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 qu’il sépare également entre tous ses serveurs de calcul. Cependant, comme l’énoncé le précise, la capacité des serveurs de calcul peut faire en sorte que l’un d’entre eux refuse une tâche lui ayant été envoyé. Nous devons donc répartir à nouveau les opérations qui se sont faites refusées. La technique pour laquelle nous avons optée est relativement simple : Si un serveur refuse une tâche, on la redistribue également sur tous les serveurs. Par exemple, si le serveur A (parmi A, B et C) refuse 9 opérations, on renvois en groupe de 3 opérations à A, B et C. le serveur A a plus de chances d’accepter une tâche plus petite, et on assume que les deux autres serveurs peuvent accepter facilement des petites tâches.

Lorsqu’on reçoit une « RemoteException » (qui peut être lancée lorsqu’on perd la connexion à un serveur de calcul), on redonne simplement la tâche à un autre serveur. Le serveur qui est déconnecté pourrait se retrouver à recevoir d’autres requêtes si un autre serveur refuse à cause de sa capacité, mais chacune des tâches lui étant envoyée sera toujours redistribué au serveur suivant (suivant un modulo du nombre de serveur pour ne pas donner la tâche à un serveur qui n’existe pas bien sûr).

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

## Serveurs de calcul

Le moyen que les serveurs de calcul ont pour refuser une tâche est simple : on tire un nombre aléatoire entre 0 et 100, et si ce dernier se retrouve dans l’intervalle de rejet (calculé selon la formule donnée dans l’énoncé) 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. La façon dont est réalisé le laboratoire est difficile à changer puisque nous n’effectuons pas des requêtes vers le répartiteur. Nous exécutons directement l’instance du répartiteur en lui passant nos paramètres en entrée. Si nous imaginons plutôt que notre architecture permet de se connecter au système via des requêtes provenant de l’extérieur, nous pouvons facilement penser à des alternatives qui permettraient d’éviter que le répartiteur devienne le maillon faible du système. Une alternative pourrait être de

# Résultats des tests de performance

## Mode sécurisé

## Mode non-sécurisé