Stage de dernière année d'école d'ingénieur / Master

Méthodes formelles pour la reconfiguration logicielle à base de composants

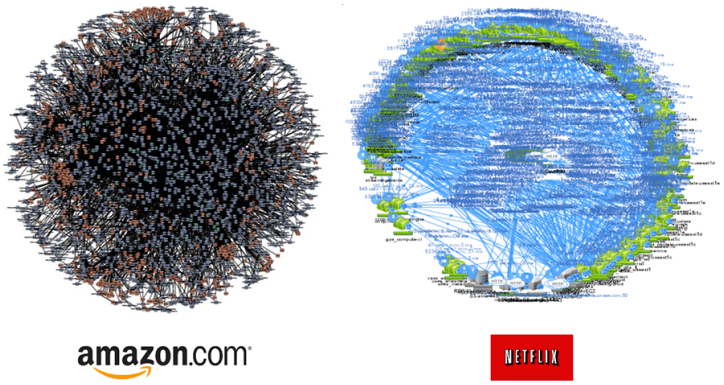
Hélène Coullon
IMT Atlantique, Inria

Frédéric Loulergue
Université d'Orléans

2021

1 Introduction

Avec l'avènement du *Cloud computing*, les logiciels et systèmes distribués orientés services sont devenus monnaie courante. Toutefois, la complexité et l'échelle de ces systèmes continuent d'augmenter, en particulier par le développement d'architectures comme le *Edge* et le *Fog* computing, et les applications qui y sont associées, comme la ville ou l'industrie intelligente, ou la voiture autonome. Ainsi, par exemple, l'architecture à base de micro-services de Netflix, Amazon, ou encore des réseaux sociaux, contient des milliers de petits composants inter-connectés les uns avec les autres, formant ce qui est appelé par la communauté une étoile noire (en référence à Star Wars). Une particularité des ces logiciels et système distribués complexes à base de services est leur durée dans le temps, autrement dit leur long cycle de vie.



Un ingrédient clé pour le succès à long terme de ces systèmes est la capacité de s'adapter à un environnement changeant, à l'évolution des exigences opérationnelles, ou à l'évolution des services logiciels et de leurs dépendances, sans affecter les fonctionnalités. Nous nous intéresseront au *déploiement* et à la *reconfiguration* de ces larges logiciels distribués dans ce stage. Le déploiement consiste en la mise en service sur une infrastructure de type *Cloud*, *Fog* ou *Edge computing* du logiciel ce qui implique une coordination complexe due aux nombreuses interactions entre les services. La reconfiguration consiste à faire perdurer dans le temps le fonctionnement de ces systèmes par le biais d'opérations telles que l'ajout

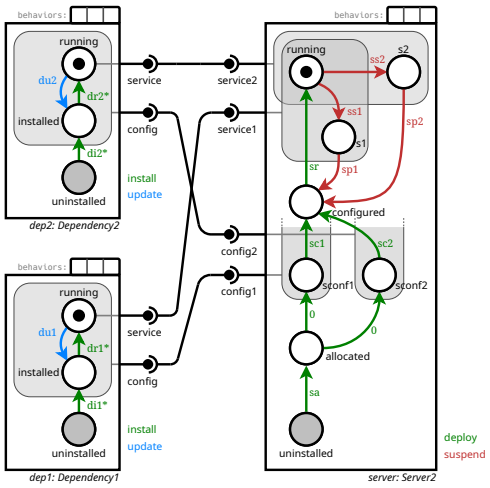
ou la suppression de composants (adaptation des fonctionnalités disponibles), la substitution de certains composants (mise à jour), la connexion ou la déconnexion de certains composants, le changement de configuration interne d'un composant, *etc.*

Gérer manuellement la mise en service ou la maintenance de systèmes distribués comme la pile Netflix atteint les limites des possibilités humaines. Des outils automatiques sont donc bien évidemment privilégiés. Il existe un très grand nombre d'outils dans la communauté "DevOps" permettant d'automatiser le déploiement, comme Ansible, Puppet, DockerCompose. Il existe également des orchestrateurs plus avancés permettant de gérer un cycle de vie et des maintenances basiques et spécifiques comme Docker Swarm ou Kubernetes. Toutefois, les possibilités de reconfiguration restent relativement limitées en production car il s'agit de modifications complexes à mettre en oeuvre, sur des services sensibles qui doivent rester hautement disponibles. Par **manque de sûreté et de tolérance aux pannes** dans ces opérations, de plus en plus de professionnels se tournent vers des outils qui créée de nouvelles instances plutôt que de modifier les anciennes (immuables). Toutefois cela implique des duplications temporaires de nombreux services, parfois coûteuses, et parfois impossibles (limitation de ressources).

Dans ce stage nous souhaitons travailler sur l'application de méthodes formelles pour la vérification et la sûreté de la reconfiguration de logiciels distribués.

2 Contexte du stage

Nos travaux ciblent spécifiquement la conception et l'exécution de reconfigurations dynamiques de systèmes logiciels à base de composants. La notion de composant utilisée ici est très générale, et s'applique par exemple aux systèmes logiciels à base de composants, aux applications orientées services, aux microservices, *etc.* Une reconfiguration dans ce contexte consiste essentiellement à passer d'une configuration du système logiciel à une autre, tout en garantissant des propriétés de sûreté et de sécurité. Par configuration, nous entendons la topologie du système distribué (c'est-à-dire les composants et leurs connexions), et l'état de ses différents composants (soit démarré / arrêté, soit une caractérisation plus fine qui peut être spécifique à chaque type de composant).



Concerto [4] est un modèle permettant de gérer le cycle de vie des composants logiciels et de coordonner leurs opérations de reconfiguration. Concerto favorise l'efficacité avec une représentation fine des dépendances et une exécution parallèle des actions de reconfiguration, à la fois au sein des composants et entre eux. Concerto possède une sémantique formelle et un modèle de performance qui peut être utilisé pour estimer le temps requis par les reconfigurations. Il existe également une implantation, ce qui a permis d'évaluer l'exactitude des estimations de performances et a démontré les gains de performances fournis par le modèle d'exécution de Concerto par rapport à l'état de l'art. Madeus [3] est

une spécialisation (un sous-ensemble) de Concerto pour gérer uniquement le déploiement des logiciels distribués.

La vérification déductive avec des outils tels que l'assistant de preuve Coq [10] et le *framework* Frama-C [7] permettent de garantir l'absence de bogues dans des systèmes logiciels complexes, et ont été utilisés dans de nombreux domaines incluant la compilation [8], la programmation parallèle [5], les systèmes à composants [9, 6], et l'Internet des Objets [1, 2] pour en citer quelques uns.

3 Tâches du stage

En fonction de l'avancée de l'étudiant(e) et de ses intérêts, les tâches suivantes seront abordées dans le stage :

1. spécification formelle en Coq de Madeus [3],
2. preuves sur le modèle Madeus, et extraction de code correct en OCaml ou Haskell,
3. langage de plus haut niveau pour la spécification en Coq d'une instance d'assemblage Madeus et preuve (semi-)automatique,
4. extension de la spécification formelle à Concerto [4].

Si le stage se déroule bien et que le travail fourni est sérieux et de bonne qualité il sera possible de continuer sur un doctorat en informatique.

4 Compétences attendues

Étudiant(e) en dernière année de Master en informatique (ou en dernière année d'école d'ingénieur en option informatique). Compétences et intérêt pour les méthodes formelles et les preuves mathématiques. Connaissances générales sur les systèmes distribués et parallèles. Attrait pour la recherche et curiosité.

5 Informations pratiques

Encadrants

— 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

Durée jusqu'à 6 mois

Indemnités montant légal de 3,90€ / heure, temps plein

Lieux du stage (lieu à choisir)

— 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.