Projet 2 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 soumettent des jobs à exécuter aux serveurs. Ces serveurs souhaitent ensuite se mettre d'accord sur l'ordre dans lequel ils vont exécuter ces jobs, et quel noeud sera responsable de son exécution. Une fois d'accord, les jobs peuvent être exécutés sur leurs noeuds respectifs, et le résultat de chaque job peut être récupéré par le client à la fin.

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 demander à ordonnancer un job (pour l'instant ce job peut juste être un numéro, l'uid du client par exemple). Les clients vont alors soumettre leur job aux serveurs. Les serveurs se mettent alors d'accord sur quel job exécuter sur quel noeud de calcul. Dans ce premier temps chaque serveur peut simplement produire un fichier représentant l'ordonnancement de ces différents jobs sur les différents noeuds de calcul, sans pour autant les exécuter. Il est alors facile de s'assurer que tous les ordonnancements sont identiques en comparant ces fichiers.

Notez que le nombre de serveurs chargés de l'ordonnancement peut être différent du nombre de noeuds de calcul disponibles, lui même différent du nombre de clients. On peut donc se retrouver avec plus de jobs à ordonnancer qu'il n'y a de noeuds de calcul, dans ce cas les jobs restants peuvent être placés en "waiting".

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 – Exécution de jobs (5 pts). Plutôt que de simplement produire un ordonnancement possible, on va maintenant réellement exécuter les jobs. Chaque job contiendra un unique fichier C++, qui sera compilé puis exécuté sur un noeud de calcul (on assumera que `g++` est disponible sur les noeuds de calcul). La sortie standard du programme exécuté sera le résultat du job, qui pourra être récupéré par le client l'ayant soumis.

Si des jobs sont placés en waiting au début, car pas assez de noeuds de calcul disponibles, ils seront ordonnancés via un consensus puis exécuté lorsqu'un job précédent terminera, libérant un noeud de calcul. Il faut donc mettre en place un concensus multi-étapes.

Nous souhaitons aussi complexifier les choses en rajoutant aux clients la possibilité de soumettre plus d'un job à la fois ou au fil du temps. Typiquement ces requêtes sont lues depuis un fichier. L'ajout de cette fonctionnalité ne doit pas altérer le concensus.

Étape 4 – Recover (3 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.

- Gestion fine des ressources disponibles sur chaque noeud de calcul, et de celles exigées par chaque job (si un job spécifie qu'il a besoin de 8 Go de RAM au minimum, il ne peut pas être exécuté sur un noeud qui n'en possède que 4).
- Vérifiez votre implémentation en utilisant un outil de vérification (Spot, SPIN, ...)