RESEARCH CENTRE

Inria Paris Centre
at Sorbonne University

2023
ACTIVITY REPORT

Project-Team
WHISPER

Well Honed Infrastructure Software for Programming Environments and Runtimes

DOMAIN

Networks, Systems and Services, Distributed Computing

THEME

Distributed Systems and middleware

Inria

Contents

Project-Team WHISPER

Creation of the Project-Team: 2015 December 01

Keywords

Computer sciences and digital sciences

A1. – Architectures, systems and networks

A1.1.1. – Multicore, Manycore

A1.1.3. – Memory models

A1.1.13. – Virtualization

A2.1.6. – Concurrent programming

A2.1.10. – Domain-specific languages

A2.2.1. – Static analysis

A2.2.5. – Run-time systems

A2.2.8. – Code generation

A2.4. – Formal method for verification, reliability, certification

A2.4.3. – Proofs

A2.5. – Software engineering

A2.5.4. – Software Maintenance & Evolution

A2.6.1. – Operating systems

A2.6.2. – Middleware

A2.6.3. – Virtual machines

Other research topics and application domains

B5. – Industry of the future

B5.2.1. – Road vehicles

B5.2.3. – Aviation

B5.2.4. – Aerospace

B6.1. – Software industry

B6.1.1. – Software engineering

B6.1.2. – Software evolution, maintenance

B6.5. – Information systems

B6.6. – Embedded systems

1 Team members, visitors, external collaborators

Research Scientists

- Julia Lawall [Team leader, INRIA, Senior Researcher, HDR]
- Jean-Pierre Lozi [INRIA, Researcher]

PhD Students

- Papa Assane Fall [INRIA]
- Victor Laforet [INRIA, from Oct 2023]
- Himadri Pandya [INRIA]

Technical Staff

- Tomas Faltin [INRIA, Engineer, from Mar 2023]
- Keisuke Nishimura [INRIA, Engineer, from Oct 2023]

Interns and Apprentices

- Constantin Gierczak-Galle [INRIA, Intern, from Mar 2023 until Jul 2023]
- Magnus Helle Kalland [INRIA, Intern, from Jul 2023 until Sep 2023]
- Victor Laforet [INRIA, Intern, from Mar 2023 until Aug 2023]

Administrative Assistant

- Nelly Maloisel [INRIA]

Visiting Scientists

- Tathagata Roy [IIIT Kalyani, from Dec 2023]
- Tathagata Roy [IIIT Kalyani, from Feb 2023 until Sep 2023]
- Tathagata Roy [IIIT Kalyani, until Jan 2023]

2 Overall objectives

The focus of Whisper is on how to develop (new) and improve (existing) infrastructure software. Infrastructure software (also called systems software) is the software that underlies all computing. Such software allows applications to access resources and provides essential services such as memory management, synchronization and inter-process interactions. Starting bottom-up from the hardware, examples of infrastructure software include operating systems, virtual machine hypervisors, managed runtime environments, and standard libraries. For such software, efficiency and correctness are fundamental. Any overhead will impact the performance of all supported applications. Any failure will prevent the supported applications from running correctly. Whisper addresses these dual problems of infrastructure software performance and correctness, both in a given instance of the software and as the software evolves.

In terms of methodology, Whisper is at the interface of the domains of operating systems, software engineering and programming languages. Our approach is to combine the study of problems in the development of real-world infrastructure software with concepts in programming language design and implementation, *e.g.*, of domain-specific languages, and knowledge of low-level system behavior. A focus

of our work is on providing support for legacy code, while taking the needs and competences of ordinary system developers into account. We will put an emphasis on achieving measurable improvements in performance and correctness in practice, and on feeding these improvements back to the infrastructure-software developer community.

We focus on two main axes. The first axis, system performance, targets the performance of applications, in both bare-metal and virtual environments, based on the impact of operating-system services. This direction not only involves proposing new algorithms, but also designing tools and methodologies to help developers understand the system behavior. When addressing these areas, we will explore the use of DSLs, when appropriate, to enhance the usability, configurability, reliability, and robustness of the proposed approaches. The second axis, system re-engineering, targets the quality of legacy infrastructure software. Infrastructure software tends to evolve faster than many of its users are able to keep up with, leading to massive ecosystem fragmentation. Building on tools such as Coccinelle for automating widespread code changes, we will investigate how to facilitate and improve the reliability of the maintenance of such variants, for example in terms of identifying, adapting, and applying relevant bug fixes.

3 Research program

The research program of the Whisper team is designed around two main axes: system performance and system re-engineering.

3.1 System performance

Our work on system performance focuses on scheduling, virtualization, synchronization, and the interactions between them. A task scheduler decides which runnable task should be able to execute on a CPU, and, in a multicore setting, on which core that execution should take place. As the task scheduler determines how much CPU time the tasks of an application receive, as well as whether the chosen CPU has effcient access to needed hardware resources (network, accelerators, caches, etc.), it has a large impact on application performance. In a general-purpose programming setting, the task scheduler is typically located in the operating system kernel, and is expected to serve the needs of all applications. It has no knowledge of future application behavior. Such a setting raises many challenges. We are currently focusing on four main issues: scheduling and virtualization, making locks aware of scheduler preemptions, control over scheduling from the user level, and task scheduling for heterogeneous architectures.

Virtualization decomposes the system support into one or more standard operating systems, known as guest OSes, that in turn run on a minimalistic operating system, known as a hypervisor, which provides isolation between the guest OSes and provides access to the hardware. A guest OS schedules application tasks based on its view of the set of available CPUs, but these CPUs are actually just tasks that are in turn scheduled by the hypervisor on the physical CPUs of the machine. This dual level of scheduling introduces the risk of contradictory scheduling strategies. We are investigating the impact of this dual level of scheduling on the performance of applications in a highly multicore setting.

Effcient lock algorithms are key to ensuring application scalability on multi-core architectures [22, 25], [5]. Two families of lock algorithms exist: (1) with blocking locks, tasks sleep in a wait list until the lock is free, whereas (2) with spin locks, tasks busy-wait on the value of an atomic boolean variable that records whether or not the lock is free. While blocking locks consume less CPU, they are also less reactive, as acquiring the lock requires waking up the acquiring task, which entails a context switch. Spin locks, on the other hand, are very reactive, as the lock is acquired by a busy-waiting task as soon as the lock is released. But spinlocks may perform very poorly in overloaded scenarios, because spinning tasks can preempt the lock-holding task, preventing any progress on the critical path. We would like to turn spin locks into blocking locks in an overloaded scenario, relying on BPF to safely inject code into the kernel to detect when the Linux kernel scheduler preempts the lock holder.

Recently, several frameworks has been developed (notably, ghOSt from Google [27] and sched-ext from Meta [20]) that allow the user level to inject new schedulers into the Linux kernel. We would like to investigate how such a framework can be used to improve performance, using scheduling-decision-making processes that are not feasible in the kernel. For example, machine learning could be beneficial for inferring task properties and guiding scheduling accordingly, but is not practical in the kernel, due

to the latency involved, memory constraints, and the lack of kernel support for floating point numbers. Our goal would be to provide a proof-of-concept, as it is not our focus to develop new machine-learning algorithms. We will also consider whether it can be useful to export other kinds of operating system services to user level, such as memory management. Finally, we will consider whether it is possible to design a generic kernel interface for exporting to the user level performance-critical operating-system services. This work is in collaboration with Alain Tchana and Renaud Lachaize of the Erods team at LIG, as part of the Inria Défi OS

The heuristics for scheduling employed by the Linux kernel have evolved over time, primarily in a homogeneous setting, where all cores have similar properties. Today, heterogeneous computing is becoming more prevalent, combining powerful processors for compute-intensive tasks with slower, more energy-efficient, processors for lighter weight tasks. Such an architecture is found in the big.LITTLE processors from Arm, typically used in mobile environments, and is now moving to desktop computing, with the Alder Lake architecture from Intel. We will assess the degree to which the Linux kernel task scheduler is able to effectively support such architectures and propose improvements, in terms of both meeting application performance requirements and saving energy. This work is in collaboration with David Bromberg and Djob Mvondo of the WIDE team at Inria-Rennes, as part of the Inria Défi OS.

3.2 System re-engineering

In the area of system re-engineering, we are interested in improving the quality of infrastructure software as well as in automating the transformations required to make it possible to use such software in new settings. Our work builds on our previous work on Coccinelle [2], a program transformation system for C code that has been extensively used on the Linux kernel and other C infrastructure software.

We are extending Coccinelle to C++ and to Rust. The extension to C++ reuses the existing Coccinelle implementation, simply adding the C++-specific constructs. The goal of this work, in collaboration with Michele Martone of the Leibniz Super computing center, is to consider how Coccinelle can be used to adapt existing HPC software for use on GPUs, by introducing the use of calls to relevant libraries. The extension to Rust, on the other hand, is a complete re-implementation of Coccinelle, to benefit from tools being developed by the Rust community for parsing and pretty printing (Rust Analyzer and rustfmt) Rust code. The reuse of these tools has made it possible to quickly develop a preliminary implementation for the full Rust language. In the future, we may consider how to generalize this process to other languages, addressing the challenge that generic parsing tools may not present information in a way that is suitable for program transformation, for example in the choice of non-terminals.

We are considering how Coccinelle can be used to facilitate long-term software maintenance. Many software projects are obliged to maintain multiple variants, *e.g.*, to meet the demands of users who rely on the properties of old variants. An essential part of software maintenance is thus to apply new bug fixes to these old variants. Intervening changes in the software, however, may imply that fixes can not be applied to older versions directly. We are looking into how Coccinelle can be used to collect and exploit information about recent code changes, to adapt new bug fixes to older source code.

Today, there are numerous tools for scanning infrastructure software for common types of vulnerabilities, such as NULL pointer dereferences and buffer overflows. Avoiding such issues, however, is not sufficient to ensure that the code behaves correctly; the code must also implement the intended algorithm. Checking algorithmic properties, however, is much more challenging, because each service implements its own algorithm. Formal verification tools such as Frama-C [21] can make it possible to express and verify such properties, but remain challenging to use, particularly for developers without experience in formal verification. The challenge is compounded by the fact that the code may evolve frequently, making any proofs quickly out of date. We are investigating how to design tools that can facilitate proofs about Linux kernel code, both in how to extract relevant parts of the code base, to reduce the amount of code that has to be considered by the proving process, and in how proofs can be evolved in response to changes in the code base, analogous to the adaptation of bug fixes over multiple versions.

4 Application domains

4.1 Linux

Linux is an open-source operating system that is used in settings ranging from embedded systems to supercomputers. Linux kernel v6.1, released in December 2022, comprises over 23 million lines of code, and supports 23 different families of CPU architectures, around 50 file systems, and thousands of device drivers. Linux is also in a rapid stage of development, with new versions being released roughly every 2.5 months. Recent versions have each incorporated around 13,500 commits, from around 1500 developers. These developers have a wide range of expertise, with some providing hundreds of patches per release, while others have contributed only one. Overall, the Linux kernel is critical software, but software in which the quality of the developed source code is highly variable. These features, combined with the fact that the Linux community is open to contributions and to the use of tools, make the Linux kernel an attractive target for software researchers. Tools that result from research can be directly integrated into the development of real software, where it can have a high, visible impact.

Starting from the work of Engler et al. [24], numerous research tools have been applied to the Linux kernel, typically for finding bugs [23, 33, 30, 37] or for computing software metrics [28, 43]. In our work, we have studied generic C bugs in Linux code [8], bugs in function protocol usage [31, 32], issues related to the processing of bug reports [41] and crash dumps [26], and the problem of backporting [38, 42], illustrating the variety of issues that can be explored on this code base. Unique among research groups working in this area, we have furthermore developed numerous contacts in the Linux developer community. These contacts provide insights into the problems actually faced by developers and serve as a means of validating the practical relevance of our work.

4.2 Device Drivers

Device drivers are essential to modern computing, to provide applications with access, via the operating system, to physical devices such as keyboards, disks, networks, and cameras. Development of new computing paradigms, such as the internet of things, is hampered because device driver development is challenging and error-prone, requiring a high level of expertise in both the targeted OS and the specific device. Furthermore, implementing just one driver is often not sufficient; today's computing landscape is characterized by a number of OSes, *e.g.*, Linux, Windows, MacOS, BSD and many real time OSes, and each is found in a wide range of variants and versions. All of these factors make the development, porting, backporting, and maintenance of device drivers a critical problem for device manufacturers, industry that requires specific devices, and even for ordinary users.

The last twenty years have seen a number of approaches directed towards easing device driver development. Réveillère *et al.* propose Devil [36], a domain-specific language for describing the low-level interface of a device. Chipounov *et al.* propose RevNic, [19] a template-based approach for porting device drivers from one OS to another. Ryzhyk *et al.* propose Termite, [39, 40] an approach for synthesizing device driver code from a specification of an OS and a device. Currently, these approaches have been successfully applied to only a small number of toy drivers. Indeed, Kadav and Swift [29] observe that these approaches make assumptions that are not satisfied by many drivers; for example, the assumption that a driver involves little computation other than the direct interaction between the OS and the device. At the same time, a number of tools have been developed for finding bugs in driver code. These tools include SDV [18], Coverity [24], CP-Miner, [34] PR-Miner [35], and Coccinelle [7]. These approaches, however, focus on analyzing existing code, and do not provide guidelines on structuring drivers.

In summary, there is still a need for a methodology that first helps the developer understand the software architecture of drivers for commonly used operating systems, and then provides tools for the maintenance o0f existing drivers.

4.3 Multicore computing

Today, multicore computing is fundamental across a wide variety of industries. Applications such as machines learning, scientific computing, image processing, etc., run large numbers of threads on highly multicore processors. Such applications must furthermore contend with a variety of hardware

architectures, with different CPU and memory topologies, CPU capacities, access to external resources such as storage, networking and accellerators, etc. Virtualization facilitates hardware sharing, but complicates access to and reasoning about hardware resources. Our work on system performance targets helping multicore applications run more efficiently, and helping developers understand their application performance, to enable them to find appropriate solutions.

5 Highlights of the year

5.1 Awards

The paper "OFence: Pairing Barriers to Find Concurrency Bugs in the Linux Kernel," by Baptiste Lepers, Josselin Giet, Julia Lawall, and Willy Zwaenepoel, received a Best Paper award at EuroSys 2023. This work grew out of our Inria Associated Team CGS that ended in 2022.

6 New software, platforms, open data

6.1 New software

6.1.1 Coccinelle

Keywords: Code quality, Evolution, Infrastructure software

Functional Description: Coccinelle is a tool for code search and transformation for C programs. It has been extensively used for bug finding and evolutions in Linux kernel code. Extensions to support C++ and Rust are in progress. A prototype has been developed for Java.

URL: https://coccinelle.gitlabpages.inria.fr/website/

Contact: Julia Lawall

Participants: Gilles Muller, Julia Lawall, Nicolas Palix, Rene Rydhof Hansen, Thierry Martinez

Partner: IRILL

6.1.2 Coccinelle for Rust

Keyword: Rust

Functional Description: Coccinelle is a tool for program matching and transformation, relying on rules expressed as fragments of source code, amounting to semantic patch. Coccinelle for Rust brings the power of Coccinelle to software written in the Rust programming language. Just as Coccinelle was designed around the practical needs for software evolution in the Linux kernel, Coccinelle for Rust has benefited from feedback from the Rust for Linux community.

URL: https://gitlab.inria.fr/coccinelle/coccinelleforrust,https://rust-for-linux.com/coccinelle-for-rust

Contact: Julia Lawall

Participants: Julia Lawall, Tathagata Roy

6.1.3 schedgraph

Keywords: Task scheduling, Linux kernel

Functional Description: Schedgraph is a collection of tools for visualizing Linux kernel scheduling traces.

URL: https://gitlab.inria.fr/schedgraph/schedgraph

Publication: hal-04001993v1

Author: Julia Lawall

Contact: Julia Lawall

6.1.4 Coccinelle for C++

Keywords: C++, Program transformation

Functional Description: Coccinelle is a tool for matching and transformation of C source code (https://coccinelle.gitlabpages.inria.fr/website/). Coccinelle for C++ is an extension of Coccinelle to handle C++ code.

URL: https://coccinelle.gitlabpages.inria.fr/website/

Publications: hal-01890314v1, hal-03266521v2

Authors: Julia Lawall, Michele Martone

Contact: Julia Lawall

7 New results

7.1 Finding Concurrency Bugs in the Linux Kernel

Participants: Baptiste Lepers *(University of Neuchâtel)*, Josselin Giet *(ENS/Inria)*, Julia Lawall *(Whisper)*, Willy Zwaenepoel *(University of Sydney)*.

Knowing which functions may execute concurrently is key to finding concurrency-related bugs. Existing tools infer the possibility of concurrency using dynamic analysis or by pairing functions that use the same locks. Code that relies on more relaxed concurrency controls is, by and large, out of the reach of such tools.

The paper "OFence: Pairing Barriers to Find Concurrency Bugs in the Linux Kernel" [11], by Baptiste Lepers, Josselin Giet, Julia Lawall, Willy Zwaenepoel, published at EuroSys 2023, proposes a new heuristic to automatically infer the possibility of concurrency in lockless code that relies on memory barriers (memory fences). Understanding the relationships between such barriers is made complex by the fact that barriers do not have a unique identifier and do not have a clearly delimited scope. To infer the possibility of concurrency between barriers, this paper proposes a novel heuristic based on matching objects accessed before and after barriers. This approach is based on the observation that barriers work in pairs: if a write memory barrier orders writes to a set of objects, then there should be a read barrier that orders reads to the same set of objects. This pairing strategy allows OFence to infer which barriers are meant to run concurrently and, in turn, check the code surrounding the barriers for concurrency-related bugs. As a example of such a bug, OFence focuses on bugs related to the incorrect placement of reads or writes relative to barriers. When it detects incorrect read or write placements in the code, OFence automatically produces a patch to fix them.

Using OFence, we have found and fixed 12 incorrect ordering constraints that could have resulted in hard-to-debug data corruption or kernel crashes. The patches have been merged in the mainline kernel. None of the bugs could have been found using existing static analysis heuristics.

The paper received a EuroSys 2023 Best Paper award.

7.2 IO performance in a virtualized setting

Participants: Damien Thenot *(Inria Benagil)*, Jean-Pierre Lozi *(Whisper)*, Gaël Thomas *(Inria Benagil)*.

Optimizing IO in a type I hypervisor such as Xen is difficult because of the cost of exchanging data between a VM and the driver. The paper "FastXenBlk: High-Performance Virtualized Disk IOs Without Compromising Isolation" [12], by Damien Thenot, Jean-Pierre Lozi, and Gaël Thomas, published at the Industry Track of Middleware 2023, addresses this challenge by proposing FastXenBlk, a new IO driver for Xen. FastXenBlk uses three mechanisms to improve IO performance. First, it uses several threads that poll multiple virtual IO queues that are exposed to a guest in order to execute IOs in parallel. Second, it batches requests in order to minimize the number of hypercalls to Xen. And third, it uses kernel bypass in order to avoid system calls during IOs. FastXenBlk is evaluated using the FIO benchmark with different access patterns and IO sizes. The evaluation shows that FastXenBlk consistently improves the latency and the throughput for all workloads as compared to tapdisk, the driver currently used in production, by a factor of up to 3×.

7.3 Trusted Execution Environments

Participants: Peterson Yuhala *(University of Neuchâtel)*, Pascal Felber *(University of Neuchâtel)*, Hugo Guiroux *(Oracl Labs)*, Jean-Pierre Lozi *(Whisper)*, Alain Tchana *(Grenoble INP)*, Valerio Schiavoni *(University of Neuchâtel)*, Gaël Thomas *(Inria Benagil)*.

Trusted execution environments like Intel SGX provide enclaves, which offer strong security guarantees for applications. Running entire applications inside enclaves is possible, but this approach leads to a large trusted computing base (TCB). As such, various tools have been developed to partition programs written in languages such as C or Java into trusted and untrusted parts, which are run in and out of enclaves, respectively. However, those tools depend on language-specific taint-analysis and partitioning techniques. They cannot be reused for other languages and there is thus a need for tools that transcend this language barrier. The paper "SecV: secure code partitioning via multi-language secure values" [13], by Peterson Yuhala, Pascal Felber, Hugo Guiroux, Jean-Pierre Lozi, Alain Tchana, Valerio Schiavoni, and Gaël Thomas, published at Middleware 2023, addresses this challenge by proposing a multi-language technique to specify sensitive code or data, as well as a multilanguage tool to analyse and partition the resulting programs for trusted execution environments like Intel SGX. The approach leverages GraalVM's Truffle framework, which provides a language-agnostic abstract syntax tree (AST) representation for programs, to provide special AST nodes called secure nodes that encapsulate sensitive program information. Secure nodes can easily be embedded into the ASTs of a wide range of languages via Truffle's polyglot API.The technique includes a multi-language dynamic taint tracking tool to analyse and partition applications based on our generic secure nodes. An extensive evaluation with micro- and macro-benchmarks shows that the technique can be used for two languages (Javascript and Python), and that partitioned programs can obtain up to 14.5% performance improvement as compared to unpartitioned versions.

7.4 Graph query languages

Participants: Jean-Pierre Lozi *(Whisper)*, Tomáš Faltín *(Whisper)*.

Reachability queries checking the existence of a path from a source node to a target node are fundamental operators for querying and processing graph data. Current approaches for index-based evaluation of reachability queries either focus on plain reachability or constraint-based reachability with only alternation of labels. The paper "A Reachability Index for Recursive Label-Concatenated Graph Queries" [14], by Chao Zhang, Angela Bonifati, Hugo Kapp, Vlad Ioan Haprian, and Jean-Pierre Lozi, published at ICDE 2023, studies, for the first time, the problem of index-based processing for recursive label-concatenated

reachability queries, referred to as RLC queries. These queries check the existence of a path that can satisfy the constraint defined by a concatenation of at most k edge labels under the Kleene plus. Many practical graph database and network analysis applications exhibit RLC queries. However, their evaluation remains prohibitive in current graph database engines. This paper introduces the RLC index, the first reachability index to efficiently process RLC queries. The RLC index checks whether the source vertex can reach an intermediate vertex that can also reach the target vertex under a recursive label-concatenated constraint. The paper proposes an indexing algorithm to build the RLC index, which guarantees the soundness and the completeness of query execution and avoids recording redundant index entries. Comprehensive experiments on real-world graphs show that the RLC index can significantly reduce both the offline processing cost and the memory overhead of transitive closure, while improving query processing up to six orders of magnitude over online traversals. Finally, the open-source implementation of the RLC index significantly outperforms current mainstream graph engines for evaluating RLC queries.

Graph engines play a crucial role in modern data analytics pipelines, serving as a middleware for handling complex queries across various domains, such as financial fraud detection. Graph queries enable flexible exploration and analysis, akin to SQL in relational databases. Among the most expressive and powerful constructs of graph querying are regular path queries (RPQs). RPQs enable support for variable-length path patterns based on regular expressions, such as (p1:Person)-/:Knows+/->(p2:Person), which searches for non-empty paths of any length between two persons. The paper "Distributed Asynchronous Regular Path Queries (RPQs) on Graphs" [10], by Tomáš Faltín, Vasileios Trigonakis, Ayoub Berdai, Luigi Fusco, Călin Iorgulescu, Jinsoo Lee, Jakub Yaghob, Sungpack Hong, and Hassan Chafi, published at the Middleware 2023 industry track, introduces a novel design for distributed RPQs that builds on top of distributed asynchronous pipelined traversals to enable (i) memory control of path explorations, with (ii) great performance and scalability. They show that with sixteen machines, their approach outperforms Neo4j by 91× on average and a relational implementation of the same queries in PostgreSQL by 230×, while maintaining low memory consumption.

8 Bilateral contracts and grants with industry

8.1 Bilateral grants with industry

Oracle, 2022-2023, 100 000 dollar gift.

Participants: Julia Lawall *(Whisper)*, Jean-Pierre Lozi *(Whisper)*, Himadri Chhaya-Shailesh, Calin Iorgulescu *(Oracle Labs)*.

This donation is the third and final in a series of donations from Oracle to support research on kernel-level task schedulers in the Whisper team. Our main work in this area has centered around the PhD of Himadri Chhaya-Shailesh on identifying and resolving bottlenecks in application performance when running in virtual machines, due to the dual level of scheduling between the guest and the host operating systems. Specifically, we are investigating how to address the problem of "phantom" vCPUs, that appear to the guest operating system to be available CPUs but that in reality have been preempted by the host operating system, thus causing the application execution to stall.

9 Partnerships and cooperations

9.1 International initiatives

9.1.1 Participation in other International Programs

Participants: Julia Lawall *(Whisper)*, Michele Martone *(Leibniz Supercomputing Center)*.

This is a BayFrance collaboration grant (5000 euros), managed by the Leibniz Supercomputing Center, permitting trips between Paris and Munich in 2023-2024. The subject of the collaboration is the extension of Coccinelle to C++ to enable the transformation of HPC code, particularly focusing on transformations to make such code run effectively on GPUs.

9.2 International research visitors

9.2.1 Visits of international scientists

Other international visits to the team

Tathagata Roy

Status intern

Institution of origin: IIIT Kalyani

Country: India

Dates: June, July, December 2023

Context of the visit: Design and development of Coccinelle for Rust.

Mobility program/type of mobility: research stay

9.2.2 Visits to international teams

Research stays abroad

- Jean-Pierre Lozi visited the team of Willy Zwaenepoel of the University of Sydney for 2 weeks (February 17 - March 3, 2023). This visit was supported by the Inria associate team CSG.
- Julia Lawall visited Michele Martone of the Leibniz Supercomputing Centre in Garching bei München, Germany for one week (October 15-21). This visit was supported by a BayFrance collaboration grant (Section 9.1.1).

9.3 National initiatives

9.3.1 ANR

VeriAmos

Participants: Xavier Rival *(Antique (PI))*, Nicolas Palix *(UGA (Erods))*, Gilles Muller *(Whisper)*, Julia Lawall *(Whisper)*, Keisuke Nishimura *(Whisper)*.

- Awarded in 2018, duration 2018 - 2022 (extended through June 2024)
- Members: Inria (Antique, Whisper), UGA (Erods)
- Funding: ANR, 121,739 euros.
- Objectives:

  General-purpose Operating Systems, such as Linux, are increasingly used to support high-level functionalities in the safety-critical embedded systems industry with usage in automotive, medical and cyber-physical systems. However, it is well known that general purpose OSes suffer from bugs. In the embedded systems context, bugs may have critical consequences, even affecting human life. Recently, some major advances have been done in verifying OS kernels, mostly employing interactive theorem-proving techniques. These works rely on the formalization of the programming

language semantics, and of the implementation of a software component, but require significant human intervention to supply the main proof arguments. The VeriAmos project is attacking this problem by building on recent advances in the design of domain-specific languages and static analyzers for systems code. We are investigating whether the restricted expressiveness and the higher level of abstraction provided by the use of a DSL will make it possible to design static analyzers that can statically and fully automatically verify important classes of semantic properties on OS code, while retaining adequate performance of the OS service. As a specific use-case, the project targets I/O scheduling components.

EMASS

Participants: Matthieu Lemerre *(CEA, EMASS PI)*, Sébastien Bardin *(CEA)*, Frédéric Recoules *(CEA)*, Xavier Rival *(Antique (Inria PI))*, Julia Lawall *(Whisper)*, Keisuke Nishimura *(Whisper)*, Kosmatov Nikolai *(Thales RT)*, Delphine Longuet *(Thales RT)*, Romain Soulat *(Thales RT)*.

- Awarded in 2023, duration 42 months (2023 - 2026).
- Members: Inria (Antique, Whisper), CEA, Thales RT
- Funding: ANR, 162,720 euros awarded to Inria.
- Objectives: Current systems programs, like OS kernels, hypervisors, or core libraries found in the bottom layer of every application stack, are today mostly written in systems programming languages, such as C, C++, or assembly, that give programmers low-level control over resource management at the expense of safety. This leads to frequent memory corruption errors in systems programs, which is one of the major cybersecurity issues today. Backed by the consortium's strong experience in advanced memory analyses (shape analyses), source and binary-level program analysis for security, software engineering by scalable static analysis on industrial case studies, and OS kernel verification, we want to provide an effective solution to this problem, even for the most challenging systems software, i.e., OS kernels and hypervisors.

  The EMASS project targets a solution to this problem relying on a static analysis of memory, using types as a base abstraction. Our goal is to be able to automatically verify the most challenging systems code, by developing a scalable compositional analysis, and by tackling other important security properties such as non-interference between the tasks of the OS, or functional contracts on the OS system calls, with the guidance of a low amount of intuitive type annotations. The project will result in strong scientific results in type-based static analysis, and in the EMASS toolkit, a pragmatic open-source solution for verifying and certifying secure systems software without formal methods expertise, whose practical use will be validated in challenging industrial case studies.

  The contribution of the Whisper team will focus on using our tools such as Coccinelle [2] and Prequel [3] to find challenging case studies, and on developing strategies for continuous verification of systems software.

DiVa: DIsaggregated VirtuAlization

Participants: Gaël Thomas *(Inria Benagil (DiVa PI))*, Mathieu Bacou *(Inria Benagil)*, Vivien Quema *(Grenoble INP)*, Jean-Pierre Lozi *(Whisper)*, Julia Lawall *(Whisper)*, Alain Tchana *(Grenoble INP)*, Daniel Hagimont *(INP Toulouse)*, Boris Teabe *(INP Toulouse)*, Julien Sopena *(Sorbonne University)*.

- Awarded in 2023, duration 7 years (2023-2030)

- Members: Inria (Benagil, Whisper), Grenoble INP, INP Toulouse, Sorbonne University
- Funding: ANR-PEPR, 321,839 euros.
- Objectives: Cloud infrastructures are currently undergoing major changes with the advent of disaggregated and edge clouds. The objective of the DiVA (DIsaggregated VirtuAlization) project is to prepare the system stack for these next generations of cloud infrastructures. In order to use these infrastructures more efficiently and to avoid wasting hardware resources, we propose to revisit the system mechanisms at the basis of any cloud infrastructure.

  In the context of disaggregated infrastructures, it becomes possible, at the scale of a few clusters of machines, to fully mutualize hardware resources. In detail, using the recent CXL protocol, any processor on any machine in the cluster can transparently access any hardware resource (memory, network, disk, accelerator) on any other server. This evolution calls for new hypervisors and operating system designs because virtual machines will become elastic (i.e., it will be possible to add new resources to a virtual machine at runtime) and distributed (i.e., the hardware resources used by a virtual machine are physically distributed). In the DiVA project, we will therefore (i) define new virtualization interfaces in order to allow a virtual machine to dynamically allocate or release hardware resources, (ii) study new scheduling and placement mechanisms in the guest operating system in order to efficiently use these distributed resources, (iii) study how we could use programmable networks to optimize the performance of virtual machines, and (iv) study how we could design transparent replication mechanisms since, in this distributed context, faults are no longer an exception but become the norm.

  In the context of edge infrastructures, small clusters of machines are connected by slow networks to powerful data centers. The small clusters perform computations close to the data sources, which avoids expensive data transfers, while having the possibility to use the computing power of a data center. This evolution requires revisiting several system mechanisms to support greater heterogeneity in terms of hardware and software architecture, and to support slow edge/core cloud network communication. In the DiVA project, we will therefore (i) design new virtual machine migration mechanisms in order to migrate virtual machines between heterogeneous machines (instruction set and hypervisor), (ii) transparently optimize the data paths of multi-tier applications distributed between the cloud and the edge in order to minimize the cloud/edge network links, and (iii) define new virtualization interfaces to optimize micro virtual machines with a short lifetime.

  In the context of DiVa, Jean-Pierre Lozi is leading a task on rack-scale scheduling, with the goal of making it possible for the scheduler to transparently migrate tasks across machines, while preserving access to open files and network connections.

10 Dissemination

10.1 Promoting scientific activities

10.1.1 Scientific events: organisation

Member of the organizing committees

- Jean-Pierre Lozi was a sponsorship chair of EuroSys 2023.

10.1.2 Scientific events: selection

Chair of conference program committees

- Julia Lawall was a program chair of the 2023 USENIX Annual Technical Conference (ATC).

Member of the conference program committees

- Julia Lawall was a member of the program committee of the following conferences that took place in 2023: GPCE, ICPC, ICSME, ICSE, TFP.

- Julia Lawall was a member of the program committee of the following workshops that took place in 2023: DIMES, Scheme, ML, ASE Journal First, EuroSys Doctoral Wokshop; ESEC-FSE ideas, visions, and reflections.
- Jean-Pierre Lozi was a member of the program committee of the following conferences that took place in 2023: USENIX ATC, EuroSys.

10.1.3 Journal

Member of the editorial boards

- Julia Lawall is a member of editorial board of Science of Computer Programming.

Reviewer - reviewing activities

- Jean-Pierre Lozi did a review for ACM Transactions on Computer Systems (TOCS).

10.1.4 Invited talks

- Julia Lawall gave an invited keynote talk at the Journées nationales of the GDR Security.
- Julia Lawall gave an invited keynote talk at ASE 2023 on "Automating Software Evolution in the Linux Kernel".
- Julia Lawall gave an invited keynote talk at GPCE 2023 on "Coccinelle: Impact and Internals".
- Jean-Pierre Lozi gave a talk at the University of Sydney on "Published and ongoing works in the Whisper team".

10.1.5 Leadership within the scientific community

- Julia Lawall is a member of the co-pilotage of IFIP France.
- Julia Lawall is the secretary of IFIP TC2 (Software: Theory and Practice) and is the French representative to IFIP TC2.
- Julia Lawall is on the advisory board of Software Heritage.

10.2 Teaching - Supervision - Juries

10.2.1 Supervision

- Julia Lawall and Jean-Pierre Lozi co-supervise the PhD of Himadri Chhaya-Shailesh at Inria Paris.
- Jean-Pierre Lozi and Julia Lawall co-supervise the PhD of Victor Laforet at Inria Paris.
- Alain Tchana, Renaud Lachaize, and Jean-Pierre Lozi co-supervise the PhD of Papa Assane Fall at the University of Grenoble (Erods).
- David Bromberg, Djob Mvondo and Julia Lawall co-supervise the PhD of Cesaire Honoré at the University of Rennes (Inria WIDE).
- David Bromberg, Djob Mvondo and Julia Lawall co-supervise the PhD of Amélie Gonzalez at the University of Rennes (Inria WIDE).
- Jean-Pierre Lozi co-supervised the PhD of Tomáš Faltín at Charles University (completed December 2023).

10.2.2 Juries

- Jean-Pierre Lozi was a member of the PhD jury of Yohan Pipereau (Telecom Sud-Paris).
- Julia Lawall was a member of the PhD jury of Florian Vanhems at the University of Lille.

10.3 Popularization

10.3.1 Interventions

- Julia Lawall gave a talk on "Graphing Tools for Scheduler Tracing" at the Kernel devroom of (FOSDEM 2023).
- Julia Lawall gave a talk on "Graphing Tools for Scheduler Tracing" at (Linux Lund 2023).
- Julia Lawall gave a talk on "Coccinelle for Rust" at (Kangrejos 2023).
- Julia Lawall gave a talk on "Graphing Tools for Scheduler Tracing" at the Tracing Microconference at the (2023 Linux Plumbers Conference).
- Julia Lawall gave a talk on "Coccinelle for Rust" at the Rust Microconference at the (2023 Linux Plumbers Conference).

11 Scientific production

11.1 Major publications

[1] J. Brunel, D. Doligez, R. R. Hansen, J. L. Lawall and G. Muller. 'A foundation for flow-based program matching: using temporal logic and model checking'. In: *The 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, POPL'09*. The 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, POPL'09. Savannah, Georgia, United States: ACM, Jan. 2009, pp. 114–126. DOI: 10.1145/1480881.1480897. URL: https://hal.science/hal-01299040.

[2] J. Lawall and G. Muller. 'Coccinelle: 10 Years of Automated Evolution in the Linux Kernel'. In: 2018 USENIX Annual Technical Conference. Boston, MA, United States, 11th July 2018. URL: https://inria.hal.science/hal-01853271.

[3] J. Lawall, D. Palinski, L. Gnirke and G. Muller. 'Fast and Precise Retrieval of Forward and Back Porting Information for Linux Device Drivers'. In: 2017 USENIX Annual Technical Conference. Santa Clara, CA, United States, 12th July 2017, p. 12. URL: https://inria.hal.science/hal-01556589.

[4] B. Lepers, R. Gouicem, D. Carver, J.-P. Lozi, N. Palix, M.-V. Aponte, W. Zwaenepoel, J. Sopena, J. Lawall and G. Muller. 'Provable Multicore Schedulers with Ipanema: Application to Work Conservation'. In: Eurosys 2020 - European Conference on Computer Systems. Heraklion / Virtual, Greece, 27th Apr. 2020. DOI: 10.1145/3342195.3387544. URL: https://hal.inria.fr/hal-02554342.

[5] J.-P. Lozi, F. David, G. Thomas, J. L. Lawall and G. Muller. 'Remote Core Locking: Migrating Critical-Section Execution to Improve the Performance of Multithreaded Applications'. In: *Proceedings of the 2012 USENIX Annual Technical Conference (USENIX ATC '12)*. 2012 USENIX Annual Technical Conference (USENIX ATC '12). Boston, MA, United States: ACM, 2012. DOI: 10.5555/2342821.2342827. URL: https://inria.hal.science/hal-00779908.

[6] G. Muller, J. L. Lawall and H. Duchesne. 'A Framework for Simplifying the Development of Kernel Schedulers: Design and Performance Evaluation'. In: *HASE - High Assurance Systems Engineering Conference*. Heidelberg, Germany: IEEE, Oct. 2005, pp. 56–65.

[7] Y. Padioleau, J. L. Lawall, R. R. Hansen and G. Muller. 'Documenting and Automating Collateral Evolutions in Linux Device Drivers'. In: *EuroSys*. Glasgow, Scotland, Mar. 2008, pp. 247–260.

[8] N. Palix, G. Thomas, S. Saha, C. Calvès, J. L. Lawall and G. Muller. 'Faults in Linux 2.6'. In: *ACM Transactions on Computer Systems* 32.2 (June 2014), 4:1–4:40.

[9] L. Serrano, V.-A. Nguyen, F. Thung, L. Jiang, D. Lo, J. Lawall and G. Muller. 'SPINFER: Inferring Semantic Patches for the Linux Kernel'. In: USENIX Annual Technical Conference. Boston / Virtual, United States, 15th July 2020. URL: https://hal.inria.fr/hal-02906912.

11.2 Publications of the year

International peer-reviewed conferences

[10] T. Faltín, V. Trigonakis, A. Berdai, L. Fusco, C. Iorgulescu, J. Lee, J. Yaghob, S. Hong and H. Chafi. 'Distributed Asynchronous Regular Path Queries (RPQs) on Graphs'. In: *Middleware Industrial Track '23: Proceedings of the 24th International Middleware Conference Industrial Track*. Middleware 2023: 24th International Middleware Conference. Bologna, Italy, Italy: ACM, 11th Dec. 2023, pp. 35–41. DOI: 10.1145/3626562.3626833. URL: https://inria.hal.science/hal-04355309.

[11] B. Lepers, J. Giet, J. Lawall and W. Zwaenepoel. 'OFence: Pairing Barriers to Find Concurrency Bugs in the Linux Kernel'. In: EuroSys 2023 : Eighteenth European Conference on Computer Systems. Rome, Italy: ACM, 8th May 2023, pp. 33–45. DOI: 10.1145/3552326.3567504. URL: https://inria.hal.science/hal-04109096.

[12] D. Thenot, J.-P. Lozi and G. Thomas. 'FastXenBlk: High-Performance Virtualized Disk IOs Without Compromising Isolation'. In: *Middleware '23: Proceedings of the 24th International Middleware Conference: Industrial Track*. Middleware '23: 24th International Middleware Conference. Bologna, Italy: ACM, 11th Dec. 2023, pp. 42–48. DOI: 10.1145/3626562.3626834. URL: https://hal.science/hal-04354563.

[13] P. Yuhala, P. Felber, H. Guiroux, J.-P. Lozi, A. Tchana, V. Schiavoni and G. Thomas. 'SecV: Secure code partitioning via multi-language secure values'. In: *Middleware '23: Proceedings of the 24th International Middleware Conference*. Middleware '23: 24th International Middleware Conference. Bologna, Italy: ACM, 27th Nov. 2023, pp. 207–219. DOI: 10.1145/3590140.3629116. URL: https://inria.hal.science/hal-04355247.

[14] C. Zhang, A. Bonifati, H. Kapp, V. I. Haprian and J.-P. Lozi. 'A Reachability Index for Recursive Label-Concatenated Graph Queries'. In: *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. ICDE 2023 - 39th IEEE International Conference on Data Engineering. Anaheim, United States, 3rd Apr. 2023. URL: https://hal.science/hal-03905483.

Reports & preprints

[15] V. Laforet, J.-P. Lozi and J. Lawall. *BPF Hybrid Lock: Using eBPF to communicate with the scheduler*. Inria; Institut Polytechnique de Paris, 8th Sept. 2023. URL: https://inria.hal.science/hal-04266815.

[16] J. Lawall, H. Chhaya-Shailesh, J.-P. Lozi and G. Muller. *Graphing Tools for Scheduler Tracing*. RR-9498. Inria Paris, 23rd Feb. 2023. URL: https://inria.hal.science/hal-04001993.

Other scientific publications

[17] V. Laforet, J.-P. Lozi and J. Lawall. 'eBPF Hybrid Lock: Scalable Spin-based User-Space Locking (Poster)'. In: SOSP 2023 - The 29th ACM Symposium on Operating Systems Principles. Koblenz, Germany, 23rd Oct. 2023. URL: https://inria.hal.science/hal-04262326.

11.3 Cited publications

[18] T. Ball, E. Bounimova, B. Cook, V. Levin, J. Lichtenberg, C. McGarvey, B. Ondrusek, S. K. Rajamani and A. Ustuner. 'Thorough Static Analysis of Device Drivers'. In: *EuroSys*. 2006, pp. 73–85.

[19] V. Chipounov and G. Candea. 'Reverse Engineering of Binary Device Drivers with RevNIC'. In: *EuroSys*. 2010, pp. 167–180.

[20] J. Corbet. 'The extensible scheduler class'. In: *Linux Weekly News* (Feb. 2023). URL: https://lwn.net/Articles/922405/.

[21] P. Cuoq, F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles and B. Yakobowski. 'Frama-C: A Software Analysis Perspective'. In: *Software Engineering and Formal Methods - 10th International Conference (SEFM)*. Thessaloniki, Greece, Oct. 2012. URL: http://pathcrawler-online.com/pubs/final_sefm12.pdf.

[22] T. David, R. Guerraoui and V. Trigonakis. 'Everything you always wanted to know about synchronization but were afraid to ask'. In: *Symposium on Operating Systems Principles (SOSP)*. 2013, pp. 33–48.

[23] I. Dillig, T. Dillig and A. Aiken. 'Sound, complete and scalable path-sensitive analysis'. In: *PLDI*. June 2008, pp. 270–280.

[24] D. R. Engler, B. Chelf, A. Chou and S. Hallem. 'Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions'. In: *OSDI*. 2000, pp. 1–16.

[25] H. Guiroux, R. Lachaize and V. Quéma. 'Multicore Locks: The Case Is Not Closed Yet'. In: *2016 USENIX Annual Technical Conference, USENIX ATC 2016, Denver, CO, USA, June 22-24, 2016*. Ed. by A. Gulati and H. Weatherspoon. USENIX Association, 2016, pp. 649–662.

[26] L. Guo, J. Lawall and G. Muller. 'Oops! Where Did That Code Snippet Come From?' In: *MSR 2014 - 11th Working Conference on Mining Software Repositories*. Hyderabad, India: ACM, May 2014, pp. 52–61. DOI: 10.1145/2597073.2597094. URL: https://inria.hal.science/hal-01080397.

[27] J. T. Humphries, N. Natu, A. Chaugule, O. Weisse, B. Rhoden, J. Don, L. Rizzo, O. Rombakh, P. Turner and C. Kozyrakis. 'GhOSt: Fast & Flexible User-Space Delegation of Linux Scheduling'. In: *Symposium on Operating Systems Principles*. Association for Computing Machinery, 2021, pp. 588–604.

[28] A. Israeli and D. G. Feitelson. 'The Linux kernel as a case study in software evolution'. In: *Journal of Systems and Software* 83.3 (2010), pp. 485–501.

[29] A. Kadav and M. M. Swift. 'Understanding modern device drivers'. In: *ASPLOS*. 2012, pp. 87–98.

[30] J. Koschel, P. Borrello, D. C. D'Elia, H. Bos and C. Giuffrida. 'Uncontained: Uncovering Container Confusion in the Linux Kernel'. In: *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 5055–5072. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/koschel.

[31] J. Lawall, J. Brunel, N. Palix, R. R. Hansen, H. Stuart and G. Muller. 'WYSIWIB: exploiting fine-grained program structure in a scriptable API-usage protocol-finding process'. In: *Software: Practice and Experience* 43.1 (Jan. 2013), pp. 67–92. DOI: 10.1002/spe.2102. URL: https://hal.science/hal-00940320.

[32] J. Lawall, B. Laurie, R. R. Hansen, N. Palix and G. Muller. 'Finding Error Handling Bugs in OpenSSL Using Coccinelle'. In: *European Dependable Computing Conference*. Valencia, Spain, Apr. 2010, pp. 191–196. DOI: 10.1109/EDCC.2010.31. URL: https://hal.science/hal-00940375.

[33] T. Li, J. Bai, Y. Sui and S. Hu. 'Path-sensitive and alias-aware typestate analysis for detecting OS bugs'. In: *ASPLOS*. ACM, 2022, pp. 859–872.

[34] Z. Li, S. Lu, S. Myagmar and Y. Zhou. 'CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code'. In: *OSDI*. 2004, pp. 289–302.

[35] Z. Li and Y. Zhou. 'PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code'. In: *Proceedings of the 10th European Software Engineering Conference*. 2005, pp. 306–315.

[36] F. Mérillon, L. Réveillère, C. Consel, R. Marlet and G. Muller. 'Devil: An IDL for Hardware Programming'. In: *4th Symposium on Operating System Design and Implementation (OSDI)*. 2000, pp. 17–30.

[37] J. Oh, N. F. Yildiran, J. Braha and P. Gazzillo. 'Finding broken Linux configuration specifications by statically analyzing the Kconfig language'. In: *ESEC/FSE*. Ed. by D. Spinellis, G. Gousios, M. Chechik and M. D. Penta. ACM, 2021, pp. 893–905.

[38] L. R. Rodriguez and J. L. Lawall. 'Increasing Automation in the Backporting of Linux Drivers Using Coccinelle'. In: *11th European Dependable Computing Conference - Dependability in Practice*. 11th European Dependable Computing Conference - Dependability in Practice. Paris, France, Nov. 2015. URL: https://hal.inria.fr/hal-01213912.

[39] L. Ryzhyk, P. Chubb, I. Kuz, E. Le Sueur and G. Heiser. 'Automatic device driver synthesis with Termite'. In: *SOSP*. 2009, pp. 73–86.

[40] L. Ryzhyk, A. Walker, J. Keys, A. Legg, A. Raghunath, M. Stumm and M. Vij. 'User-Guided Device Driver Synthesis'. In: *OSDI*. 2014, pp. 661–676.

[41] R. K. Saha, J. L. Lawall, S. Khurshid and D. E. Perry. 'On the Effectiveness of Information Retrieval Based Bug Localization for C Programs'. In: *ICSME 2014 - 30th International Conference on Software Maintenance and Evolution*. IEEE. Victoria, Canada, Sept. 2014, pp. 161–170. DOI: 10.1109/ICSME.2014.38. URL: https://inria.hal.science/hal-01086082.

[42] F. Thung, D. X. B. Le, D. Lo and J. L. Lawall. 'Recommending Code Changes for Automatic Backporting of Linux Device Drivers'. In: *32nd IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. Raleigh, North Carolina, United States, Oct. 2016. URL: https://hal.inria.fr/hal-01355859.

[43] W. Wang and M. Godfrey. 'A Study of Cloning in the Linux SCSI Drivers'. In: *Source Code Analysis and Manipulation (SCAM)*. IEEE, 2011.