Rapport projet RSA : Friedman Distributor

Jean-Philippe Eisenbarth et Valentin Gianinni

Présentation du sujet

Dans l'optique de vouloir calculer des nombres de Friedman de manière rapide, nous avons mis au point une solution pour y parvenir. Cette solution est qu'un serveur distribue des nombres à des clients afin qu'il calcul si le nombre est un nombre de Friedman ou non. Un nombre de Friedman (ou nombre autodigitaux) est un nombre qui est égale à une expression arithmétique qui combine chaque chiffre de manière à ce que le résultat soit égal ce nombre. Par exemple 25 est le premier nombre de Friedman car 25=5^2. Vous pouvez voir que dans l'expression arithmétique il y a les chiffres du nombre. Ces chiffres peuvent êtres assemblés ensembles, ou alors séparés par des opérations que sont l'addition, la soustraction, la division et la puissance (il est possible d'ajouter d'autres opérations comme des racines carrés, …). La vérification pour savoir si le nombre est un nombre de Friedman est assez complexe et coûteuse en temps et en CPU, c'est pour cela que nous la distribuons. L'algorithme de vérification prend en entrée un nombre, ensuite c'est le serveur qui va le fournir au client qui le fournira lui même au script python. (Pour des raison de temps et de technique nous n'avons pas implémenté l'algorithme en entier, nous avons une première version en Ruby mais nous avons trouvé une version complète en Python sur Internet)
Grâce à ce système de serveur multi-clients, il est possible de distribuer le calcul des nombres de Friedman et de collecter les résultat de chaque client.
Peut-être allons nous en découvrir de nouveau…

Modélisation

Pour réaliser ce sujet nous avons réfléchi sur la manière dont on pourrait le concrétiser. D'un coté nous devons avoir un serveur qui distribue des nombres et gère des clients. Et de l'autre coté un client qui calcul et renvoie un résultat au serveur.
Le coté serveur se décompose en deux parties, à savoir une partie réseaux qui gérera les sockets, les clients, l'activité des clients et également les pertes de clients n'ayant pas fini les calcul. La deuxième partie est un programme qui génère des nombres, ici séquentiellement, les donne au serveur et récupère les résultats pour les enregistrer dans un fichier.

Dans la partie client il y a une partie réseaux également qui est chargé d'écouter le serveur et d'échanger avec lui. Nous avons aussi une partie algorithmique (dans notre cas un script python) qui calculera de manière récursive des solutions pour un nombre donné.

Nous avons mis en place un protocole avec des *APDU* afin que les dialogues clients-serveurs soient réglementés.

Tout d'abord le client doit se présenter au serveur en demandant une connexion avec la commande "CNX". Le serveur répondra par "COK" si la connexion est accepté, sinon il enverra un message d'erreur.
Une fois le client connecté, celui-ci lance un thread qui s'abonnera sur un canal multicast. Ce canal permet au serveur de vérifier que les clients sont toujours en activités par le biais de *PING*/*PONG*. Le serveur envoie de manière périodique sur la canal la commande *PIN*. Les client recevant un *PIN* répondrons pas *PON* via le socket de dialogue. Le serveur mettra alors à jour la date de la dernière réponse au *PING*. Si un client n'a pas répondu depuis un un certain temps défini alors il sera supprimé de la liste des clients et le nombre qui lui était attribué sera attribué au prochain client.

Pour demander un nombre, le client envoie la commande "GET", le serveur répond avec un "NBR" suivie du nombre choisi. Le client calcul s'il s'agit d'un nombre de Friedman et renvoie une réponse avec la commande "RES" suivi du nombre ainsi que de la solution séparée avec ":". S'il n'y a pas de solution, alors un # sera envoyé à la place du champ de solution.

Lorsqu'un client veut quitter le serveur, il envoie la commande "DNX" et le serveur renverra un "DOK" pour dire qu'il a bien été supprimé.

Maintenant que la conception et les ADPU sont fait, place au code.

Conception

Le serveur utilise la fonction *select* pour accepter les nouveaux client et dialoguer avec les anciens. Un thread est lancé pour envoyer des *PING* en multicast à tout les clients.
Le client utilise une connexion TCP pour le dialogue avec le serveur, cette connexion permet de demander des nombres et d'envoyer les résultats. Il utilise également un socket multicast en réception pour réceptionner les *PING* du serveur, il répond via le socket TCP de dialogue.

Le code est conçu en partie de manière générique, c'est à dire que le générateur de nombre peut être changé facilement par d'autres générateurs (nombre aléatoire,chaîne de caractères, …) Et le script python peut également être remplacé par un autre pour faire d'autre calcul comme chercher si un nombre est premier, chercher le hash d'une chaîne de caractère et le comparer à un autre, faire du bruteforce sur des logiciels ou site web, …

Le serveur assure également le calcul des données. Il associe le nombre donné par le générateur au client, si celui-ci meurt et qu'il n'y a pas de résultats alors le nombre sera donné au prochain client qui en réclame un. Ainsi le code du générateur est plus simple car il ne gère pas les pertes de clients.

Utilisation

Un makefile est disponible pour compiler, il suffit de faire “make” pour compiler les client et le serveur. Sinon faire “make client” pour compiler le client et “make server” pour le serveur.

Pour lancer le serveur :
./bin/Server PORT
Port est le numéro du port TCP utilisé pour le lien avec les clients
Pour lancer le client
./bin/Client IP PORT
Avec l’IP et le PORT du serveur

Par défaut le client fera 5 calculs, il est possible de changer ce nombre en lançant :
./bin/Client IP PORT -n N
avec N le nombre de calcul à faire, -1 si une infinité