Software Qual J (2018) 26:851–853
https://doi.org/10.1007/s11219-018-9415-9

Guest editorial: special issue on concurrent software quality

Zijiang Yang[1] · **Ting Liu**[2] · **Daniel Xiapu Luo**[3] · **Chao Wang**[4]

Published online: 25 July 2018

Today, multi-core hardware and cloud platforms have become ubiquitous, which puts us at a fundamental turning point in software development. In order for software applications to benefit from the continued exponential advances in hardware systems, the applications will need to be well-written concurrent programs. Although for the past decade we have witnessed incrementally more programmers writing concurrent programs, the vast majority of applications today are still sequential due to the lack of effective tools that support the development of concurrent programs. This trend necessitates the use of advanced methods to redesign the existing tools that remain optimized for sequential program development. We are interested in research that advances the state of the art in different phases of concurrent software development, with the goal to help developers write high quality concurrent programs.

Following an open call for papers, the special issue received a total of 19 submissions, of which 2 survey papers and 9 research papers were accepted for publication. Each manuscript was reviewed by at least two reviewers.

✉ Zijiang Yang
zijiang.yang@wmich.edu

Ting Liu
tingliu@mail.xjtu.edu.cn

Daniel Xiapu Luo
csxluo@comp.polyu.edu.hk

Chao Wang
wang626@usc.edu

[1] Western Michigan University, 1903 Western Michigan Avenue, Kalamazoo, MI 49008, USA

[2] Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China

[3] The Hong Kong Polytechnic University, 11 Yuk Choi Rd, Hung Hom, Hong Kong

[4] University of Southern California, Los Angeles, CA 90007, USA

The first survey paper "A systematic survey on automated concurrency bug detection, exposing, avoidance and fixing techniques", by Haojie Fu, Zan Wang, Xiang Chen and Xiangyu Fan classifies existing research on concurrency bugs into four categories: automated concurrency bug detection, exposing bugs, avoiding bugs, and fixing bugs. Classical benchmarks and several research groups on concurrency are also introduced. In addition, the challenge issues are stated, together with the state of art and future directions.

The second survey paper "A Survey on Dynamic Mobile Malware Detection", by Ping Yan and Zheng Yan introduces the definition, evolution, classification, and security threats of mobile malware. It compares, analyzes and comments on existing methods of dynamic mobile malware detection. Several open issues are proposed to motivate future research directions.

The paper "Contributions for the Structural Testing of Multithreaded Programs: Coverage Criteria, Testing Tool and Experimental Evaluation", by Silvana Morita Melo, Simone do Rocio Senger Souza, Felipe Santos Sarmanho, and Paulo Sergio Lopes Souza discusses test coverage criteria for the structural testing of multithreaded programs in terms of cost, effectiveness, and strength. Experiments were designed to prove the efficiency of these coverage criteria.

In "Asynchronous Multi-Process Timed Automata" Guoqiang Li, Li Liu and Akira Fukuda presents an asynchronous multi-process timed automata (APTA) model to support the verification of asynchronous programs. A Process Timed Automaton was used to abstract the process and multisets were applied to buffer triggered states.

The paper "A Fault Tolerant Election-based Deadlock Detection Algorithm in Distributed Systems" by Wei Lu, Yong Yang, Liqiang Wang, Weiwei Xing, Xiaoping Che and Lei Chen improveds previous work to tolerate a certain extent of communication disconnection between the control processes. The new central controller could detect the deadlock by monitoring whether there is at least one process that communicate with other processes.

In "Detecting Potential Deadlocks Through Change Impact Analysis" Chelsea Metcalf and Tuba Yavuz statically detects program code changes that may introduce potential deadlocks (for multithreaded Java applications). A heuristic method was designed to automatically infer the intended lock order. The authors demonstrate its performance on several large-scale software projects.

The paper "System-Level Attacks against Android by Exploiting Asynchronous Programming" by Ting Chen, Xiaoqi Li, Xiapu Luo and Xiaosong Zhang checks whether the asynchronous constructs are used properly on the Android platform and whether hackers can take advantage of unprotected intents to launch attacks. A large number of unprotected intents were found, which could be sent by third-party applications without protection.

In "A Multi-Aspect Online Tuning Framework for HPC Applications" Michael Gerndt, Siegfried Benkner, Eduardo Cesar, Enes Bajrovic, Jiri Dokulil, Carla Guillen, Robert Mijakovic and Anna Sikora proposes the Periscope Tuning Framework (PTF) to address performance issues, combining both performance analysis and performance tuning. PTF supports tuning of different performance aspects of parallel applications through a set of tuning plugins. PTF also provides meta-plugins that combine individual plugins.

The paper "Performance Tuning for Actor Programs Through Decoupled Concurrency" by Tai Nguyen and Xinghui Zhao addresses the problem of separating a program's concurrency from its functionality through two tuning policies: static tuning and dynamic tuning. Their experiments show that this approach is effective in achieving high performance on high-end computing architectures. Moreover, it does not introduce extra overhead on the hardware.

In “Dynamic Structure Measurement for Distributed Software” Wuxia Jin, Ting Liu, Yu Qu, Qinghua Zheng, Di Cui and Jianlei Chi proposes intra-component and inter-component dependencies to analyze the structure of distributed software, instead of calls or dependencies among functions. These new indexes were useful to guide designers and developers to refactor the structure of distributed software.

The paper “Cooperative Decoupled Processes” by Andi Bejleri, Mira Mezini, Patrick Eugster and Elton Domnori proposes an event-driven programming model called CDP (cooperative decoupled processes) to bridge the gap between decoupling and reasoning about global control flow without sacrificing decoupling. The authors discuss how the CDP model addresses the issues of stack management and shared events while retaining decoupling in three dimensions (space, synchronization and time).

We would like to thank all the contributors for their hard work in preparing and updating the manuscripts. We would also like to thank the reviewers for their time and the effort in providing constructive comments. Finally, we thank the Editor-in-Chief, Professor Rachel Harrison, and the editorial staff for their patience and hard work in getting this special issue ready for publication.