Projet 1 ALGOREP 2023

Étienne Renault

September 16, 2022

Rendu. Ce projet est à rendre pour le 21 novembre 2022 au plus tard à 23h59. Il s'effectue par groupes de 5 ou 6 au choix (pas moins sauf dérogation accordée explicitement de ma part). Le rendu s'effectuera par mail (un par groupe) avec comme objet [ALGOREP][2023][projet] nom1 nom2 nom3 nom4 nom5. Le rendu se composera de code (commenté !), d'un `Makefile` (je veux juste faire make, pas avoir à gérer les dépendances à votre place), et d'un `README`.

Soutenance. Vous présenterez votre travail pendant une soutenance organisée de la manière suivante :

- 15 minutes de présentation
- 5 minutes de démo "live" de votre projet
- 5 minutes de questions

Cette présentation portera sur vos **choix d'implémentation**, mettra en avant des **mesures de performances**, et détaillera vos tests.

Pénalités. Tout retard donnera lieu à une pénalité. Tout projet qui ne compile pas avec la simple comande make donnera aussi lieu à une pénalité. Toute archive qui en se décompressant donne autre chose qu'un répertoire `nom1_nom2_nom3_nom4_nom5` donnera lieu à une pénalité. Tout groupe avec un mauvais nombre de participant donnera lieu à pénalité. Une absence à la présentation finale donnera lieu à une pénalité. Tout mail mal nommé donnera lieu à une pénalité.

Notation. Votre note d'UE sera composée à 70% de la note de projet, et à 30% de la note de soutenance.

Avant-propos. Dans les dernières versions, MPI offre la possibilité d'avoir une mémoire partagée. Il est bien sur hors de question ici de se reposer la-dessus. Je veux que vous travailliez via le passage par message. De plus MPI offre de nombreux langages de programmation ; vous pouvez choisir celui que vous souhaitez du moment qu'il reste mainstream : si vous avez un doute sur ce qu'est un langage mainstream demandez moi ! Enfin pensez que votre application puisse être testée avec un nombre de processus différent de celui que vous utiliserez ! Vous pouvez également utiliser Go, ses goroutines et ses channels.

Description du Projet

Ce projet est volontairement imprécis. L'objectif étant de vous faire réfléchir et regarder l'état de l'art. Pensez à mettre dans votre présentation vos remarques, vos lectures et les problèmes rencontrés/résolus.

Description Nous souhaitons mettre en place un système clients / serveurs avec un mécanisme permettant de contrôler ou d'injecter des fautes dans le système. L'idée générale est la suivante : les clients proposent des valeurs / commandes aux serveurs. Ces serveurs souhaitent ensuite se mettre d'accord sur l'ordre dans lequel ils vont accepter, puis exécuter ces commandes. Une fois d'accord, il les écriront dans un fichier de log, et les exécuteront. Ces commandes, à terme, permettront de manipuler un stockage de fichier. Chaque serveur doit, à la fin de l'exécution, avoir le même fichier de log, et les mêmes fichiers stockés. Il s'agit donc d'une forme de réplication de logs et de fichiers.

Étape 1 – Consensus (5 pts). Afin de vous guider dans le projet, procédons par étapes, en augmentant le niveau de difficulté. La première étape consiste à construire un système dans lequel chaque client va posséder une valeur/commande originale (typiquement son uid). Les clients vont alors soumettre leurs valeurs aux serveurs. Les serveurs se mettent alors d'accord. Lorsque la valeur est répliquée, le serveur notifie le client. Lorsque toutes les valeurs ont été intégrées le système se termine. Comme chaque serveur a sa propre version du log, il est alors facile de s'assurer que tous les logs sont identiques. La mise en place d'un script permettant de demander interactivement le nombre de clients et de serveurs avant d'appeler MPI est sans doute une bonne idée.

Étape 2 – REPL (5 pts). L'objectif est maintenant de créer un processus qui va artificiellement modifier le comportement de notre système (autrement dit un injecteur de fautes). Ce processus est donc en "dehors" de notre système mais va envoyer des messages pour interagir avec les processus du système. (Pensez que maintenant les processus du système doivent être capable de lire des messages venant d'horizons différents). Pour le moment notre REPL doit être capable d'envoyer les ordres suivants :

`SPEED` **(low, medium, high)** qui va impacter la vitesse d'execution d'un processus. Autrement dit, il va y avoir maintenant un timer permettant de ralentir la vitesse des différents processus du système.

`CRASH` qui va simuler la mort d'un processus. Tous les messages reçus seront ignorés (a l'exception de ceux de la REPL).

`START` qui permet de dire aux client qu'ils peuvent, à partir de maintenant effectuer leurs demandes. Typiquement, cela permet de d'abord spécifier la vitesse d'exécution avant de lancer le calcul.

Cette REPL doit être implémentée sous la forme d'une invite de commande permettant d'entrer des comamndes à la syntaxe que je vous laisse définir. Tant que les conditions de votre consensus sont respectées, les commandes précédentes ne doivent pas impacter les résultats. Notez qu'à n'importe quel moment nous pouvons lancer les commandes précédentes (à l'exception de START lancée une seule fois).

Étape 3 – Command List et Stockage de fichiers (4 pts). Nous souhaitons maintenant complexifier les choses en rajoutant aux clients la possibilité d'envoyer plusieurs requêtes. Typiquement ces requêtes sont lues depuis un fichier. L'ajout de cette fonctionnalité ne doit pas altérer le concensus. Il faut donc mettre en place un concensus multi-étapes.

Ces requêtes ne seront plus de simples valeurs, mais des commandes à exécuter. L'objectif est ici d'implémenter un système de stockage de fichiers distribué, où chaque noeud réplique l'entièreté des fichiers. Les commandes qui peuvent être envoyées par les clients sont les suivantes :

`LOAD filename` demande au système de stocker un nouveau fichier, une fois le stockage effectué le système doit répondre au client avec un `UID` identifiant ce fichier

`LIST` retourne au client la liste actuelle d'`UID` des fichiers stockés.

`DELETE uid` supprime le fichier associé à cet `UID` du système.

`APPEND uid some text` ajoute le texte passé en fin de fichier, sur une nouvelle ligne.

Étape 4 – Recover (4 pts). Nous souhaitons maintenant mettre en place une commande RECOVERY pour notre REPL. Autrement dit, la possibilité pour un processus de revenir dans le système. Cette partie est la plus difficile car il faut mettre en place un protocole permettant de retrouver l'état correct du système. La moitiée des points sera afféctée à un comportement sans erreur pendant cette phase, l'autre moitié si des processus échouent pendant cette phase.

Misc (2 pts). Pour l'aspect général du projet.

Bonus (2 pts). N'attaquez les bonus que si vous avez confiance dans votre projet. Quel que soit le nombre de bonus effectués il n'y a pas la possibilité de dépasser 20 / 20.

- Implémentez une nouvelle commande permettant de faire un snapshot
- Vérifiez votre implémentation en utilisant un outil de vérification (Spot, SPIN, ...)