Compte-Rendu TP4 Application Réseaux :

Serveur simple :

Si deux client veulent se connecter au même serveur, le deuxième client ne peut pas se connecter puisque le serveur n'attend pas de connexion.
Pour l'instant le serveur n'attend qu'une seule connexion puis gère la communication avec ce client.

Solution ? :

Pour que le serveur puisse gérer plusieurs connexions, il faudrait qu'il attende plusieurs fois une connexion client. Cependant l'attente d'une connexion est bloquante et si un client envoie des données au serveur pendant ce temps, elles ne seront pas prisent en compte.
Pour palier à ce problème on peut créer après chaque connexion un thread gérant indépendamment des autres threads la communication avec le nouveau client.
Le serveur doit admettre une boucle infini dans laquelle le serveur attend une connexion puis lance un thread gérant ce client (puis ferait une nouvelle itération de la boucle).

Thread :

Test de Performance 1 :

Lorsque Stress1 ne ferme pas les clients, les connexions n'arrivent pas dans l'ordre de Stress1. Plus le nombre de clients stressants à connecter est grand, plus les connexions sont désordonnées.

Lorsque Stress1 ferme les clients, les connexions arrivent toutes dans l'ordre et ce quelque soit le nombre de clients stressants à créer (observé jusqu'à 5000 clients stressants).

On peut donc dire qu'il y a une différence entre si les clients sont fermés ou non. Les connexions arrivent dans le désordre si les clients ne sont pas fermés, tandis que si ils le sont les connexions arrivent dans l'ordre et toutes les connexions se font bien plus rapidement. (~160 millisecondes si les clients sont fermés contre ~1150 millisecondes sinon pour 100 clients stressants).

Pool de Threads :

Les pool dynamiques ont deux constructeurs, le premier Executors.newCachedThreadPool() qui renvoie un Executors qui utilise le ThreadFactory par défaut (c'est la manière de créer un thread), le second qui prend en paramètre un ThreadFactory qui renvoie un Excutors utilisant le ThreadFactory donné en paramètre.

Pour les pool statiques il existe de la même manière deux constructeurs pour les même raisons (ici ces constructeurs prennent en paramètre un nombre de thread pouvant être créer, et potentiellement un ThreadFactory).

Test de Performance 2 :

L'utilisation des pool dynamiques ralentit légèrement le serveur, plus le nombre de clients augmente plus le serveur ralentit.
Cependant son utilisation de la mémoire est amoindrie et il pourra accueillir plus de clients sur une longue durée (comme les threads ne se fermait pas et que leur nombre sont limité au bout d'un moment le serveur n'aura pas pu créer plus de threads)

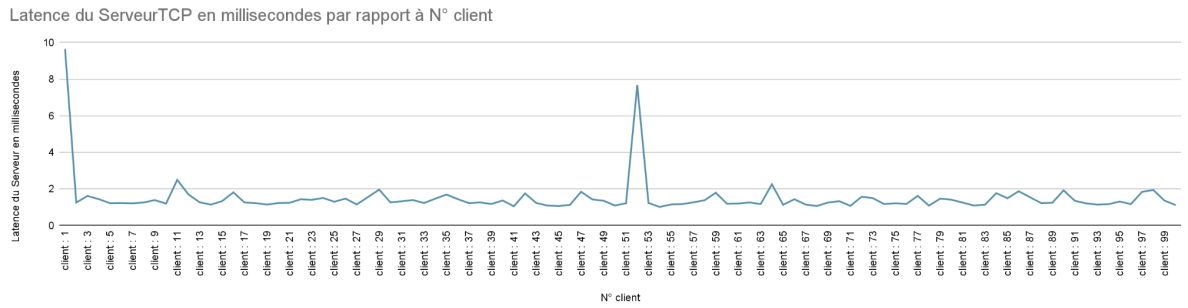
Test de Performance 3 :

L'utilisation des pool statiques ralentit le serveur en fonction du nombre maximal de thread utilisé, le temps de réponse du serveur est (~)constant.

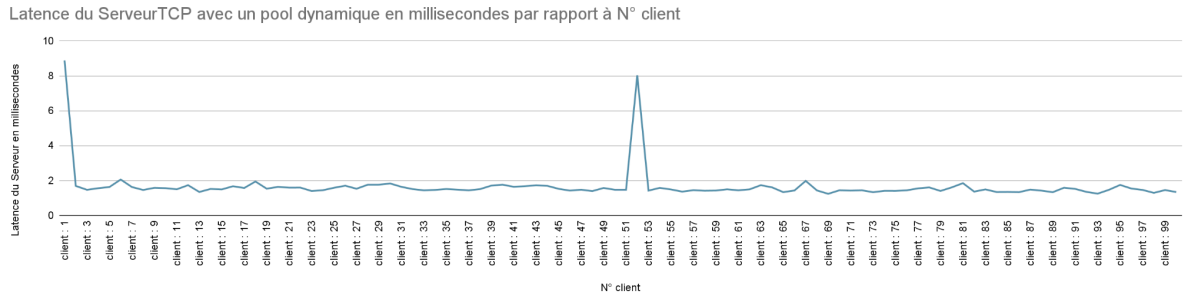
Test de Performance 4 :

Bilan :

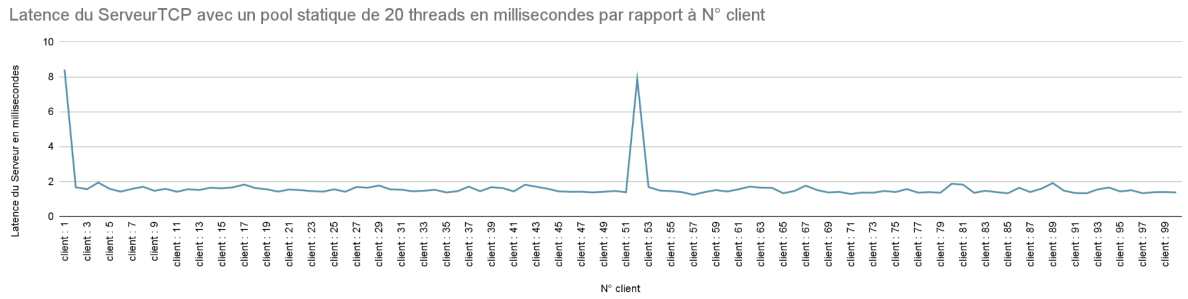
Le serveur ne réutilisant pas les threads est le plus rapide, cependant il est le plus instable et n'est pas optimisé en mémoire n'y sur la durée.



Le serveur avec un pool dynamique est le 2ème plus rapide, optimise la mémoire en fonction du nombre de clients.



Le serveur avec un pool statique est le 3ème plus rapide, pour une mémoire limitée il est le plus adapté (limitation par les threads du nombre de clients).



Le serveur avec un pool voleur est le plus lent.