Système de fichiers efficace pour MirageOS

Proposition de stage de recherche (niveau L3/M1)

2022

Encadrant

Armaël Guéneau, chargé de recherche, LMF et Inria Saclay (armael.gueneau@inria.fr)

Résumé

Le projet MirageOS peut être vu comme un écosystème de bibliothèques OCaml implémentant des abstractions fournies ordinairement par un système d'exploitation. Une de ces abstractions, qui fait défaut actuellement, est un *système de fichiers* généraliste. J'ai récemment travaillé à adapter FSCQ, un système de fichiers implémenté en Coq, pour le déployer comme une bibliothèque OCaml. Malheureusement, les performances du code OCaml que j'ai obtenu ne sont pas encore satisfaisantes. Dans ce stage, je propose de travailler à améliorer les performances de ce système de fichiers afin de le rendre utilisable dans le cadre de MirageOS.

Situation du sujet

Le projet MirageOS [MS13, mir] se propose comme une alternative radicale à la séparation classique entre application et système d'exploitation. MirageOS définit un ensemble d'interfaces et de bibliothèques modulaires, implémentées en OCaml ; celles-ci fournissent certaines des fonctionnalités que l'on trouve dans un système d'exploitation standard, mais peuvent être inclues à la demande et liées directement à l'application finale qui les utilise.

Cette architecture, avec ses bibliothèques séparées aux rôles bien définis, fait du projet MirageOS un cadre attirant pour l'application de techniques de vérification formelle. On peut ainsi se concentrer sur la vérification de bibliothèques individuelles ; tant que l'on obtient une bibliothèque OCaml implémentant une interface MirageOS, alors cette bibliothèque est potentiellement utilisable par l'ensemble de la communauté MirageOS, et peut être déployée « dans la vraie vie » en combinaison avec d'autres composants de l'écosystème MirageOS.

Un exemple de bibliothèque utile pour MirageOS est un système de fichiers (permettant de stocker une hiérarchie de fichiers et dossiers sur un disque dur ou *block device* arbitraire). Actuellement, MirageOS dispose de bibliothèques implémentant des systèmes de fichiers plus ou moins spécialisés ou au stade de prototypes, mais pas de système de fichiers généraliste et efficace.

À cet effet, j'ai récemment commencé un projet visant à adapter et déployer le système de fichiers FSCQ [CZC+15b, CZC+15a, CCK+17] pour MirageOS. FSCQ est implémenté et formellement vérifié en Coq ; les auteurs de FSCQ utilisent le mécanisme d'« extraction » de Coq pour obtenir du

code Haskell exécutable à partir de l'implémentation en Coq. Afin de pouvoir utiliser FSCQ dans MirageOS, j'ai écrit du code permettant d'obtenir une implémentation OCaml de FSCQ (également via l'extraction de Coq), travail qui a été intégré par les mainteneurs de FSCQ [fsc].

Malheureusement, les performances du code OCaml obtenu laissent pour l'instant à désirer. La raison principale est que l'implémentation de FSCQ a été optimisée spécifiquement pour le cas de l'extraction du code Coq vers Haskell. En particulier, l'implémentation de FSCQ suppose aujourd'hui que le code après extraction utilisera l'*évaluation paresseuse* et des structures de données paresseuses par défaut—ce qui est le cas pour Haskell mais pas OCaml.

Dans ce stage, je propose de commencer par mesurer rigoureusement les performances du code OCaml généré actuellement par l'extraction Coq, puis de l'optimiser, afin d'atteindre des performances satisfaisantes permettant son utilisation au sein de MirageOS. Une première piste serait de configurer le mécanisme d'extraction de Coq pour utiliser des structures de données *explicitement paresseuses* en OCaml au lieu de certaines structures de données strictes (par exemple, des listes paresseuses au lieu de listes par défaut). Une autre piste est d'utiliser des structures de données supportant efficacement certaines opérations utilisées fréquemment par l'implémentation : par exemple, utiliser des *catenable sequences* [Oka99, §10.2.1] (avec concatenation en $O(1)$ amorti) à la place de simples listes ; ou encore la bibliothèque Sek [sek]. Bien sûr, dans certains cas, il sera plus simple de modifier directement l'implémentation Coq de FSCQ afin de produire directement du coq OCaml efficace après extraction—dans ce cas, il faudrait idéalement mettre à jour la preuve de correction en Coq.

Objectifs

Le programme de travail est le suivant.

1. Se familiariser avec le code existant et mettre en place des *tests de performance* pour la version OCaml de FSCQ, et ce pour les différentes opérations supportées (lecture ou écriture dans un fichier, recherche dans un dossier, suppression d'un fichier, etc).
2. Identifier les opérations les plus lentes, et les *profiler* pour comprendre la cause (en utilisant un outil comme `perf`). Identifier les occurences liées à une (mauvaise) utilisation des structures de données strictes d'OCaml par un code s'attendant à des structures de données paresseuses.
3. Remplacer les structures de données utilisées inefficacement par des structures paresseuses, et mesurer l'impact du changement en terme de performances. Essayer également en utilisant des structures de données optimisant certaines opérations spécificiques ; si besoin, concevoir une ou des structures de données « sur-mesure » adaptées aux besoins de l'implémentation de FSCQ (à grand renfort de *Purely Functional Data Structures* d'Okasaki [Oka99]).

Pré-requis

De bases solides en algorithmique et en programmation sont indispensables. Une familiarité avec le langage OCaml est très souhaitable.

Détails pratiques

Le stage se déroulera au LMF, bâtiment 650, sur le plateau de Saclay. (Campus Universitaire, Rue Raimond Castaing, Bâtiment 650, 91190 Gif-sur-Yvette)

Références

[CCK+17] Haogang Chen, Tej Chajed, Alex Konradi, Stephanie Wang, Atalay İleri, Adam Chlipala, M Frans Kaashoek, and Nickolai Zeldovich. Verifying a high-performance crash-safe file system using a tree specification. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 270–286, 2017.

[CZC+15a] Haogang Chen, Daniel Ziegler, Tej Chajed, Adam Chlipala, M Frans Kaashoek, and Nickolai Zeldovich. Using crash hoare logic for certifying the fscq file system. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 18–37, 2015.

[CZC+15b] Haogang Chen, Daniel Ziegler, Adam Chlipala, M Frans Kaashoek, Eddie Kohler, and Nickolai Zeldovich. Specifying crash safety for storage systems. In *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*, 2015.

[fsc] Pull-request #18 on the FSCQ Github repository. https://github.com/mit-pdos/fscq/pull/18.

[mir] The MirageOS website. mirage.io.

[MS13] Anil Madhavapeddy and David J Scott. Unikernels : Rise of the virtual library operating system : What if all the software layers in a virtual appliance were compiled within the same safe, high-level language framework ? *Queue*, 11(11) :30–44, 2013.

[Oka99] Chris Okasaki. *Purely Functional Data Structures*. 1999.

[sek] The Sek library. https://gitlab.inria.fr/fpottier/sek.