Sujet de stage L3 informatique : Implémentation d'un algorithme d'accord de treillis pour la réplication de machine à l'aide de CRDT

Encadrants : François-Xavier Dupé, Arnaud Labourel, Alessia Milani
DALGO, LIS, Aix-Marseille Université, CNRS
Courriel : francois-xavier.dupe@lis-lab.fr, alessia.milani@lis-lab.fr, arnaud.labourel@lis-lab.fr

1 Problème de l'accord de treillis

Le problème du consensus est un problème fondamental pour les systèmes distribués tolérants aux pannes. Il consiste pour un ensemble de machines (appelées processus par la suite) à se mettre d'accord sur une valeur ou, par extension, sur une séquence de valeurs. Malheureusement, ce problème devient indécidable dans un système asynchrone si on autorise un processus à tomber en panne [6]. Le problème de l'accord de treillis (*lattice agreement* en anglais) est un problème plus facile à résoudre que le consensus dans le sens où il reste décidable même en présence de processus tombant en panne. Contrairement au problème du consensus, les différents processus ne sont pas obligés de décider une valeur commune. Le problème se définit à partir de la notion de demi-treillis clos par le haut, mais il est plus facile de se ramener aux treillis ensemblistes en prenant comme valeurs d'entrées et de sortie des ensembles. On suppose donc que tous les processus commence avec une valeur d'entrée qui est un ensemble (pouvant représenter un ensemble de mises à jour) et doit décider d'une valeur en sortie. Les conditions à respecter sont les suivantes :

- Validité : la valeur décidée d'un processus est un sur-ensemble de sa valeur d'entrée et correspond à l'union des valeurs d'entrées d'un ensemble de processus ;
- Stabilité : un processus ne décide qu'une seule valeur (impossible de changer la valeur une fois celle-ci décidée) ;
- Consistance : les valeurs décidées par deux processus sont comparables ;
- Vivacité : chaque processus décide en un temps fini d'une valeur.

L'accord sur les treillis a été d'abord été utilisé pour concevoir des algorithmes distribués dans des systèmes à mémoire partagée [4, 3, 7]. Durant ce stage, on s'intéressera à l'implémentation d'algorithmes résolvant ce problème dans un système à passage par messages [5, 15]. Plus spécifiquement, on s'intéressera à sa généralisation qui consiste en une version itérée du problème dans laquelle chaque processus reçoit une suite de valeurs d'entrées potentiellement infinie et les processus (correspondant à des serveurs) doivent valider les valeurs des autres processus en un temps fini. L'objectif est donc pour chaque processus de maintenir un ensemble de valeurs validées respectant les conditions suivantes :

- Validité : à tout moment, la valeur validée d'un processus correspond à l'union d'un ensemble de valeurs d'entrées de processus ;
- Stabilité : l'ensemble des valeurs validées d'un processus ne fait que croître ;
- Consistance : les ensembles de valeurs validées par deux processus sont comparables (un des deux ensembles est inclus dans l'autre) ;
- Vivacité : chaque valeur d'entrée d'un processus est ajoutée à chaque ensemble de valeurs validées d'un processus en un temps fini.

2 Réplication de machine à états et CRDT

L'accord de treillis généralisé est souvent utilisé pour créer une machine à état répliquée entre plusieurs serveurs. Intuitivement, l'idée est de dupliquer sur plusieurs serveurs une machine sur laquelle on peut faire des mises à jour (opérations qui modifient l'état de la machine) et des lectures (opérations ne modifiant pas l'état de la machine, mais renvoyant une valeur). Les serveurs mettent en place un système d'accord de treillis généralisé dans

lequel les valeurs d'entrées sont les mises à jour et utilise alors ce système pour valider les mises à jour à appliquer localement sur la machine afin de répondre aux requêtes en lecture. Contrairement à des systèmes basés sur un algorithme de consensus qui peuvent facilement répliquer n'importe quelle machine à état, car les processus se mettent d'accord sur l'ordre des mises à jour à appliquer, les systèmes basés sur l'accord de treillis nécessitent que la machine à répliquer ait certaines propriétés. L'état de la machine doit être un type de données répliqué sans conflit (*Conflict-Free Replicated Data Type*, abrégé CRDT) [13] qui permettent aux machines répliquées de converger vers un état commun quel que soit l'ordre de leurs mises à jour, garantissant ainsi une cohérence éventuelle forte. Les CRDT sont utilisés dans de nombreuses applications comme des éditeurs collaboratifs en temps réel. Ils permettent d'obtenir une collaboration fluide sans impact sur la cohérence des données, malgré une latence élevée ou des pannes. Ils existent deux types distincts de CRDT qui sont en théorie équivalent (l'un pouvant émuler l'autre) mais qui ont des différences en pratiques :

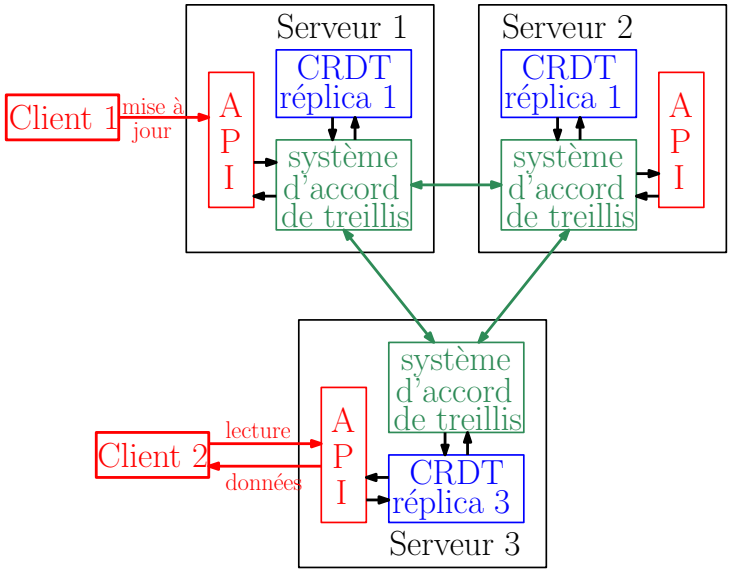
— *Convergent Replicated Data Types (CvRDT)* : reposant sur l'envoi de l'état de la machine et une opération de fusion d'états correspondant à la borne supérieure d'un demi-treillis ;

— *Commutative Replicated Data Types (CmRDT)* : reposant sur l'envoi de mises à jour ayant la propriété que les mises à jour qui ne sont pas liées par une relation de causalité commutent (donne le même état final quel que soit leur ordre d'application).

Ils existent de nombreux CRDT pour des types de données et applications assez différents : registres [13], édition texte [12], graphes [11, 14].

3 Objectifs du stage

Le but du stage est d'implémenter un système logiciel permettant de maintenir une machine répliquée (par exemple pour l'édition de texte collaborative entre plusieurs serveurs) via un mécanisme d'accord de treillis et l'utilisation de CRDT. Chaque serveur maintiendra donc une copie des données sous la forme d'un CRDT, communiquera avec les autres serveurs afin de valider les mises à jour via un système d'accord de treillis et sera connecté à des applications clients qui seront à l'origine des requêtes de mises à jour et de lecture via une API. La figure ci-dessous donne un schéma de l'organisation d'un tel système avec 3 serveurs.



L'implémentation du projet se fera dans le langage de programmation Kotlin [8] qui est un langage de programmation orienté objet et fonctionnel, avec un typage statique. Ce langage a été conçu comme une alternative moderne à Java (syntaxe plus concise, coroutine pour la gestion des tâches asynchrones, programmation fonctionnelle mieux intégrée dans le langage, ...) et est de plus utilisé pour le code déployé sur les serveurs, mais aussi pour les applications clients.

— Un module CRDT implémentant un système de données de type CRDT. On pourra imaginer plusieurs types de données (registres [13], édition texte [12], graphes [11, 14]). Le stockage des données pourra se faire soit dans une base de données SQL (en utilisant HyperSQL [2] et exposed [1]) ou bien noSQL via [10].

— Un module d'accord de treillis utilisant la bibliothèque [9] afin d'assurer la communication entre les serveurs.
— Un module API gérant une API HTTP CRUD de mise à jour et de lecture de données.
— Des applications clients interrogeant les serveurs via l'API REST.

Les modules devront le plus possible être indépendants dans le sens où il devra être facile de changer un des modules sans changer les autres parties du système. Le travail à réaliser étant conséquent et demandant des compétences assez variées à la fois sur le plan technique et théorique, le travail pourra être divisé entre plusieurs stagiaires.

Références

[1] Exposed, an orm framework for kotlin. `https://github.com/JetBrains/Exposed`.

[2] Systèmes de gestion de bases de données hypersql database. `https://hsqldb.org/`.

[3] Hagit Attiya and Arie Fouren. Adaptive and efficient algorithms for lattice agreement and renaming. *SIAM Journal on Computing*, 31(2) :642–664, 2001.

[4] Hagit Attiya, Maurice Herlihy, and Ophir Rachman. Atomic snapshots using lattice agreement. *Distributed Computing*, 8 :121–132, 1995.

[5] Jose M Faleiro, Sriram Rajamani, Kaushik Rajan, Ganesan Ramalingam, and Kapil Vaswani. Generalized lattice agreement. In *Proceedings of the 2012 ACM symposium on Principles of distributed computing*, pages 125–134, 2012.

[6] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2) :374–382, April 1985.

[7] Michiko Inoue, Toshimitsu Masuzawa, Wei Chen, and Nobuki Tokura. Linear-time snapshot using multi-writer multi-reader registers. In *Distributed Algorithms : 8th International Workshop, WDAG'1994 Terschelling, The Netherlands, September 29–October 1, 1994 Proceedings 8*, pages 130–140. Springer, 1994.

[8] JetBrains. Langage de programmation kotlin. `https://kotlinlang.org/`.

[9] JetBrains. Library ktor. `https://ktor.io/`.

[10] inc MongoDB. Système de gestion de bases de données mongodb. `https://hsqldb.org/`.

[11] Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. Data consistency for p2p collaborative editing. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 259–268, 2006.

[12] Nuno Preguiça, Joan Manuel Marquès, Marc Shapiro, and Mihai Letia. A commutative replicated data type for cooperative editing. In *2009 29th IEEE International Conference on Distributed Computing Systems*, pages 395–403. IEEE, 2009.

[13] Marc Shapiro, Nuno Preguiça, Carlos Baquero, Marek Zawirski, et al. Convergent and commutative replicated data types. *Bulletin of EATCS*, 2(104), 2013.

[14] Stephane Weiss, Pascal Urso, and Pascal Molli. Logoot-undo : Distributed collaborative editing system on p2p networks. *IEEE transactions on parallel and distributed systems*, 21(8) :1162–1174, 2010.

[15] Xiong Zheng, Changyong Hu, and Vijay Garg. Lattice agreement in message passing systems, 07 2018.