POLYTECHNIQUE
MONTRÉAL

LE GÉNIE
EN PREMIÈRE CLASSE

INF4410-Systèmes répartis et infonuagique

Travail Pratique 2

Réalisé par :

Messaoud Fazil 1605802

Thierno Barry 1550237

Remis le 11 novembre 2016

Introduction :

A travers ce travail pratique nous avons eu à implémenter un système réparti composé d'un répartiteur de taches et de plusieurs serveurs de calculs. Le répartiteur a pour tâche d'analyser et de repartir correctement aux serveurs une taches composé de plusieurs opérations mathématiques que les serveurs de calculs doivent effectuer. Les serveurs une fois que leurs calculs sont terminé retournent les résultats au répartiteur. Ce type de système est un système de type Serveurs-Clients. A travers ce rapport nous allons expliquer les différents choix d'implémentations que nous avons pris afin de rencontrer les exigences décrites dans l'énoncé.

Détails d'implémentation

Serveur de calcul

Les serveurs sont créés en passant en paramètre, lors de leur création, le port qui va être utilisé, le nombre maximal d'opérations que le serveur peut accepter dans une tache et le pourcentage de faux résultat qu'il devras retourner.

Avant d'exécuter une tache le serveur appelle la fonction acceptTask() afin de voir si il peut se permettre d'accepter la tache ou si le nombre d'opérations est trop élevé. Nous avons décidé d'implémenter cette fonction comme elle décrite dans l'énoncé et de retourner True si le taux obtenue est inferieur a 0,5. Le serveur vérifie également si le taux de résultats malicieux est inférieur au taux mis en place par le paramètre obtenue lors de la création du serveur. Si le taux est inférieur au paramètre, le serveur appelle la fonction executeFakeOperations() de la classe Compute, qui retourne un tableau d'entier aléatoire de la taille du tableau des opérations que le serveur aurait dû effectué.

Le serveur reçoit du répartiteur un tableau de chaine de caractère contenant les opérations a effectué. Dans le cas normal, c'est-à-dire ou le serveur n'est pas surchargé et qu'il décide de renvoyer les bons résultats, il instancie la classe Compute en lui passant en paramètre la liste des opérations. Puis le serveur appelle la fonction executeOperations() qui effectue les opérations que le serveur a reçu.

Classe Compute

La classe Compute est une classe ayant pour rôle d'appeler la bonne méthode d'après le nom de l'opération contenue dans le tableau des opérations reçu en paramètre lors de son instanciation.

Classe DistributorSecure

La classe DistributorSecure joue le rôle de distributor dans le cas où le client est lancé en mode secure. Lors de son instanciation on exécute la fonction run(), elle s'occupe de créer le bon nombre de thread d'après le nombre de serveur instancier et d'appeler la fonction loadServerStub() sur chacun des serveurs.

Chaque thread représente donc l'exécution d'un serveur, ces threads s'arrête lorsque le curseur se déplaçant à travers la liste des opérations a effectué soit égale aux nombres d'Operations en question. Chaque thread a un accès concurrent a la liste des opérations afin de construire un

ArrayList contenant une liste d'opération construite en fonction du nombre d'opérations maximales supposé pour le serveur en question. Ensuite le thread envoi la tâche à travers l'objet de type Calculator. Si le résultat de la requête est null, cela signifie que soit le serveur est en panne, alors la tache construite précédemment est réinséré dans la liste des opérations a effectué. Soit le serveur refuse de traiter la tâche du au nombre d'opérations trop élevé, dans ce cas on divise le nombre d'opération maximal supposé pour ce serveur par 2 puis on réinsère la tache dans la liste des opérations. Si le serveur nous retourne un résultat, celui-ci est stocké dans la liste des résultats. Le résultats finale est calculé dans cette classe à partir de la liste des résultats qui est un attribut global de cette classe.

Classe DistributorNotSecure

Cette classe est appelée dans le cas où le client est lancé en mode not secure. Elle est très similaire a la classe précédente, sauf qu'ici dans le thread d'exécution de chacun des serveurs, on envoie deux requêtes vers deux serveurs différents contenant la même tache. Le 1er serveur est le serveur du thread s'exécutant tandis que le 2em est sélectionner aléatoirement. Si les résultats retournés ne concordent pas, la tâche est réinsérée dans la liste des opérations.

Classe Calculator

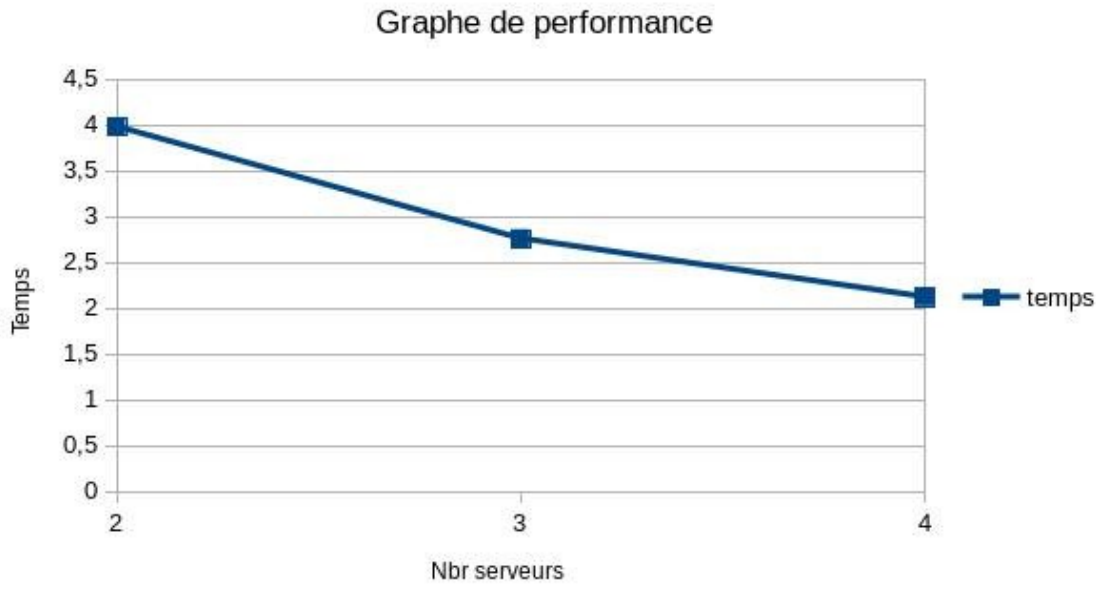
Cette classe représente un serveur de calcul du coté client. Elle contient plusieurs attributs décrivant le comportement du serveur distant en plus de contenir la méthode pour envoyer une tache au serveur.

Classe Client

Cette classe est la porte d'entrée du programme client. Elle lit les fichiers contenant les opérations a effectué et le fichier contenant les propriétés des serveurs distant (adresse ip et port). Puis de là on accède soit à la classe DistributorNotSecure ou DistributorSecure d'après les paramètres reçus.

Tests de performance – Mode sécurisé

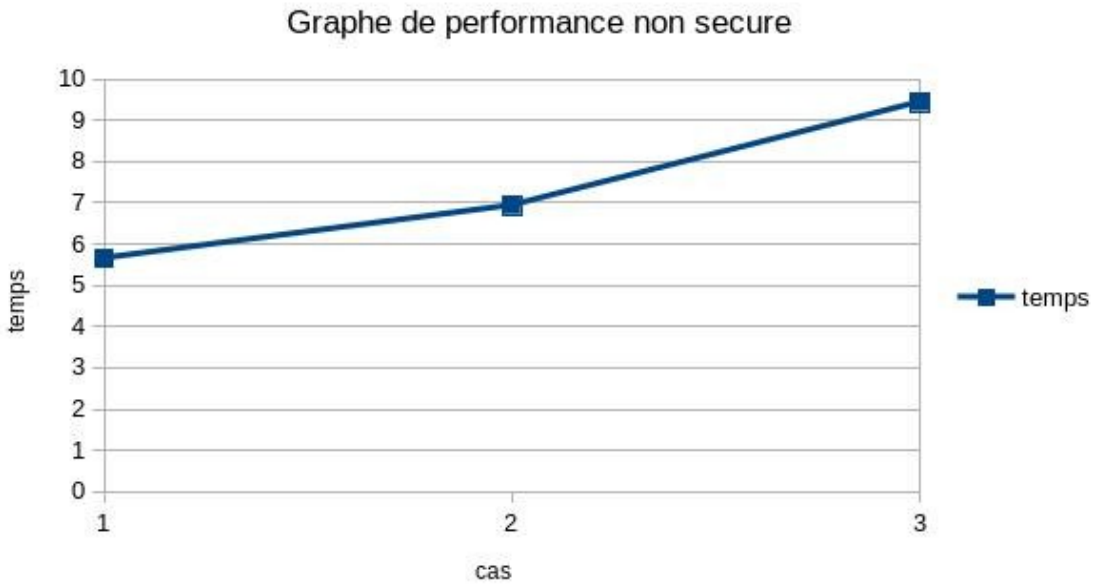
Afin d'obtenir le graphique du temps d'exécution en fonction du nombre de serveurs de calculs nous avons utilisé la fonction time dans linux lors de l'appel de notre client. Nous avons pris les valeurs real que time retourne à la fin de l'exécution du client. Voici le graphique que nous avons obtenu :



Le graphique obtenue nous confirme que l'utilisation de plusieurs serveurs de calculs permet de diminuer le temps d'exécution du client. Donc la répartition des opérations est globalement efficace. On remarque que le temps d'exécution ne diminue pas autant entre 3 et 4 serveur qu'entre 2 et 3 serveurs même si le nombre d'opérations que le 4em serveur accepte est plus grand. Cela est dû au fait que c'est le même document qui a été utilisé comportant 2000 opérations à réaliser. S'il y aurait plus d'opération la différence se verrait plus.

Test de performance – Mode non-sécurisé

Nous avons effectué nos tests de la même manière que le mode sécurisé mais en respectant les contraintes d'avoir des serveurs malicieux. Voici le graphique que nous avons obtenue pour les trois cas :



On voit que le temps d'exécution augmente même si le nombre de serveurs de calcul demeure le même. On peut conclure donc que le fait d'avoir des serveurs malicieux et devoir vérifier les résultats retournés par les serveurs fait augmenter le temps d'exécution du client.

Question

Afin de rendre d'améliorer la résilience du répartiteur on pourrait instancier plusieurs répartiteurs sur différentes machines. Cela pourrait être par exemple sur les machines avec les serveurs. Il faudrait également prévoir que pour chaque modification du répartiteur opérant il faudrait copier son état vers les autres instances des répartiteurs afin de garder le système cohérent. Il faudrait également prévoir un système de redirection du répartiteur opérant (répartiteur principal qui exécute la fonction pour repartir les opérations) vers une autre instance de répartiteur.

L'avantage de cette architecture est d'avoir plusieurs instances de répartiteurs cohérent vers qui les serveurs peuvent retourner leurs résultats au cas où le principal subirait une panne. Le principal inconvénient est la forte consommation de ressource réseaux afin d'envoyer les données du répartiteur principal vers les autres. De plus on peut toujours subir une panne si le code du répartiteur est bugée ou si les données ne se réplique pas correctement ce qui mènerais à un état inconsistant du système. Cette solution sert principalement a diminué le risque de panne plutôt que de l'éliminer.