Verification and Validation of Concurrent and Distributed Systems (Track Summary)

Marieke Huisman[1] and Cristina Seceleanu[2]

[1] University of Twente, Enschede, The Netherlands
[2] Mälardalen University, Sweden

Abstract. Usually, greater concurrency is the goal of any distributed system, yet distribution also introduces issues of consistency and separate failure domains. With the increase of device connectivity and virtualization techniques, developing correct and reliable concurrent and distributed systems characterized by high performance is notoriously difficult. This requires novel verification techniques, or extensions, adaptations and improvements of existing ones, to address emergent problems. The track on Verification and Validation of Concurrent and Distributed Systems aims to discuss key challenges that need to be tackled in order to enable the efficient and scalable assurance of modern concurrent and distributed systems, as well as present methods and tools that bear the promise to achieve the latter.

1 Motivation and Goals

Concurrent and distributed systems are becoming omnipresent for two reasons. First of all, concurrency and distribution are necessary to fulfill performance requirements of modern software. Second, such systems' paradigms are a natural fit with most underlying application domains. However, concurrent and distributed systems add a lot of extra complexity to systems, and allow many different kinds of errors to occur, which cannot happen in sequential software. As Leslie Lamport, known for his seminal work in distributed systems, famously said: "A distributed system is one in which the failure of a computer you did not even know existed can render your own computer unusable" [13]. Similarly, for concurrent systems, an error might occur in one execution, and then disappear in the next execution of the system.

Nevertheless, over the last years, we see a plethora of different tools and techniques to reason about distributed systems [7, 12, 17] and concurrent software [3, 5, 8, 10, 14, 16] being developed and applied under different specific scenarios.

The next step is then to think about how to develop verification techniques for systems that combine distributed and concurrent aspects. One can refine this ambition by asking: How can verification techniques for concurrent systems benefit from verification techniques for distributed systems, and vice versa?

The **Verification and Validation of Concurrent and Distributed Systems**(VVCDS) track focuses on providing answers to these questions, by presenting invited papers that propose models, techniques, and tools for the rigorous

analysis of various concurrent and distributed systems. The included contributions give a good overview of the current state-of-the-art in the verification of concurrent and distributed systems, and propose solutions to difficult problems related to modern topics such as cloud-native microservice architectures, blockchain synchronization, or validation and dynamic monitoring of multi-threaded programs, but also to long-standing issues such as ensuring quality and correctness of distributed protocols used in industry, taming the complexity of distributed systems design via incremental development, or applying academic tools for verifying distributed systems in an industrial context. The track closes with a discussion to look further ahead: given the current-state-of-the-art, how can we combine verification and validation techniques for concurrency and distribution such that not only the systems' specific issues are tackled, but also the scalability and applicability in industry of the proposed approaches are achieved. For this, we would like to understand similarities and differences between concurrent yet not distributed, and truly distributed systems, and their respective techniques of verification and validation, in an attempt to leverage the key strengths of such approaches and reduce their potential weaknesses.

Finally, we would like to express our deep gratitude to the ISoLA organisers, in particular Prof. Tiziana Margaria and Prof. Bernhard Steffen, for working so hard to provide such a wonderful platform for our and other tracks, enabling lively and creative interaction between individuals and communities, helping us all to not forget the bigger picture of working for the development of systems that people can rely on.

2 Overview of Contributions

In *Step-wise Development of Provably Correct Actor Systems* [1], the authors Bernhard K. Aichernig and Benedikt Maderbacher present an approach for the incremental formal development of actor systems via refinement, in the Event-B tool. The assumption is that distributed software modeled using the actor-based paradigm benefits from the latter's simple asynchronous message passing for interprocess communication, and does not suffer from the common pitfall of shared mutable state. The technique is shown on Agha's classical factorial algorithm, which has been proven correct via a series of refinement steps, starting from an abstract description. The authors have also proven deadlock-freeness and convergence from which the termination of a single computation follows. The paper shows that the key to handling complexity is to keep the actor model simple enough yet as faithful to reality as possible, such that all proofs can be resolved automatically.

In *Violation Witnesses and Result Validation for Multi-Threaded Programs* [2], the authors Dirk Beyer and Karlheinz Friedberger present how the standard format for violation witnesses for program analysis tools is extended to multi-threaded programs. It discusses what information about threading needs to be captured in the witness. It turns out that the main information that is needed is the thread identifier that executes an instruction, whereas other information

about monitors etc. does not have to be kept. The paper also presents a validation tool that can be used to confirm detected violations. An extensive experimental evaluation is done, which confirms that for larger problems validation time is faster than the original verification time.

In *Tendermint Blockchain Synchronization: Formal Specification and Model Checking* [4], the authors Sean Braithwaite, Ethan Buchman, Igor Konnov, Zarko Milosevic, Ilina Stoilkovska, Josef Widder, and Anca Zamfir present a formal specification of the blockchain synchronization protocol of Tendermint, called Fastsync. The protocol is firstly specified in English language, after which it is decomposed and abstracted in TLA+. Various safety and liveness properties are encoded in the property language of checkers TLC and Apalache, and the resulting specifications are model checked. The generated counter-examples have led to better understanding of different issues of both the specification and implementation of Fastsync. The authors discuss also the lessons learned, including the scalability issues that has forced them to resort to bounded model checking with Apalache, in order to account for faulty peers too.

In *Safe Sessions of Channel Actions in Clojure: A Tour of the Discourje Project* [6], the authors Ruben Hamers and Sung-Shik Jongmans give an overview of the Discourje project. Discourje supports dynamic monitoring of concurrent Clojure programs. The monitored properties describe the expected system behavior at an abstract level, and the monitor implementation then checks whether the implementation behaves as specified. The system is illustrated by three different examples, each illustrating different aspects of the specifications and implementations. It also discusses how the monitor is added into the implementation. The paper completes with a short summary of the Discourje formalization.

In *Modular Verification of Liveness Properties of the I/O Behavior of Imperative Programs* [9], the author Bart Jacobs describes a modular verification technique to reason about I/O behaviour of programs. The verification technique allows to verify properties such as eventually something will happen, response and reactive properties, and persistence properties (something will eventually become true forever). The paper first illustrates typical specifications for all these patterns. It then formalizes the verification technique, and discusses how verification proceeds for some of the examples discussed earlier.

In *Formal Verification of an Industrial Distributed Algorithm: an Experience Report* [11], the authors Nikolai Kosmatov, Delphine Longuet and Romain Soulat report on experiences with modeling and verification of some consensus algorithms. Their paper explains that even though the literature contains many verified consensus algorithms, in industrial practice slight variations are often needed, so we need techniques to reason about those easily. The paper sketches a consensus algorithm that is used at Thales on a distributed internet-of-things system. The algorithm is modeled in two different ways: fully explicitly and in the form of an abstract model, where a single node is modeled, interacting with a model that represents the rest of the network. The authors experiment with 3 different tools (SafeProver, CBMC and KLEE) to analyze the model, and they discuss the lessons learned from these experiments. In particular, the

experiments show that it is indeed possible to use formal analysis tools in an industrial setting, but more work is needed to turn this into daily industrial practice.

In *Deploying TESTAR to enable remote testing in an industrial CI pipeline: a case-based evaluation* [15], the authors Fernando Pastor Ricós, Pekka Aho, Tanja Vos, Ismael Torres Boigues, Ernesto Calás Blasco, and Héctor Martínez Martínez describe the application of an academic tool for testing, called TESTAR, on a commercially-available distributed system. The technical challenges of a distributed software system, which the tool has not been initially designed for, are described, as well as how these gaps have been bridged. The paper also highlights the differences between industry and academia, in approaching problems and their corresponding classification, respectively.

In *A Formal Model of the Kubernetes Container Framework* [18], the authors Gianluca Turin, Andrea Borgarelli, Simone Donetti, Einar Broch Johnsen, S. Lizeth Tapia Tarifa, and Ferruccio Damiani develop a formal model of resource consumption and scaling for Kubernetes containerized micro-services. The framework, encoded in Real-time ABS, is intended to provide a platform in which various configurations can be assessed before the actual deployment. The authors validate the model by comparing an instance of the framework, under several scenarios, to observations of a real system running on a high-performance application cluster called HPC4AI. The work paves the way towards the model-based development of native-cloud solutions based on Kubernetes.

References

1. Bernhard K. Aichernig and Benedikt Maderbacher. Step-wise development of provably correct actor systems. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA 2020*, LNCS. Springer, 2020. (in this volume).
2. Dirk Beyer and Karlheinz Friedberger. Violation witness and result validation for multi-threaded programs. implementation and evaluation with CPAchecker. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA 2020*, LNCS. Springer, 2020. (in this volume).
3. S. Blom, S. Darabi, M. Huisman, and W. Oortwijn. The VerCors Tool Set: Verification of Parallel and Concurrent Software. In *iFM*, volume 10510 of *LNCS*, pages 102 – 110. Springer, 2017.
4. Sean Braithwaite, Ethan Buchman, Igor Konnov, Zarko Milosevic, Ilina Stoilkovska, Josef Widder, and Anca Zamfir. Tendermint blockchain synchronization: Formal specification and model checking. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA 2020*, LNCS. Springer, 2020. (in this volume).
5. P. da Rocha Pinto, T. Dinsdale-Young, and P. Gardner. TaDA: A logic for time and data abstraction. In *European Conference on Object-Oriented Programming (ECOOP)*, LNCS. Springer, 2014.
6. Ruben Hamers and Sung-Shik Jongmans. Safe sessions of channel actions in clojure: A tour of the discourje project. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA 2020*, LNCS. Springer, 2020. (in this volume).
7. C. Hawblitzel, J. Howell, M. Kapritsos, J.R. Lorch, B. Parno, M.L. Roberts, S.T.V. Setty, and B. Zill. Ironfleet: Proving Practical Distributed Systems Correct. In

Proceedings of the 25th Symposiumon Operating Systems Principles, SOSP 2015, pages 1–17. ACM, 2015.
8. B. Jacobs, J. Smans, P. Philippaerts, F. Vogels, W. Penninckx, and F. Piessens. VeriFast: A powerful, sound, predictable, fast verifier for C and Java. In *NFM*, 2011.
9. Bart Jacobs. Modular verification of liveness properties of the I/O behavior of imperative programs. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA 2020*, LNCS. Springer, 2020. (in this volume).
10. R. Jung, D. Swasey, F. Sieczkowski, K. Svendsen, A. Turon, L. Birkedal, and D. Dreyer. Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning. In *POPL*, pages 637–650. ACM, 2015.
11. Nikolai Kosmatov, Delphine Longuet, and Romain Soulat. Formal verification of an industrial distributed algorithm: an experience report. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA 2020*, LNCS. Springer, 2020. (in this volume).
12. M. Krogh-Jespersen, A. Timany, M.E. Ohlenbusch, S.O. Gregersen, and L. Birkedal. Aneris: A Mechanised Logic for Modular Reasoning about Distributed Systems. In *Proceedings of European Symposium on Programming, Programming Languages and Systems, ESOP 2020*, volume 12075 of *LNCS*, pages 336–365. Springer Cham, 2020.
13. Leslie Lamport. Distribution, May 1987. Email message sent to a DEC SRC bulletin board at 12:23:29 PDT on 28 May 87.
14. P. Müller, M. Schwerhoff, and A.J. Summers. Viper - a verification infrastructure for permission-based reasoning. In *VMCAI*, 2016.
15. Fernando Pastor Ricós, Pekka Aho, Tanja Vos, Ismael Torres Boigues, Ernesto Calás Blasco, and Héctor Martínez Martínez. Deploying testar to enable remote testing in an industrial ci pipeline: a case-based evaluation. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA 2020*, LNCS. Springer, 2020. (in this volume).
16. I. Sergey, A. Nanevski, and A. Banerjee. Specifying and Verifying Concurrent Algorithms with Histories and Subjectivity. In *ESOP*, volume 9032 of *LNCS*, pages 333–358. Springer, 2015.
17. I. Sergey, J.R. Wilcox, and Z. Tatlock. Programming and Proving with Distributed Protocols. In *Proceedings of PACMPL2(POPL)*, volume 2, pages 28:1–28:30. ACM, 2018.
18. Gianluca Turin, Andrea Borgarelli, Simone Donetti, Einar Broch Johnsen, S. Lizeth Tapia Tarifa, and Ferruccio Damiani. A formal model of the kubernetes container framework. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA 2020*, LNCS. Springer, 2020. (in this volume).