distributed System Invariants. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE, pages 1149–1159, New York, NY, USA, 2018. ACM. doi:10.1145/3180155.3180199.

[2] Nodir Kodirov, Sam Bayless, Fabian Ruffy, Ivan Beschastnikh, Holger H. Hoos, and Alan J. Hu. VNF Chain Allocation and Management at Data Center Scale. In *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems*, ANCS, pages 125–140, New York, NY, USA, 2018. ACM. doi:10.1145/3230718.3230724.

[3] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. General LTL Specification Mining. In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering*, ASE, pages 81–92, Washington, DC, USA, 2015. IEEE Computer Society. doi:10.1109/ASE.2015.71.

[4] Yanyan Zhuang, Eleni Gessiou, Steven Portzer, Fraida Fund, Monzur Muhammad, Ivan Beschastnikh, and Justin Cappos. NetCheck: Network Diagnoses from Blackbox Traces. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 115–128, Seattle, WA, 2014. USENIX Association. URL: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/zhuang.

References

[P1] Alexandra Fedorova, Craig Mustard, Ivan Beschastnikh, Julia Rubin, Augustine Wong, Svetozar Miucin, and Louis Ye. Performance Comprehension at WiredTiger. In *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 353–363, Nov 2018.

[P2] Ivan Beschastnikh, Yuriy Brun, Michael D. Ernst, and Arvind Krishnamurthy. Inferring Models of Concurrent Systems from Logs of Their Behavior with CSight. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE, pages 468–479, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2756-5. doi:10.1145/2568225.2568246.

[P3] Ivan Beschastnikh, Yuriy Brun, Jenny Abrahamson, Michael D. Ernst, and Arvind Krishnamurthy. Using Declarative Specification to Improve the Understanding, Extensibility, and Comparison of Model-Inference Algorithms. *IEEE Transactions on Software Engineering (extended and revised version of ICSE 2013 full paper)*, 41(4):408–428, April 2015. ISSN 0098-5589. doi:10.1109/TSE.2014.2369047.

[P4] Tony Ohmann, Michael Herzberg, Sebastian Fiss, Armand Halbert, Marc Palyart, Ivan Beschastnikh, and Yuriy Brun. Behavioral Resource-aware Model Inference. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE, pages 19–30, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3013-8. doi:10.1145/2642937.2642988.

[P5] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. General LTL Specification Mining. In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering*, ASE, pages 81–92, Washington, DC, USA, 2015. IEEE Computer Society. ISBN 978-1-5090-0025-8. doi:10.1109/ASE.2015.71.

[12] Our recent work in this area is on Arxiv: https://arxiv.org/abs/1808.04866

[P6] Tien-Duy B. Le, Xuan-Bach D. Le, David Lo, and Ivan Beschastnikh. Synergizing Specification Miners through Model Fissions and Fusions. In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering*, ASE, pages 115–125, Nov 2015. doi:10.1109/ASE.2015.83.

[P7] Stewart Grant, Hendrik Cech, and Ivan Beschastnikh. Inferring and Asserting Distributed System Invariants. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE, pages 1149–1159, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5638-1. doi:10.1145/3180155.3180199.

[P8] Keheliya Gallaba, Ali Mesbah, and Ivan Beschastnikh. Don't Call Us, We'll Call You: Characterizing Callbacks in Javascript. In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10, Oct 2015. doi:10.1109/ESEM.2015.7321196.

[P9] Keheliya Gallaba, Quinn Hanam, Ali Mesbah, and Ivan Beschastnikh. Refactoring Asynchrony in JavaScript. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 353–363, Sept 2017. doi:10.1109/ICSME.2017.83.

[P10] Matheus Nunes, Harjeet Lalh, Ashaya Sharma, Augustine Wong, Svetozar Miucin, Alexandra Fedorova, and Ivan Beschastnikh. Studying Multi-threaded Behavior with TSViz. In *Proceedings of the 39th International Conference on Software Engineering Companion (Tool demonstration track)*, ICSE, pages 35–38, Piscataway, NJ, USA, 2017. IEEE Press. ISBN 978-1-5386-1589-8. doi:10.1109/ICSE-C.2017.9.

[P11] Daniel Rozenberg, Ivan Beschastnikh, Fabian Kosmale, Valerie Poser, Heiko Becker, Marc Palyart, and Gail C. Murphy. Comparing Repositories Visually with Repograms. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR, pages 109–120, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4186-8. doi:10.1145/2901739.2901768.

[P12] Ivan Beschastnikh, Patty Wang, Yuriy Brun, and Michael D. Ernst. Debugging Distributed Systems. *Commun. ACM*, 59(8):32–37, July 2016. ISSN 0001-0782. doi:10.1145/2909480.

[P13] Yanyan Zhuang, Eleni Gessiou, Steven Portzer, Fraida Fund, Monzur Muhammad, Ivan Beschastnikh, and Justin Cappos. NetCheck: Network Diagnoses from Blackbox Traces. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 115–128, Seattle, WA, 2014. USENIX Association. ISBN 978-1-931971-09-6. URL `https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/zhuang`.

[P14] Justin Cappos, Yanyan Zhuang, Albert Rafetseder, and Ivan Beschastnikh. Tsumiki: A Meta-Platform for Building Your Own Testbed. *IEEE Transactions on Parallel and Distributed Systems*, 2018. ISSN 1045-9219. doi:10.1109/TPDS.2018.2846242.

[P15] Sam Bayless, Nodir Kodirov, Ivan Beschastnikh, Holger H. Hoos, and Alan J. Hu. Scalable Constraint-based Virtual Data Center Allocation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI, pages 546–554. AAAI Press, 2017. ISBN 978-0-9992411-0-3. URL `http://dl.acm.org/citation.cfm?id=3171642.3171722`.

[P16] Nodir Kodirov, Sam Bayless, Fabian Ruffy, Ivan Beschastnikh, Holger H. Hoos, and Alan J. Hu. VNF Chain Allocation and Management at Data Center Scale. In *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems*, ANCS, pages 125–140, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5902-3. doi:10.1145/3230718.3230724.

[P17] Amanda Carbonari and Ivan Beschasnikh. Tolerating Faults in Disaggregated Datacenters. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets-XVI, pages 164–170, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5569-8. doi:10.1145/3152434.3152447.

[P18] Wali Ahmed Usmani, Diogo Marques, Ivan Beschastnikh, Konstantin Beznosov, Tiago Guerreiro, and Luís Carriço. Characterizing Social Insider Attacks on Facebook. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI, pages 3810–3820, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4655-9. doi:10.1145/3025453.3025901.

[P19] Alborz Amir-Khalili, Soheil Kianzad, Rafeef Abugharbieh, and Ivan Beschastnikh. Scalable and Fault Tolerant Platform for Distributed Learning on Private Medical Data. In *Machine Learning in Medical Imaging*, pages 176–184, Cham, 2017. Springer International Publishing. ISBN 978-3-319-67389-9.