Programmation répartie

PRR 2021 / Labo 4

Temps à disposition : 8 périodes.
Travail à réaliser par groupe de 2 étudiants (pas forcément les mêmes que pour les labos précédents, mais en en informant l'enseignant dans ce cas).
Un étudiant qui effectuerait le labo seul (avec justification et après autorisation explicite de l'enseignant) n'aura à rendre qu'une des deux parties demandées.
La présence aux labos est obligatoire.
En cas de copie manifeste entre les rendus de deux labos, tous les étudiants des deux groupes se verront affecter la note de 1.

Labo distribué le jeudi 23 décembre à 16h35. Vous réutiliserez le repo github ***privé*** *dont vous donnerez l'accès à l'enseignant et à l'assistant (comptes* ***patricklac*** *et* ***raphaelracine*** *sous github) et dont vous leur communiquerez l'adresse par email avec le nom des étudiants du groupe. Vous devrez pousser votre travail en l'état sur ce repo au moins une fois par semaine, en début de chaque séance de labo.*

A rendre le jeudi 27 janvier 2022 à 16h35. *Un readme indiquant comment utiliser votre programme et précisant ce qui a été réalisé, ce qui fonctionne et ce qui reste à faire, devra être déposé dans votre repository. Un email rappelant vos noms et repository confirmera votre rendu qui sera accessible dans la branche master (ou main).*

Objectifs

- Comprendre le fonctionnement de paradigmes de programmation dans un réseau réparti.
- Réaliser des communications UDP unicast en GO.
- Gérer la concurrence d'accès aux variables avec des goroutines et des channels en GO.

Introduction

Le labo est composé de 2 parties qui permettent chacune de déterminer quel est le plus court chemin entre 2 nœuds dans un graphe quelconque (pas forcément une arborescence) de 2 façons : pour la première partie, en utilisant un algorithme synchrone (ondulatoire) et pour la deuxième partie, en utilisant un algorithme asynchrone (sondes et échos).

Description commune aux 2 parties

Pour les 2 parties, une fois le réseau réparti établi et sur une demande initiale, il s'agit d'obtenir par échange de messages entre voisins le nombre minimum de nœuds à traverser pour atteindre les autres nœuds et le numéro de leur voisin immédiat sur le chemin le plus court pour les joindre. En cas d'existence de plus courts chemins de même longueur pour atteindre un même nœud, un chemin quelconque sera retenu parmi ceux-ci.

Les communications se feront en mode non-connecté (UDP unicast). On supposera le réseau stable et sans erreurs : pas de perte de message ni de panne de serveur.

Un fichier de configuration contiendra la description du réseau : nombre de processus serveur, leurs numéros, adresses et numéros de port, ainsi que la description du graphe qui relie les serveurs (liste des numéros des voisins immédiats de chaque nœud). Chaque serveur devra pouvoir être démarré avec son numéro en paramètre.

Une application client permettra de démarrer le processus de recherche. On ne fera pas de nouvelle demande tant qu'une recherche précédente n'est pas terminée : il n'est pas demandé de gérer de recherches simultanées.

A l'issue d'une recherche, chaque serveur affichera dans sa console sa connaissance des plus courts chemins pour atteindre chaque autre nœud. L'application client n'a rien à afficher.

Il n'est pas demandé de test automatisé mais votre readme contiendra la description de tests manuels et des résultats obtenus.

Comme pour les labos précédents, il sera tenu compte dans chaque partie de la qualité de votre code source (packages, …), de sa conformité aux bonnes pratiques du langage GO, et aussi de la concurrence d'accès avec des goroutines et des canaux, et non avec des verrous.

Chaque partie sera valorisée pour 25 points, 15 points pour son bon fonctionnement et 10 points pour la qualité de votre rendu (documentation, tests, code source).

Première partie : algorithme ondulatoire

Une fois le réseau établi (pas de contrôle demandé), l'application client permettra de déclencher une recherche de plus court chemin simultanément sur chaque nœud du réseau (par message UDP direct, sans passer par le graphe).

Chaque serveur participera alors à des vagues successives selon l'algorithme ondulatoire jusqu'à obtenir la liste des plus courts chemins vers chaque nœud (nombre de nœuds traversés, nœud suivant sur le chemin) et l'affichera. On suppose qu'il ne connait pas le « diamètre » du réseau : il s'arrêtera après avoir obtenu la liste complète des plus courts chemins selon l'algorithme expliqué en cours.

Deuxième partie : algorithme sondes et échos

Une fois le réseau établi (pas de contrôle demandé), l'application client permettra de déclencher une recherche de plus court chemin sur un nœud de numéro saisi par l'utilisateur. Il transmettra un message de demande au seul serveur correspondant.

Ce serveur initial activera alors une recherche selon l'algorithme de sondes et échos jusqu'à obtenir la liste des plus courts chemins vers chaque nœud (nombre de nœuds traversés, nœud suivant sur le chemin) et l'affichera.

De plus chaque nœud traversé par la recherche affichera également sa connaissance partielle des plus courts chemins lorsqu'il l'aura constituée (au moment où il la transmettra à son « parent »).

L'algorithme implémenté permettra de traiter sans boucler les cycles dans le graphe, selon l'algorithme expliqué en cours.