INF4410 – Systèmes répartis et infonuagique

TP2 – Services distribués et gestion des pannes

Présenté à :

Houssem Daoud

Travail réalisé par :

Gabrielle Bourdages - 1579702

Xavier Coupal - 1584246

Alexandre Rose - 1580973

École Polytechnique de Montréal

16 novembre 2015

Choix de conception

Répartiteur

Le répartiteur commence son travail en toute simplicité. Il reçoit une liste d'opération interprète et place dans une file. Par la suite, le répartiteur ouvre une connexion avec chaque serveur de calcul défini dans le fichier de configuration puis démarre un fil d'exécution par serveur. Ces fils d'exécutions sont en charge d'envoyer des opérations et de collecter les résultats. Puisque l'ordre des opérations n'a aucune importance, chaque fil d'exécution prend un nombre *n* d'opérations dans la file et les envoie à son serveur de calcul. S'il advenait que le serveur de calcul, trop occupé, refusait l'opération, les opérations sont conservées et renvoyées jusqu'à ce qu'elles soient acceptées.

Afin de trouver la taille optimale des opérations pour chaque serveur de calcul, nous utilisons un algorithme d'apprentissage par renforcement qui augmente ou diminue la taille *n* des opérations envoyées en fonction d'un refus ou d'une acceptation.

Dans le cas d'une panne, le fil d'exécution du serveur fautif prendra fin et remettra ses opérations non complétées dans la file d'opération. Nous nous assurons de réactiver les autres fils d'exécutions s'ils avaient déjà complété leur travail.

Pour s'assurer qu'il n'y ait pas de problème de concurrence, nous avons utilisé une ConcurrentLinkedQueue qui permet d'effectuer des opérations de manière atomique et sans risque d'interbloquage.

Lors de la fin de l'exécution des opérations, on additionne les résultats partiels collectés des différents serveurs de calculs et on obtient le résultat final.

Pour ce qui est du mode malicieux, on sait que chaque tâche doit être exécutée par tous les serveurs pour valider si une réponse peut être prise en compte. Ainsi, on ne peut pas être aussi efficace en divisant les tâches parmi tous les serveurs de calculs. On commence donc par envoyer toutes les opérations en bloc à tous les serveurs. Lorsqu'on a le résultat total de tous les serveurs de calcul, on collectionne les résultats dans une mappe dont la clé est le résultat obtenu et la valeur est le nombre d'occurrence de ce résultat parmi les serveurs. Si on a un résultat qui est retourné par plus de la moitié des serveurs, on le conserve. Notre répartiteur ne fait rien en particulier si aucune réponse est retenue puisque les cas de testent sont toujours assurés d'avoir une réponse correcte (un seul serveur malicieux sur trois).

Advenant une panne d'un serveur en plein calcul pendant que nous sommes en mode non-sécurisé, il se pourrait qu'on obtienne un résultat de -1 (réponse retournée si aucune réponse n'est jugée fiable).

Serveurs de calcul

Pour simuler les ressources des serveurs de calcul, nous utilisons la formule fournie avec l'énoncé : on tire un nombre aléatoire entre 0 et 100, et si ce dernier se retrouve dans l'intervalle de rejet (calculé en fonction de la taille de l'opération demandée et la capacité définie dans le fichier de configuration) on n'exécute pas les opérations et on lance une exception qu'on peut attraper et gérer du côté du répartiteur.

Les serveurs de calcul acceptent un argument qui indique l'indice de malice. Il s'agît d'un nombre entre 0 et 100 qui indique la chance que le serveur renvois une réponse erronée. Comme pour le taux de rejet,

on fonctionne avec un nombre aléatoire et s'il tombe dans l'intervalle considéré malicieux, on ajoute un nombre aléatoire à la réponse obtenue lors du calcul.

Question 1

Comme nous n'avons qu'un seul répartiteur qui a la lourde responsabilité de répartir toutes les requêtes, on doit trouver une alternative à cette répartition des tâches qui est centralisée. Une solution permettant d'éviter d'avoir un répartiteur central et unique serait de distribuer la tâche de répartition entre les différents serveurs de calcul. Premièrement, le client enverrait la liste complète des opérations à effectuer à un des nœuds de calcul, lequel serait identifiable par DNS et pourrait être caché derrière un proxy inverse, ce qui permet de cacher l'architecture distribuée de l'application. Une fois la liste d'opération envoyée, le premier serveur de calcul en enverrait une copie complète à chacun des autres serveurs. Ensuite, les serveurs utiliseraient un protocole pour communiquer et se diviser la tâche, par exemple en sous sections. Il serait aussi possible de procéder à l'élection d'un répartiteur dynamiquement, lequel serait en charge d'attribuer des sections à calculer à chacun des autres nœuds. De cette manière, une panne d'un des serveurs serait remédiable facilement, les tâches seraient redistribuées et un nouveau répartiteur serait élu. De plus, il serait possible pour les nœuds de s'échanger les réponses partielles obtenues et ainsi réduire le temps qu'il faudrait pour se rétablir d'une défaillance.

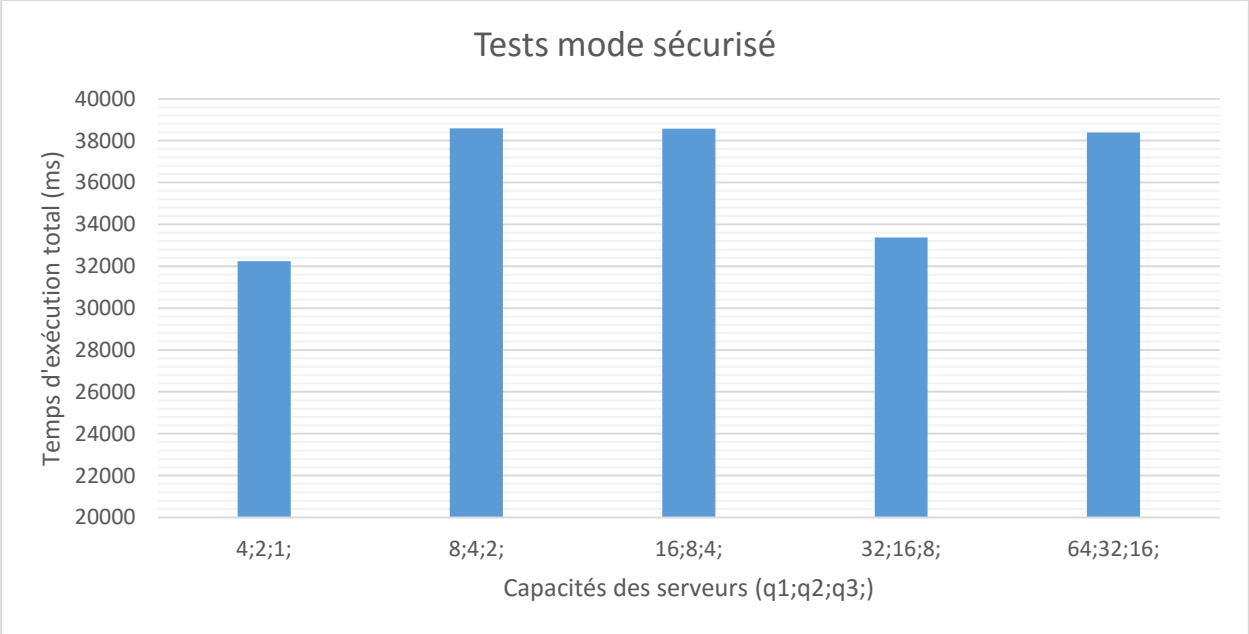
L'avantage principal de cette architecture sont bien sur l'élimination d'un répartiteur unique, éliminant un point de défaillance à notre système distribué. Toutefois, il serait toujours possible que le résultat ne soit pas calculé s'il advenait que le premier serveur connaisse une défaillance avant l'envoi complet des opérations aux autres nœuds ou encore s'il commettait une erreur en les transmettant. De plus, cette architecture nécessite beaucoup plus de synchronisation entre les différents serveurs et dépend du transfert de beaucoup plus d'information, ce qui pourrait réduire légèrement les performances. Il faudrait aussi mettre en place un système qui permettrait au client de se reconnecter au service et de récupérer sa réponse si jamais le nœud avec lequel il a ouvert une connexion se déconnectait avant la fin des calculs, sans quoi notre point de défaillance ne serait que déplacé.

Finalement, il est aussi possible que cette solution échoue si tous les nœuds font défaut en même temps, ou encore s'il reste plus de nœuds malicieux que de nœuds sains dans le cas d'un environnement non-sécurisé.

Résultats des tests de performance

Mode sécurisé

Pour nos tests en mode sécurisé, nous avons commencé par tester avec des petits nombres pour la capacité des serveurs, pour ensuite les augmenter. Nous nous sommes toujours assuré que la capacité du premier serveur était le double du second et que celui-ci était le double du troisième. On peut voir sur le graphique que lorsque la capacité des serveurs est la plus basse, c'est-à-dire avec une capacité de 4, 2 et 1, le temps d'exécution est le plus rapide. Par contre, il n'y a pas une grosse différence entre le plus long et le plus cour. Le temps d'exécution augmente lorsqu'on augmente la capacité des différents serveurs, mais pas de façon linéaire. Effectivement on peut voir que lorsque la capacité des serveurs est de 32, 16 et 8 on obtient le deuxième temps le plus rapide. Ainsi, les différentes capacités des serveurs n'influencent pas le temps d'exécution finale. On peut donc déduire que la communication entre les *workers* et le répartiteur est très rapide est influence très peux le temps d'exécution. La combinaison de faible latence et haut taux de transfert fait en sorte la pénalité pour une requête refusée est négligeable face au temps des opérations. Les différences entre les résultats pourraient s'expliquer par l'ordonnancement aléatoire des tâches, en particulier lors de la dernière série d'envoi.



Mode non-sécurisé

Pour les tests en mode non-sécurisé, puisque la capacité influençait très peu le temps d'exécution, nous avons utilisé les capacités suivantes : 16, 8 et 4. Nous arrivions à avoir un léger taux de refus sur tous les serveurs. Ensuite, nous avons rajouté le taux de malice à un serveur à la fois. On peut voir qu'il n'y a aucune différence entre les trois possibilités de serveur malicieux. Ce résultat est attendu puisque le répartiteur doit attendre le résultat final des trois serveurs. Les résultats prennent donc plus de temps qu'en mode sécurisé puisqu'il n'y a pas de partage de tâche entre les serveurs à cause du serveur malicieux, éliminant ainsi le gain en performance du système distribué.