Research internship (Master 2 level)

Formal methods for component-based distributed systems reconfiguration

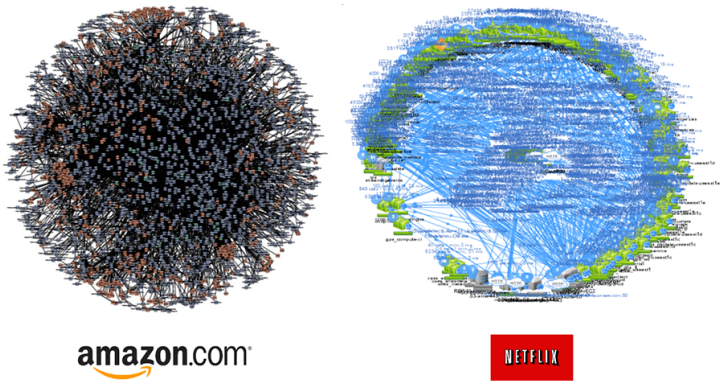
Hélène Coullon
IMT Atlantique, Inria

Frédéric Loulergue
Université d'Orléans

2021

1 Introduction

With the advent of Cloud computing, service-oriented software and distributed systems have become commonplace. However, the complexity and scale of these systems continue to increase, in particular through the development of architectures such as Edge and Fog computing, and associated applications such as smart city or smart industry, or the autonomous car. For example, the micro-services-based architecture of Netflix, Amazon, or social networks contains thousands of small interconnected components, forming what the community calls a black star (in reference to Star Wars). A particularity of these complex service-based software and distributed systems is their duration over time, in other words their long life cycle.



A key ingredient for the long-term success of these systems is the ability to adapt to a changing environment, or the evolution of software services and their dependencies, without affecting functionality. We will focus on the deployment and reconfiguration of these large distributed software systems in this internship. Deployment consists in the commissioning of the software into Cloud, Fog or Edge computing infrastructures, which implies complex coordination due to the numerous interactions between services. Reconfiguration consists in keeping these systems available, efficient, etc. over time through operations such as adding or removing components (adapting the available functionalities), substituting certain components (updating), connecting or disconnecting certain components, changing the internal configuration of a component, etc.

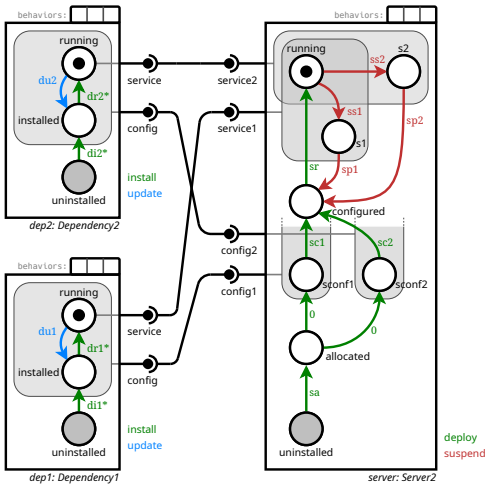
Manually managing the deployment or reconfiguration of distributed systems like the Netflix stack reaches the limits of human possibilities. Automatic tools are therefore obviously preferred. There is a very large number of tools in the DevOps community to automate deployment, such as Ansible, Puppet, DockerCompose etc. There are also more advanced orchestrators to manage the life cycle of components and applications, and to handle basic and specific maintenance, such as Docker Swarm or Kubernetes. However, reconfiguration possibilities remain relatively limited in production because it involves complex modifications to be implemented on sensitive services that must remain highly available.

Due to the **lack of safety and fault tolerance** in these operations, more and more professionals are turning to tools that create new instances rather than modify old (immutable) ones. However, this implies temporary duplication of many services, sometimes costly, and sometimes impossible (limited resources).

In this internship we want to work on the application of formal methods for the verification and safety of the reconfiguration of distributed software systems.

2 Context

Our work specifically targets the design and execution of dynamic reconfiguration of component-based software systems. The notion of component used here is very general, and applies for example to component-based software systems, service-oriented applications, micro-services, *etc.*. A reconfiguration in this context essentially consists of switching from one system configuration to another, while ensuring safety and security properties. By configuration, we mean the topology of the distributed system (i.e. the components and their connections), and the state of its various components (either started / stopped, or a finer characterization that can be specific to each type of component).



Concerto [4] is a model for managing the life cycle of software components and coordinating their reconfiguration operations. Concerto promotes efficiency with a fine representation of dependencies and a parallel execution of reconfiguration actions, both within and between components. Concerto has formal semantics, and a performance model that can be used to estimate the time required for reconfiguration. There is also an implementation, which has allowed the validation of the accuracy of the performance estimation, and has demonstrated the performance gains provided by Concerto compared to the state of the art. Madeus [3] is a specialization (a subset) of Concerto to manage only the deployment of distributed software systems.

Deductive verification with tools such as the Coq [10] proof assistant, and the Frama-C [7] framework, help to ensure the absence of bugs in complex software systems, and have been used in many areas

including compilation [8], parallel programming [5], component-based systems [9, 6], and the Internet of Things [1, 2], to name a few.

3 Expected work

Depending on the student's progress and interests, the following tasks will be covered during the internship :

1. formal specification in Coq of Madeus [3],
2. generic proofs on Madeus, and automatic code extraction in OCaml or Haskell,
3. higher-level language in Coq to specify a Madeus deployment instance and to easily make proofs on it,
4. formal specification in Coq of Concerto [4] to extend the one of Madeus.

If the internship goes well and the work provided is serious and of good quality it will be possible to continue on a PhD in computer science.

4 Skills

Student in the last year of a Master's degree in Computer Science (or in the last year of an engineering school with a computer science option). Skills and interest in formal methods and mathematical proof. General knowledge of distributed and parallel systems. Research interest and curiosity.

5 Additional information

Advisors

— Hélène Coullon, IMT Atlantique & Inria équipe Stack, helene.coullon@imt-atlantique.fr

— Frédéric Loulergue, Université d'Orléans, LIFO, équipe LMV, frederic.loulergue@univ-orleans.fr

Duration up to 6 months

Salary legal amount of 3,90€ / hour, full time

Location (to choose)

— IMT Atlantique, équipe Inria Stack, laboratoire LS2N à Nantes

— Université d'Orléans, équipe LMV, laboratoire LIFO à Orléans

Références

[1] Allan Blanchard, Nikolai Kosmatov, and Frédéric Loulergue. Ghosts for Lists : A Critical Module of Contiki Verified in Frama-C. In *Nasa Formal Methods*, number 10811 in LNCS, pages 37–53. Springer, 2018. doi:10.1007/978-3-319-77935-5_3.

[2] Allan Blanchard, Nikolai Kosmatov, and Frédéric Loulergue. Logic against ghosts : Comparison of two proof approaches for a list module. In *ACM Symposium on Applied Computing (SAC)*, pages 2186–2195. ACM, 2019. doi:10.1145/3297280.3297495. Best Paper Award.

[3] Maverick Chardet, Hélène Coullon, Christian Pérez, Dimitri Pertin, Charlène Servantie, and Simon Robillard. Enhancing Separation of Concerns, Parallelism, and Formalism in Distributed Software Deployment with Madeus. working paper or preprint, June 2020. URL https://hal.inria.fr/hal-02737859.

[4] Maverick Chardet, Hélène Coullon, and Simon Robillard. Toward safe and efficient reconfiguration with concerto. *Sci Comput Program*, 203, 2021. doi:10.1016/j.scico.2020.102582.

[5] Nuno Gaspar, Ludovic Henrio, and Eric Madelaine. Bringing Coq into the world of GCM distributed applications. *Int J Parallel Prog*, 42(4) :643–662, 2014. doi:10.1007/s10766-013-0264-7.

[6] Nuno Gaspar, Ludovic Henrio, and Eric Madelaine. Painless support for static and runtime verification of component-based applications. In *Fundamentals of Software Engineering (FSEN)*, volume 9392 of *LNCS*, pages 259–274. Springer, 2015. doi:10.1007/978-3-319-24644-4_18.

[7] Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. Frama-C : A software analysis perspective. *Formal Asp. Comput.*, 27(3) :573–609, 2015. doi:10.1007/s00165-014-0326-7. URL http://frama-c.com.

[8] Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7) :107–115, 2009. doi:10.1145/1538788.1538814.

[9] Frédéric Loulergue, Wadoud Bousdira, and Julien Tesson. Calculating Parallel Programs in Coq using List Homomorphisms. *Int J Parallel Prog*, 45 :300–319, 2017. doi:10.1007/s10766-016-0415-8.

[10] The Coq Development Team. The Coq proof assistant version 8.12. http://coq.inria.fr, 2020.