Projet : Algorep

Rendu. Ce projet est à rendre pour le 22 novembre 2024 au plus tard à 23h42. Il s'effectue par groupes de 4 ou 5 au choix. Le rendu s'effectuera sur moodle (un par groupe). Le rendu se composera :

- du code (commenté !) ;
- d'un Makefile (idéalement, le projet doit se compiler juste en lançant "make", pas de dépendances à gérer de mon côté).
- d'un README qui contiendra, entres autres, la liste des participants de votre groupe.

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

- 15 minutes de présentation,
- 5 minutes de démo de votre projet ;
- 5 minutes de questions.

Cette présentation portera sur vos choix d'implémentation, et pourra discuter 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 commande `make` donnera lieu à une pénalité. Une absence à la présentation finale 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 sûr hors de question ici de se reposer là-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 peut être testée avec un nombre de processus différent de celui que vous utiliserez !

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. Le but de ce projet est de créer un système de tableau collaboratif. 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 modifier le tableau ("dessiner un trait de la position (x,y) à (x',y')...", "changer de couleur") Chaque serveur doit, à la fin de l'exécution, avoir le même fichier de log, et le même texte résultat. Il s'agit donc d'une forme de réplication de logs.

Étape 1 – Réplication des logs(5pts).

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 commande originale (typiquement son uid). Les clients vont alors soumettre leurs commandes aux serveurs, qui va les répliquer. Le but de cette étape est de concevoir un algorithme qui s'assure de la **cohérence** du système : tous les serveurs et clients doivent posséder le même historique/séquence de commandes, en fin d'exécution.

NB : il est bien sûr possible d'implémenter un algorithme de consensus (Raft, ou Paxos séquentiel) pour arbitrer les conflits d'ordonnancement, mais proposer un mécanisme de résolution automatique est également possible. (voir le concept de Conflict-free Replicated DataType.)

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 commandes à la syntaxe que je vous laisse définir. Tant que les conditions de cohérence 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 – Commandes d'édition (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.

Ces requêtes ne seront plus de simples valeurs, mais des commandes à exécuter. L'objectif est ici d'implémenter un système d'édition réparti, où chaque nœud réplique le même historique. Les commandes qui peuvent être envoyées par les clients sont les suivantes :

`COLOR XXXXXX` change la couleur du curseur du client.

`THICKNESS X` change l'épaisseur du trait.

`DRAW (X,Y,X',Y')` trace un segment de (x, y) à (x', y').

`CLEAR` efface le tableau.

On pourra attacher des informations supplémentaires à chaque commande, une fois stockée dans les logs (par exemple, la date, l'uid du client, etc.)

Étape 4 – Recover (3 pts).

Nous souhaitons maintenant mettre en place une commande RECOVERY pour notre REPL. Autrement dit, la possibilité pour un processus ayant un "CRASH" 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 moitié des points sera affectée à un comportement sans erreur pendant cette phase, l'autre moitié si aucun processus n'échoue pendant cette phase.

Misc (3 pts).

Pour l'aspect général du projet.

Bonus (2 pts).

- Réalisez une interface (GUI, web) d'édition
- Proposez une méthode de vérification de vos protocoles (en utilisant Spot, SPIN, ...)