RESEARCH CENTRE

Paris

IN PARTNERSHIP WITH:
CNRS, Sorbonne Université (UPMC)

2021
ACTIVITY REPORT

Project-Team

WHISPER

Well Honed Infrastructure Software for Programming Environments and Runtimes

IN COLLABORATION WITH: Laboratoire d'informatique de Paris 6 (LIP6)

DOMAIN

Networks, Systems and Services, Distributed Computing

THEME

Distributed Systems and middleware

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

A2.3.1. – Embedded systems

A2.3.3. – Real-time systems

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

B6.3.3. – Network Management

B6.5. – Information systems

B6.6. – Embedded systems

1 Team members, visitors, external collaborators

Research Scientists

- Gilles Muller [Team leader, Inria, Senior Researcher, HDR]
- Julia Lawall [Team leader, Inria, Senior Researcher, HDR]

PhD Students

- Yoann Ghigoff [Orange Labs]
- Josselin Giet [École Normale Supérieure de Paris]
- Pierre Nigron [Inria]

Technical Staff

- Jeremie Dautheribes [Inria, Engineer, until May 2021]
- Corentin De Souza [Inria, Engineer, until Jul 2021]
- Rehan Malak [Inria, Engineer, until Nov 2021]
- Thierry Martinez [Inria, Engineer]
- Himadri Pandya [Inria, Engineer, until Nov 2021, PhD student from Dec 2021]

Administrative Assistants

- Christine Anocq [Inria]
- Nelly Maloisel [Inria]

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 include virtual machine hypervisors, operating systems, managed runtime environments, standard libraries, and browsers, which amount to the new operating system layer for Internet applications. 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. Since computing now pervades our society, with few paper backup solutions, correctness of software at all levels is critical. Formal methods are increasingly being applied to operating systems code in the research community [32, 37, 64]. Still, such efforts require a huge amount of manpower and a high degree of expertise which makes this work difficult to replicate in standard infrastructure-software development.

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 aim at providing solutions that can be easily learned and adopted by system developers in the short term. Such solutions can be tools, such as Coccinelle [1, 7, 8] for transforming C programs, or domain-specific languages such as Devil [4], Bossa [6] and Ipanema [3] for designing drivers and kernel schedulers. Due to the small size of the team, Whisper mainly targets operating system kernels and runtimes for programming languages. We put an emphasis on achieving measurable improvements in performance and safety in practice, and on feeding these improvements back to the infrastructure software developer community.

3 Research program

3.1 Program analysis

A fundamental goal of the research in the Whisper team is to elicit and exploit the knowledge found in existing code. To do this in a way that scales to a large code base, systematic methods are needed to infer code properties. We may build on either static [27, 28, 29] or dynamic analysis [45, 47, 50]. Static analysis consists of approximating the behavior of the source code from the source code alone, while dynamic analysis draws conclusions from observations of sample executions, typically of test cases. While dynamic analysis can be more accurate, because it has access to information about actual program behavior, obtaining adequate test cases is difficult. This difficulty is compounded for infrastructure software, where many, often obscure, cases must be handled, and external effects such as timing can have a significant impact. Thus, we expect to primarily use static analyses. Static analyses come in a range of flavors, varying in the extent to which the analysis is *sound*, *i.e.*, the extent to which the results are guaranteed to reflect possible run-time behaviors.

One form of sound static analysis is *abstract interpretation* [28]. In abstract interpretation, atomic terms are interpreted as sound abstractions of their values, and operators are interpreted as functions that soundly manipulate these abstract values. The analysis is then performed by interpreting the program in a compositional manner using these abstracted values and operators. Alternatively, *dataflow analysis* [36] iteratively infers connections between variable definitions and uses, in terms of local transition rules that describe how various kinds of program constructs may impact variable values. Schmidt has explored the relationship between abstract interpretation and dataflow analysis [58]. More recently, more general forms of symbolic execution [27] have emerged as a means of understanding complex code. In symbolic execution, concrete values are used when available, and these are complemented by constraints that are inferred from terms for which only partial information is available. Reasoning about these constraints is then used to prune infeasible paths, and obtain more precise results. A number of works apply symbolic execution to operating systems code [24, 25].

While sound approaches are guaranteed to give correct results, they typically do not scale to the very diverse code bases that are prevalent in infrastructure software. An important insight of Engler et al. [31] was that valuable information could be obtained even when sacrificing soundness, and that sacrificing soundness could make it possible to treat software at the scales of the kernels of the Linux or BSD operating systems. Indeed, for certain types of problems, on certain code bases, that may mostly follow certain coding conventions, it may mostly be safe to *e.g.*, ignore the effects of aliases, assume that variable values are unchanged by calls to unanalyzed functions, etc. Real code has to be understood by developers and thus cannot be too complicated, so such simplifying assumptions are likely to hold in practice. Nevertheless, approaches that sacrifice soundness also require the user to manually validate the results. Still, it is likely to be much more efficient for the user to perform a potentially complex manual analysis in a specific case, rather than to implement all possible required analyses and apply them everywhere in the code base. A refinement of unsound analysis is the CEGAR approach [26], in which a highly approximate analysis is complemented by a sound analysis that checks the individual reports of the approximate analysis, and then any errors in reasoning detected by the sound analysis are used to refine the approximate analysis. The CEGAR approach has been applied effectively on device driver code in tools developed at Microsoft [21]. The environment in which the driver executes, however, is still represented by possibly unsound approximations.

Going further in the direction of sacrificing soundness for scalability, the software engineering community has recently explored a number of approaches to code understanding based on techniques developed in the areas of natural language understanding, data mining, and information retrieval. These approaches view code, as well as other software-reated artifacts, such as documentation and postings on mailing lists, as bags of words structured in various ways. Statistical methods are then used to collect words or phrases that seem to be highly correlated, independently of the semantics of the program constructs that connect them. The obliviousness to program semantics can lead to many false positives (invalid conclusions) [42], but can also highlight trends that are not apparent at the low level of individual program statements. We have previously explored combining such statistical methods with more traditional static analysis in identifying faults in the usage of constants in Linux kernel code [40].

3.2 Domain Specific Languages

Writing low-level infrastructure code is tedious and difficult, and verifying it is even more so. To produce non-trivial programs, we could benefit from moving up the abstraction stack to enable both programming and proving as quickly as possible. Domain-specific languages (DSLs), also known as *little languages*, are a means to that end [5] [48].

Traditional approach. Using little languages to aid in software development is a tried-and-trusted technique [60] by which programmers can express high-level ideas about the system at hand and avoid writing large quantities of formulaic C boilerplate.

This approach is typified by the Devil language for hardware access [4]. An OS programmer describes the register set of a hardware device in the high-level Devil language, which is then compiled into a library providing C functions to read and write values from the device registers. In doing so, Devil frees the programmer from having to write extensive bit-manipulation macros or inline functions to map between the values the OS code deals with, and the bit-representation used by the hardware: Devil generates code to do this automatically.

However, DSLs are not restricted to being “stub” compilers from declarative specifications. The Bossa language [6] is a prime example of a DSL involving imperative code (syntactically close to C) while offering a high-level of abstraction. This design of Bossa enables the developer to implement new process scheduling policies at a level of abstraction tailored to the application domain.

Conceptually, a DSL both abstracts away low-level details and justifies the abstraction by its semantics. In principle, it reduces development time by allowing the programmer to focus on high-level abstractions. The programmer needs to write less code, in a language with syntax and type checks adapted to the problem at hand, thus reducing the likelihood of errors.

Certifying DSLs. While automated and interactive software verification tools are progressively being applied to larger and larger programs, we have not yet reached the point where large-scale, legacy software – such as the Linux kernel – could formally be proved “correct”. DSLs enable a pragmatic approach, by which one could realistically strengthen a large legacy software by first narrowing down its critical component(s) and then focus verification efforts onto these components.

3.3 Research direction: Tools for improving legacy infrastructure software

A cornerstone of our work on legacy infrastructure software is the Coccinelle program matching and transformation tool for C code. Coccinelle has been in continuous development since 2005. Today, Coccinelle is extensively used in the context of Linux kernel development, as well as in the development of other software, such as wine, python, kvm, git, and systemd. Currently, Coccinelle is a mature software project, and no research is being conducted on Coccinelle itself. Instead, we leverage Coccinelle in other research projects [22, 23, 49, 51, 56, 57, 59, 46, 41], both for code exploration, to better understand at a large scale problems in Linux development, and as an essential component in tools that require program matching and transformation. The continuing development and use of Coccinelle is also a source of visibility in the Linux kernel developer community. We submitted the first patches to the Linux kernel based on Coccinelle in 2007. Since then, almost 9000 patches have been accepted into the Linux kernel based on the use of Coccinelle, including thousands by over 400 developers from outside our research group.

Our recent work has focused on driver porting. Specifically, we have considered the problem of porting a Linux device driver across versions, particularly backporting, in which a modern driver needs to be used by a client who, typically for reasons of stability, is not able to update their Linux kernel to the most recent version. When multiple drivers need to be backported, they typically need many common changes, suggesting that Coccinelle could be applicable. Using Coccinelle, however, requires writing backporting transformation rules. In order to more fully automate the backporting (or symmetrically forward porting) process, these rules should be generated automatically. We have carried out a preliminary study in this direction with David Lo of Singapore Management University; this work, published at ICSME 2016 [62], is limited to a port from one version to the next one, in the case where the amount of change required is limited to a single line of code. Whisper has been awarded an ANR PRCI grant (completed in 2021) to collaborate with the group of David Lo on scaling up the rule inference process and proposing a fully automatic porting solution.

3.4 Research direction: developing infrastructure software using Domain Specific Languages

We wish to pursue a *declarative* approach to developing infrastructure software. Indeed, there exists a significant gap between the high-level objectives of these systems and their implementation in low-level, imperative programming languages. To bridge that gap, we propose an approach based on domain-specific languages (DSLs). By abstracting away boilerplate code, DSLs increase the productivity of systems programmers. By providing a more declarative language, DSLs reduce the complexity of code, thus the likelihood of bugs.

Traditionally, systems are built by accretion of several, independent DSLs. For example, one might use Devil [4] to interact with devices, Bossa [6] to implement the scheduling policies. However, much effort is duplicated in implementing the back-ends of the individual DSLs. Our long term goal is to design a unified framework for developing and composing DSLs. By providing a single conceptual framework, we hope to amortize the development cost of a myriad of DSLs through a principled approach to reusing and composing them.

Beyond the software engineering aspects, a unified platform brings us closer to the implementation of mechanically-verified DSLs. A key benefit would be to provide – by construction – a formal, mechanized semantics to the DSLs thus developed. Such a semantics would offer a foundation on which to base further verification efforts, while allowing interaction with non-verified code. We advocate a methodology based on incremental, piece-wise verification. While building fully-certified systems from the top-down is a worthwhile endeavor [37], we wish to explore a bottom-up approach by which one focuses first and foremost on crucial subsystems and their associated properties.

Our current work on DSLs focuses on the design of domain-specific languages for domains where there is a critical need for code correctness, and corresponding methodologies for proving properties of the run-time behavior of the system.

4 Application domains

4.1 Linux

Linux is an open-source operating system that is used in settings ranging from embedded systems to supercomputers. The most recent release of the Linux kernel, v5.15, comprises over 20 million lines of code, and supports 30 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. [30], numerous research tools have been applied to the Linux kernel, typically for finding bugs [29, 44, 52, 61] or for computing software metrics [34, 63]. In our work, we have studied generic C bugs in Linux code [8], bugs in function protocol usage [38, 39], issues related to the processing of bug reports [55] and crash dumps [33], and the problem of backporting [51, 62], 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, who was supervised by G. Muller, proposes Devil [4], a domain-specific language for describing the low-level interface of a device. Chipounov *et al.* propose RevNic, [25] a template-based approach for porting device drivers from one OS to another. Ryzhyk *et al.* propose Termite, [53, 54] 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 [35] 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 [21], Coverity [30], CP-Miner, [43] PR-Miner [44], 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 of existing drivers.

5 Social and environmental responsibility

5.1 Impact of research results

Environmental responsability The Whisper team is actively pursuing research on process scheduling for the Linux kernel. A current area of interest is concentrating threads on fewer cores in a multicore setting, in order to both reduce the execution time and to increase the number of cores that can enter a deep idle state, thus reducing energy consumption. A first work in this direction was published at USENIX ATC 2020. We are continuing to work in this area as part of our collaboration with Oracle (Section 9.1).

6 Highlights of the year

6.1 Awards

- Reda Gouicem's 2020 PhD thesis on "Thread Scheduling in Multi-core Operating Systems", supervised by Gilles Muller and Julien Sopena was recognized (accessit) as part of the Prix de Thèse du GDR RSD/ASF 2021 (link).

- Gilles Muller received the Senior researcher prize of the GDR RSD/ASF for 2021 (link)

7 New software and platforms

We continue to maintain our tools for searching and transforming C code and for searching in the development history of C code projects.

7.1 New software

7.1.1 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.

URL: `http://coccinelle.lip6.fr`

Contact: Julia Lawall

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

Partners: LIP6, IRILL

7.1.2 Prequel

Keywords: Code search, Git

Scientific Description: The commit history of a code base such as the Linux kernel is a gold mine of information on how evolutions should be made, how bugs should be fixed, etc. Nevertheless, the high volume of commits available and the rudimentary filtering tools provided mean that it is often necessary to wade through a lot of irrelevant information before finding example commits that can help with a specific software development problem. To address this issue, we propose Prequel (Patch Query Language), which brings the descriptive power of code matching to the problem of querying a commit history.

Functional Description: Prequel is a tool for searching for complex patterns in the commits of software managed using git.

URL: `http://prequel-pql.gforge.inria.fr/`

Contact: Julia Lawall

Participants: Gilles Muller, Julia Lawall

Partners: LIP6, IRILL

8 New results

8.1 Software engineering for infrastructure software

Our main work in this area has been on the problem of inferring transformation rules from change examples for the purpose of fully automating large-scale evolutions in infrastructure software. This work has been carried out in collaboration with David Lo and Lingxiao Jiang of Singapore Management University, in the context of the ANR ITrans project. We also have further explored the use of Coccinelle for HPC code, with Michele Martone of the Leibniz Supercomputing Centre.

8.1.1 Transformation rule inference

Participants: Julia Lawall, Gilles Muller, David Lo *(Singapore Management University)*, Lingxiao Jiang *(Singapore Management University)*, Abhik Roychoudhury *(National University of Singapore)*, Gregory Duck *(National University of Singapore)*.

This year marked the end of our ANR project ITrans (Section 10.2) in collaboration with David Lo and Lingxiao Jiang at Singapore Management University. To round out the project, we published some final work on transformation rule inference for Android and for Python software using machine-learning libraries. The paper "AndroEvolve: Automated Update for Android Deprecated-API Usages" [13], published in the ICSE 2021 demo track, utilizes data flow analysis to solve the problem of out-of-method-boundary variables and variable denormalization to remove temporary variables, in order to produce a more natural updated program. The paper "Characterization and Automatic Updates of Deprecated Machine-Learning API Usages" [14], published at ICSME 2021, presents a tool to automate deprecated machine-learning (ML) API usage updates. We first present an empirical study to better understand how updates of deprecated ML API usages in Python can be done. The study involves a dataset of 112 deprecated APIs from Scikit-Learn, TensorFlow, and PyTorch. Guided by the findings of our empirical study, we propose MLCatchUp, a tool to automate the updates of Python deprecated API usages, that automatically infers the API migration transformation through comparison of the deprecated and updated API signatures. These transformations are expressed in a Domain Specific Language (DSL). We evaluate MLCatchUp using a dataset containing 267 files with 551 API usages that we collected from public GitHub repositories. In our dataset, MLCatchUp can detect deprecated API usages with perfect accuracy, and update them correctly for 80.6% of the cases. We further improve the accuracy of MLCatchUp in performing updates by allowing it to accept an additional user input that specifies the transformation constraints in the DSL for context-dependent API migration. Using this addition, MLCatchUp can make correct updates for 90.7% of the cases. Finally, the paper "MLCatchUp: Automated Update of Deprecated Machine-Learning APIs in Python" [15], also at ICSME 2021, presents a demo of the MLCatchUp work.

Beyond the ITrans project, we have worked with Abhik Roychoudhury and Gregory Duck of the National University of Singapore on the problem of backporting bug fixes to older versions of the Linux kernel. Whenever a bug or vulnerability is detected in the Linux kernel, the kernel developers will endeavour to fix it by introducing a patch into the mainline version of the Linux kernel source tree. However, many users run older “stable” versions of Linux, meaning that the patch should also be “backported” to one or more of these older kernel versions. This process is error-prone and there is usually a long delay in publishing the backported patch. We have targeted this problem in the paper "Automated patch backporting in Linux (experience paper)" [18], published at ISSTA 2021. Based on an empirical study, we show that around 8% of all commits submitted to Linux mainline are backported to older versions, but often more than one month elapses before the backport is available. Hence, we propose a patch backporting technique that can automatically transfer patches from the mainline version of Linux into older stable versions. Our approach first synthesizes a partial transformation rule based on a Linux mainline patch. This rule can then be generalized by analysing the alignment between the mainline and target versions. The generalized rule is then applied to the target version to produce a backported patch. We have implemented our transformation technique in a tool called FixMorph and evaluated it on 350 Linux mainline patches. FixMorph correctly backports 75.1% of them. Compared to existing techniques, FixMorph improves both the precision and recall. Apart from automation of software maintenance tasks, patch backporting helps in reducing the exposure to known security vulnerabilities in stable versions of the Linux kernel.

Two years after it was accepted and made available in "early access", the paper "PatchNet: Hierarchical Deep Learning-Based Stable Patch Identification for the Linux Kernel" [10] has finally appeared in IEEE TSE.

8.1.2 Refactoring of HPC code

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

With Michele Martone of the Leibniz Supercomputing Centre in Munich, we have been investigating the application of Coccinelle to HPC code, specifically for converting from the Arrays of Structures to the Structures of Arrays representation. We published an overview of this work, targeting the Gadget software, simulating large-scale structure (galaxies and clusters) formation, in the paper "Refactoring for Performance with Semantic Patching: Case Study with Recipes" [16], presented at the C2PO 2021 workshop.

8.2 Scalability

Participants: Gilles Muller, Yoann Ghigoff, Kahina Lazri *(Orange Labs)*, Julien Sopena *(Delys)*.

In-memory key-value stores are critical components that help scale large internet services by providing low-latency access to popular data. Memcached, one of the most popular key-value stores, suffers from performance limitations inherent to the Linux networking stack and fails to achieve high performance when using high-speed network interfaces. While the Linux network stack can be bypassed using DPDK based solutions, such approaches require a complete redesign of the software stack and induce high CPU utilization even when client load is low.

To overcome these limitations, in the paper "BMC: Accelerating Memcached using Safe In-kernel Caching and Pre-stack Processing", presented at NSDI 2021 [12], we present BMC, an in-kernel cache for Memcached that serves requests before the execution of the standard network stack. Requests to the BMC cache are treated as part of the NIC interrupts, which allows performance to scale with the number of cores serving the NIC queues. To ensure safety, BMC is implemented using eBPF. Despite the safety constraints of eBPF, we show that it is possible to implement a complex cache service. Because BMC runs on commodity hardware and requires modification of neither the Linux kernel nor the Memcached application, it can be widely deployed on existing systems. BMC optimizes the processing time of Facebook-like small-size requests. On this target workload, our evaluations show that BMC improves throughput by up to 18x compared to the vanilla Memcached application and up to 6x compared to an optimized version of Memcached that uses the SO_REUSEPORT socket flag. In addition, our results also show that BMC has negligible overhead and does not deteriorate throughput when treating non-target workloads.

This work was done as part of the PhD of Yoann Ghigoff in collaboration with Orange Labs (Section 9.1).

8.3 Virtualization

Participants: Gilles Muller, Alain Tchana *(ENS Lyon)*, Antonio Barbalace *(University of Edinburgh)*.

Microservices. Nowadays, Cloud and Edge applications are deployed as a set of several small units communicating with each other - the microservice model. Moreover, each unit - a microservice, may be implemented as a virtual machine, container, function, etc., spanning the different Cloud and Edge service models including IaaS, PaaS, FaaS. A microservice is instantiated upon the reception of a request (e.g., an http packet or a trigger), and a rack-level or data-center-level scheduler decides the placement for such a unit of execution considering, for example, data locality and load balancing. Different units, as well as multiple instances of the same unit, may be running on a single server at the same time.

When multiple microservices are running on the same server, some may be busy-waiting - i.e., waiting for events (or requests) sent by other units. However, these "idle" units are consuming CPU time that could be used by other running units or cloud utility functions on the server (e.g., monitoring daemons). In a controlled experiment, we observe that units can spend up to 20% - 55% of their CPU time waiting, thus a great amount of CPU time is wasted; these values significantly grow when overcommitting CPU resources (i.e., units' CPU reservations exceed server CPU capacity), where we observe up to 69% – 75%. This is a result of the lack of information/context about what is running in each unit from the server CPU scheduler perspective.

In the vision paper "Tell me when you are sleepy and what may wake you up!" [17], published at SoCC 2021, we first provide evidence of the problem and discuss several research questions. Then, we propose an handful of solutions worth exploring that consist in revisiting hypervisor and host OS scheduler designs to reduce the CPU time wasted on idle units. Our proposal leverages the concepts of informed scheduling, and monitoring for internal and external events. Based on the aforementioned solutions, we propose our initial implementation on Linux/KVM.

Paging. Nested/Extended Page Table (EPT) is the current hardware solution for virtualizing memory in virtualized systems. It induces a significant performance overhead due to the 2D page walk it requires, thus incurring 24 memory accesses on a TLB miss (instead of the 4 memory accesses in a native system). This 2D page walk comes from the utilization of paging for managing virtual machine (VM) memory. In the paper "(No)Compromis: paging virtualization is not a fatality" [19], at VEE 2021, we show that paging is not necessary in the hypervisor. Our solution Compromis, a novel Memory Management Unit, uses direct segments for VM memory management combined with paging for VM's processes. This is the first time that a direct segment based solution is shown to be applicable to the entire VM memory while keeping applications unchanged. Relying on the 310 studied datacenter traces, the paper shows that it is possible to provision up to 99.99% of the VMs using a single memory segment. The paper presents a systematic methodology for implementing Compromis in the hardware, the hypervisor and the datacenter scheduler. Evaluation results show that Compromis outperforms the two popular memory virtualization solutions: shadow paging and EPT by up to 30% and 370% respectively.

Security. The vulnerability window of a hypervisor regarding a given security flaw is the time between the identification of the flaw and the integration of a correction/patch in the running hypervisor. Most vulnerability windows, regardless of severity, are long enough (several days) that attackers have time to perform exploits. Nevertheless, the number of critical vulnerabilities per year is low enough to allow an exceptional solution. In the paper "Mitigating vulnerability windows with hypervisor transplant" [20], presented at EuroSys 2021, we introduce hypervisor transplant, a solution for addressing the vulnerability window of critical flaws. It involves temporarily replacing the current datacenter hypervisor (e.g., Xen) which is subject to a critical security flaw, by a different hypervisor (e.g., KVM) which is not subject to the same vulnerability.

We build HyperTP, a generic framework that combines in a unified way two approaches: in-place server micro-reboot-based hypervisor transplant (noted InPlaceTP) and live VM migration-based hypervisor transplant (noted MigrationTP). We describe the implementation of HyperTP and its extension for transplanting Xen with KVM and vice versa. We also show that HyperTP is easy to integrate with the OpenStack cloud computing platform. Our evaluation results show that HyperTP delivers satisfactory performance: (1) MigrationTP takes the same time and impacts virtual machines (VMs) with the same performance degradation as normal live migration. (2) the downtime imposed by InPlaceTP on VMs is in the same order of magnitude (1.7 seconds for a VM with 1 vCPU and 1 GB of RAM) as in-place upgrade of homogeneous hypervisors based on server micro-reboot.

8.4 Real-time systems

Participants: Gilles Muller, Isabelle Puaut *(Inria Rennes)*.

Modern processors raise a challenge for WCET estimation, since detailed knowledge of the processor microarchitecture is not available. In the paper "WE-HML: hybrid WCET estimation using machine learning for architectures with caches", published at RTCSA 2021 [11], we propose a novel hybrid WCET estimation technique, WE-HML, in which the longest path is estimated using static techniques, whereas machine learning (ML) is used to determine the WCET of basic blocks. In contrast to existing literature using ML techniques for WCET estimation, WE-HML (i) operates on binary code for improved precision of learning, as compared to the related techniques operating at source code or intermediate code level; (ii) trains the ML algorithms on a large set of automatically generated programs for improved quality of learning; (iii) proposes a technique to take into account data caches. Experiments on an ARM Cortex-A53 processor show that for all benchmarks, WCET estimates obtained by WE-HML are larger than all possible execution times. Moreover, the cache modeling technique of WE-HML allows an improvement of 65 percent on average of WCET estimates compared to its cache-agnostic equivalent.

9 Bilateral contracts and grants with industry

9.1 Bilateral contracts with industry

Orange Labs, 2019-2022, 30 000 euros.

Participants: Gilles Muller, Yoann Ghigoff, Julien Sopena *(Delys)*, Kahina Lazri *(Orange Labs)*.

The purpose of this contract is to design application-specific proxies so as to speed up network services. The PhD of Yoann Ghigoff is supported by a CIFRE fellowship as part of this contract. Some results from this project are described in Section 8.2.

DGA-Inria, 2019-2022, 60 000 euros.

Participants: Pierre Nigron, Pierre-Evariste Dagand *(CNRS)*.

The purpose of this PhD grant is to develop a high-performance, certified packet processing system. The PhD of Pierre Nigron is supported by this grant.

9.2 Bilateral grants with industry

Oracle, 2020-2021, 100 000 dollar gift.

Participants: Gilles Muller, Julia Lawall, Jean-Pierre Lozi *(Oracle)*.

Operating system schedulers are often a performance bottleneck on multicore architectures because in order to scale, schedulers cannot make optimal decisions and instead have to rely on heuristics. Detecting that performance degradation comes from the scheduler level is extremely difficult because the issue has not been recognized until recently, and with traditional profilers, both the application and the scheduler can affect the monitored metrics in the same way.

The first objective of this project was to produce a profiler that makes it possible to find out whether a bottleneck during application runtime is caused by the application itself, by suboptimal OS scheduler behavior, or by a combination of the two. Such a profiler should enable understanding, analyzing and classifying performance bottlenecks that are caused by schedulers, to help the user understand the root cause of the performance issue. Following this, the second objective of this project is to use the profiler to better understand which kinds of workloads suffer from poor scheduling, and to propose new algorithms, heuristics and/or a new scheduler design that will improve the situation. Finally, the third objective is to devise a methodology that makes it possible to track scheduling bottlenecks in a specific workload using the profiler, to understand them, and to fix them either at the application or at the scheduler level. We believe that the combination of these three contributions will make it possible to fully harness the power of multicore architectures for any workload.

As part of this project, we have already identified frequency scaling and the “fork/wait” paradigm as a source of inefficiency in modern multicore machines. The design of a scheduler, Smove, to address this issue was published at USENIX ATC 2020. Subsequently, we have continued to study the impact of core frequency on application performance, using our developed scheduler tracing tools.

10 Partnerships and cooperations

10.1 International initiatives

10.1.1 Inria associate team not involved in an IIL or an international program

CSG

Title: Proving Concurrent Multi-Core Operating Systems

Duration: 2019 -> 2022

Coordinator: Willy Zwaenepoel (willy.zwaenepoel@sydney.edu.au)

Partners:

- University of Sydney

Inria contact: Gilles Muller, Julia Lawall

Summary: The goal of this project is to cooperate on the development of proved multicore schedulers. Building on our design of a DSL (Ipanema) for expressing scheduling policies, we are working on (i) Better understanding of what should the best scheduler for a given multicore application, (ii) Proving the correctness of the C code generated from the DSL policy and of the Ipanema abstract machine, (iii) Extending the Ipanema DSL to the domain of I/O request scheduling [3], (iv) Designing a provable complete concurrent kernel. This project has resulted in a publication at EuroSys 2020, and intersects with the delegation of Virginia Aponte (2019-2020), our collaboration with Oracle, and our ANR VeriAmos project, with Antique and the University of Grenoble.

10.2 National initiatives

10.2.1 ANR

ITrans

Participants: Julia Lawall *(PI)*, Gilles Muller, Lucas Serrano, Van-Anh Nguyen, David Lo *(Singapore Management University (PI))*, Lingxiao Jiang *(Singapore Management University)*.

- Awarded in 2016, duration 2017 - 2021
- Members: LIP6 (Whisper), Singapore Management University
- Funding: ANR PRCI, 287,820 euros.

- Objectives:

  Large, real-world software must continually change, to keep up with evolving requirements, fix bugs, and improve performance, maintainability, and security. This rate of change can pose difficulties for clients, whose code cannot always evolve at the same rate. This project targets the problems of *forward porting*, where one software component has to catch up to a code base with which it needs to interact, and *back porting*, in which it is desired to use a more modern component in a context where it is necessary to continue to use a legacy code base, focusing on the context of Linux device drivers. In this project, we take a *history-guided source-code transformation-based* approach, which automatically traverses the history of the changes made to a software system, to find where changes in the code to be ported are required, gathers examples of the required changes, and generates change rules to incrementally back port or forward port the code. The developed tool Spinfer, published at USENIX ATC 2020 [9], is able to infer transformation rules from sets of examples of a wide range of Linux kernel changes. This is a critical building block in developing a technique to back and forward port drivers for the Linux operating system to various earlier and later kernel versions with high accuracy while requiring minimal developer effort.

VeriAmos

Participants: Xavier REival *(Antique (PI))*, Nicolas Palix *(UGA (Erods))*, Gilles Muller, Julia Lawall, Rehan Malak.

- Awarded in 2018, duration 2018 - 2022
- 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.

11 Dissemination

11.1 Promoting scientific activities

11.1.1 Scientific events: organisation

General chair, scientific chair

- Julia Lawall: Organizer of the 20th meeting of the IFIP WG 2.11 (Program generation), Feb 2021 (link)

Member of the organizing committees

- Gilles Muller is president of the steering committee of Compas
- Julia Lawall is a member of the steering committee of ASE (since 2019)

11.1.2 Scientific events: selection

Chair of conference program committees

- Julia Lawall: Chair of the Doctoral workshop at ICSME 2021, with Ferdian Thung

Reviewer

- Gilles Muller: NSDI 2021
- Julia Lawall: SOSP 2021, ICSE 2021, ECOOP 2021, EuroSys 2021, FSE 2021 (Visions and reflections track, Doctoral symposium), Benevol 2021

11.1.3 Journal

Member of the editorial boards

- Julia Lawall: Member of the editorial board of Science of Computer Programming, since 2008.

Reviewer - reviewing activities

- Julia Lawall has reviewed articles for journals such as Science of Computer Programming and IEEE Transactions on Software Engineering.

11.1.4 Invited talks

- Julia Lawall, PEPM 2021, "Program Manipulation of C Code: From Partial Evaluation to Semantic Patches for the Linux Kernel" (video)

11.1.5 Leadership within the scientific community

- Julia Lawall is the chair of the ASF/RSD prize committee (responsable for the PhD prize, junior researcher prize, and senior researcher prize) with Xavier Lagrange, 2021-2023.

11.1.6 Scientific expertise

- Gilles Muller: Vice-president of ANR committee 25.

11.1.7 Supervision

- Gilles Muller: supervisor of the PhD of Yoann Ghigoff (Jun 2019 - May 2022), with Julien Sopena (Delys) and Kahina Lazri (Orange)
- Gilles Muller: co-supervisor of the PhD of Josselin Giet (Sep 2020 - Aug 2023), with Xavier Rival (Antique)
- Julia Lawall: supervisor of the PhD of Himadri Pandya (Dec 2021 - Nov 2024)

11.1.8 Juries

- Julia Lawall: PhD jury member for Quentin Dufour, University of Rennes, February 2021
- Julia Lawall: PhD jury member for Nicolas Jeannerod, University of Paris, April 2021
- Julia Lawall: PhD jury member for Frederic Recoules, Université Grenoble Alpes, September 2021
- Julia Lawall: HDR jury member for Mathieu Acher, University of Rennes, November 2021
- Julia Lawall: PhD reporter for Hugo Martin, University of Rennes, December 2021

12 Scientific production

12.1 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: *POPL*. Savannah, GA, USA: ACM, Jan. 2009, pp. 114–126.

[2] L. Burgy, L. Réveillère, J. L. Lawall and G. Muller. ‘Zebu: A Language-Based Approach for Network Protocol Message Processing’. In: *IEEE Trans. Software Eng.* 37.4 (2011), pp. 575–591.

[3] 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.

[4] F. Mérillon, L. Réveillère, C. Consel, R. Marlet and G. Muller. ‘Devil: An IDL for hardware programming’. In: *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI)*. San Diego, California: USENIX Association, Oct. 2000, pp. 17–30.

[5] G. Muller, C. Consel, R. Marlet, L. P. Barreto, F. Mérillon and L. Réveillère. ‘Towards Robust OSes for Appliances: A New Approach Based on Domain-specific Languages’. In: *Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System*. Kolding, Denmark, 2000, pp. 19–24.

[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.

12.2 Publications of the year

International journals

[10] T. Hoang, J. Lawall, Y. Tian, R. J. Oentaryo and D. Lo. ‘PatchNet: Hierarchical Deep Learning-Based Stable Patch Identification for the Linux Kernel’. In: *IEEE Transactions on Software Engineering* 47.11 (Nov. 2021), pp. 2471–2486. DOI: 10.1109/TSE.2019.2952614. URL: https://hal.inria.fr/hal-02373994.

International peer-reviewed conferences

[11] A. N. Amalou, I. Puaut and G. Muller. 'WE-HML: hybrid WCET estimation using machine learning for architectures with caches'. In: RTCSA 2021 - 27th IEEE International Conference on Embedded Real-Time Computing Systems and Applications. Online Virtual Conference, France: IEEE, 18th Aug. 2021, pp. 1–10. URL: https://hal.inria.fr/hal-03280177.

[12] Y. Ghigoff, J. Sopena, K. Lazri, A. Blin and G. Muller. 'BMC: Accelerating Memcached using Safe In-kernel Caching and Pre-stack Processing'. In: NSDI'21 - 18th USENIX Symposium on Networked Systems Design and Implementation. Virtual event, United States: USENIX Association, 12th Apr. 2021, pp. 487–501. URL: https://hal.inria.fr/hal-03361644.

[13] S. Haryono, F. Thung, D. Lo, L. Jiang, J. Lawall, H. Jin Kang, L. Serrano and G. Muller. 'AndroEvolve: Automated Update for Android Deprecated-API Usages'. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion).* 2021 IEEE/ACM 43rd International Conference on Software Engineering. Madrid / Virtual, Spain: IEEE, 25th May 2021, pp. 1–4. DOI: 10.1109/ICSE-Companion52605.2021.00021. URL: https://hal.inria.fr/hal-03504710.

[14] S. A. Haryono, F. Thung, D. Lo, J. Lawall and L. Jiang. 'Characterization and Automatic Updates of Deprecated Machine-Learning API Usages'. In: ICSME 2021 - IEEE International Conference on Software Maintenance and Evolution. Luxembourg City / Virtual, Luxembourg, 29th Sept. 2021. DOI: 10.1109/ICSME52107.2021.00019. URL: https://hal.inria.fr/hal-03361379.

[15] S. A. Haryono, F. Thung, D. Lo, J. Lawall and L. Jiang. 'MLCatchUp: Automated Update of Deprecated Machine-Learning APIs in Python'. In: ICSME 2021 - 37th IEEE International Conference on Software Maintenance and Evolution. Luxembourg City / Virtual, Luxembourg, 27th Sept. 2021. DOI: 10.1109/ICSME52107.2021.00061. URL: https://hal.inria.fr/hal-03361370.

[16] M. Martone and J. Lawall. 'Refactoring for Performance with Semantic Patching: Case Study with Recipes'. In: C3PO'21: Compiler-assisted Correctness Checking and Performance Optimization for HPC (ISC workshop). virtual, Germany, 13th Nov. 2021, pp. 226–232. DOI: 10.1007/978-3-030-90539-2_15. URL: https://hal.inria.fr/hal-03266521.

[17] D. Mvondo, A. Barbalace, A. Tchana and G. Muller. 'Tell me when you are sleepy and what may wake you up!' In: SoCC 2021 - ACM Symposium on Cloud Computing. Seattle WA USA, United States: ACM, 1st Nov. 2021, pp. 562–569. DOI: 10.1145/3472883.3487013. URL: https://hal.archives-ouvertes.fr/hal-03503825.

[18] R. Shariffdeen, X. Gao, G. J. Duck, S. H. Tan, J. Lawall and A. Roychoudhury. 'Automated Patch Backporting in Linux (Experience Paper)'. In: ISSTA 2021: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. Aarhus (virtual), Denmark, 11th July 2021. DOI: 10.1145/3460319.3464821. URL: https://hal.inria.fr/hal-03359062.

[19] B. Teabe Djomgwe, P. Yuhala, A. Tchana, F. Hermenier, D. Hagimont and G. Muller. '(No)Compromis: Paging Virtualization Is Not a Fatality'. In: *à paraître*. VEE 2021 - 17th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Détroit, Michigan / Virtual, United States, 16th Apr. 2021, pp. 1–12. URL: https://hal.archives-ouvertes.fr/hal-03183858.

[20] D. N. Tu, B. Teabe Djomgwe, A. Tchana, G. Muller and D. Hagimont. 'Mitigating vulnerability windows with hypervisor transplant'. In: *EuroSys '21: Proceedings of the Sixteenth European Conference on Computer Systems.* EuroSys 2021 - European Conference on Computer Systems. Edinburgh / Virtual, United Kingdom: Association for Computing Machinery, 2021, pp. 1–14. DOI: 10.1145/3447786.3456235. URL: https://hal.archives-ouvertes.fr/hal-03183856.

12.3 Cited publications

[21] 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.

[22] T. F. Bissyandé, L. Réveillère, J. L. Lawall, Y.-D. Bromberg and G. Muller. 'Implementing an Embedded Compiler using Program Transformation Rules'. In: *Software: Practice and Experience* 45.2 (Feb. 2015), pp. 177–196. URL: https://hal.archives-ouvertes.fr/hal-00844536.

[23] T. F. Bissyandé, L. Réveillère, J. L. Lawall and G. Muller. 'Ahead of Time Static Analysis for Automatic Generation of Debugging Interfaces to the Linux Kernel'. In: *Automated Software Engineering* (May 2014), pp. 1–39. DOI: 10.1007/s10515-014-0152-4. URL: https://hal.archives-ouvertes.fr/hal-00992283.

[24] C. Cadar, D. Dunbar and D. R. Engler. 'KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs'. In: *OSDI*. 2008, pp. 209–224.

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

[26] E. Clarke, O. Grumberg, S. Jha, Y. Lu and H. Veith. 'Counterexample-guided abstraction refinement for symbolic model checking'. In: *J. ACM* 50.5 (2003), pp. 752–794.

[27] L. A. Clarke. 'A system to generate test data and symbolically execute programs'. In: *IEEE Transactions on Software Engineering* 2.3 (1976), pp. 215–222.

[28] P. Cousot and R. Cousot. 'Abstract Interpretation: Past, Present and Future'. In: *CSL-LICS*. 2014, 2:1–2:10.

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

[30] 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.

[31] D. R. Engler, D. Y. Chen, A. Chou and B. Chelf. 'Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code'. In: *SOSP*. 2001, pp. 57–72.

[32] L. Gu, A. Vaynberg, B. Ford, Z. Shao and D. Costanzo. 'CertiKOS: A Certified Kernel for Secure Cloud Computing'. In: *Proceedings of the Second Asia-Pacific Workshop on Systems (APSys)*. 2011, 3:1–3:5.

[33] L. Guo, J. L. Lawall and G. Muller. 'Oops! Where did that code snippet come from?' In: *11th Working Conference on Mining Software Repositories, MSR*. Hyderabad, India: ACM, May 2014, pp. 52–61.

[34] 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.

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

[36] G. A. Kildall. 'A Unified Approach to Global Program Optimization'. In: *POPL*. 1973, pp. 194–206.

[37] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch and S. Winwood. 'seL4: formal verification of an OS kernel'. In: *SOSP*. 2009, pp. 207–220.

[38] J. L. 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 Experience* 43.1 (2013), pp. 67–92.

[39] J. L. Lawall, B. Laurie, R. R. Hansen, N. Palix and G. Muller. 'Finding Error Handling Bugs in OpenSSL using Coccinelle'. In: *Proceeding of the 8th European Dependable Computing Conference (EDCC)*. Valencia, Spain, Apr. 2010, pp. 191–196.

[40] J. L. Lawall and D. Lo. 'An automated approach for finding variable-constant pairing bugs'. In: *25th IEEE/ACM International Conference on Automated Software Engineering*. Antwerp, Belgium, Sept. 2010, pp. 103–112.

[41] J. L. 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, July 2017, p. 12. URL: https://hal.inria.fr/hal-01556589.

[42] C. Le Goues and W. Weimer. 'Specification Mining with Few False Positives'. In: *TACAS*. Vol. 5505. Lecture Notes in Computer Science. York, UK, Mar. 2009, pp. 292–306.

[43] 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.

[44] 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.

[45] D. Lo and S.-C. Khoo. 'SMArTIC: towards building an accurate, robust and scalable specification miner'. In: *FSE*. 2006, pp. 265–275.

[46] J.-P. Lozi, F. David, G. Thomas, J. L. Lawall and G. Muller. 'Fast and Portable Locking for Multicore Architectures'. In: *ACM Transactions on Computer Systems* (Jan. 2016). DOI: 10.1145/2845079. URL: https://hal.inria.fr/hal-01252167.

[47] S. Lu, S. Park and Y. Zhou. 'Finding Atomicity-Violation Bugs through Unserializable Interleaving Testing'. In: *IEEE Transactions on Software Engineering* 38.4 (2012), pp. 844–860.

[48] M. Mernik, J. Heering and A. M. Sloane. 'When and How to Develop Domain-specific Languages'. In: *ACM Comput. Surv.* 37.4 (Dec. 2005), pp. 316–344. URL: http://dx.doi.org/10.1145/1118890.1118892.

[49] M. C. Olesen, R. R. Hansen, J. L. Lawall and N. Palix. 'Coccinelle: Tool support for automated CERT C Secure Coding Standard certification'. In: *Science of Computer Programming*. Special Issue on Selected Contributions from the Open Source Software Certification (OpenCert) Workshops 91.B (Oct. 2014), pp. 141–160. URL: https://hal.inria.fr/hal-01096185.

[50] T. Reps, T. Ball, M. Das and J. Larus. 'The Use of Program Profiling for Software Maintenance with Applications to the Year 2000 Problem'. In: *ESEC/FSE*. 1997, pp. 432–449.

[51] 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.

[52] C. Rubio-González, H. S. Gunawi, B. Liblit, R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau. 'Error propagation analysis for file systems'. In: *PLDI*. Dublin, Ireland: ACM, June 2009, pp. 270–280.

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

[54] 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.

[55] R. Saha, J. L. Lawall, S. Khurshid and D. E. Perry. 'On the Effectiveness of Information Retrieval based Bug Localization for C Programs'. In: *International Conference on Software Maintenance and Evolution (ICSME)*. Victoria, BC, Canada, Sept. 2014.

[56] 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://hal.inria.fr/hal-01086082.

[57] S. Saha, J.-P. Lozi, G. Thomas, J. L. Lawall and G. Muller. 'Hector: Detecting resource-release omission faults in error-handling code for systems software'. In: *DSN 2013 - 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE/IFIP. Budapest, Hungary: IEEE Computer Society, June 2013, pp. 1–12. DOI: 10.1109/DSN.2013.6575307. URL: https://hal.inria.fr/hal-00918079.

[58] D. A. Schmidt. 'Data Flow Analysis is Model Checking of Abstract Interpretations'. In: *POPL*. 1998, pp. 38–48.

[59] P. Senna Tschudin, J. L. Lawall and G. Muller. '3L: Learning Linux Logging'. In: *BElgian-NEtherlands software eVOLution seminar (BENEVOL 2015)*. Lille, France, Dec. 2015. URL: https://hal.inria.fr/hal-01239980.

[60] M. Shapiro. 'Purpose-built languages'. In: *Commun. ACM* 52.4 (2009), pp. 36–41.

[61] R. Tartler, D. Lohmann, J. Sincero and W. Schröder-Preikschat. 'Feature consistency in compile-time-configurable system software: facing the Linux 10,000 feature problem'. In: *EuroSys*. 2011, pp. 47–60.

[62] 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`.

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

[64] J. Yang and C. Hawblitzel. ‘Safe to the Last Instruction: Automated Verification of a Type-safe Operating System’. In: *PLDI*. 2010, pp. 99–110.