TP2 – Services distribués et gestion des pannes

INF4410 - Systèmes répartis et infonuagique - Michel Dagenais

POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

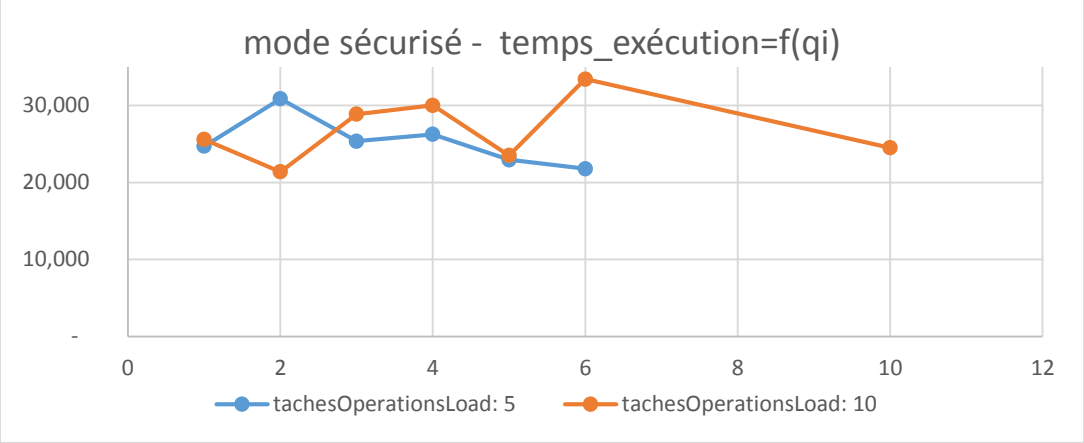
Antoine Giraud – #1761581

Le 10 novembre 2014

I. Test de performance

Pour les tests, en modifiant chaque fichier pour chaque cas nous avons sorti les temps globaux de calcul lors de chaque sucureCompute ou nonSecureCompute. Il a fallu se connecter sur différentes machines des labs de Poly en ssh : ssh login@Ip.serveur.polymtl.ca

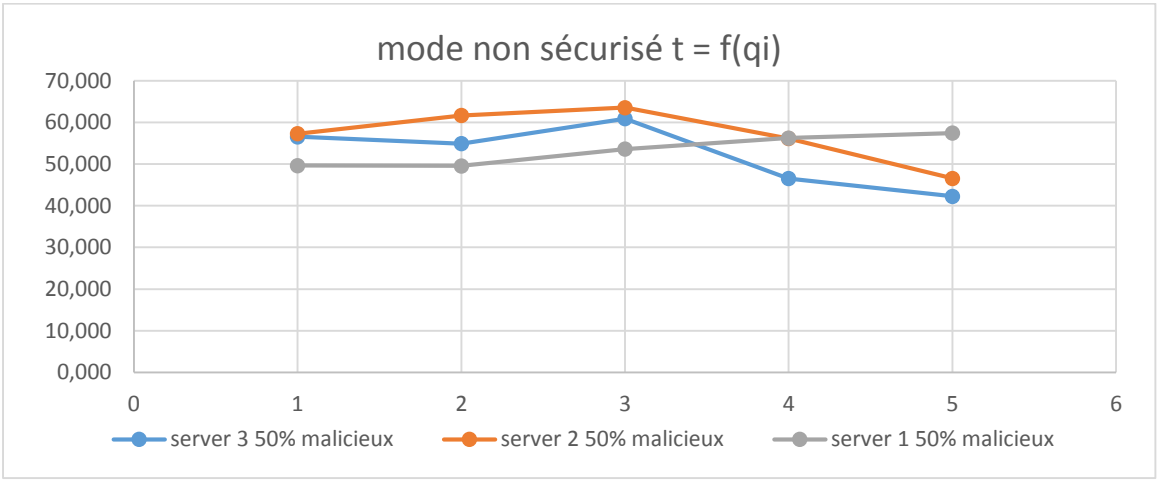
A. Mode sécurisé



En mode sécurisé, la taille de qi n'a que peu d'influence. On voit que l'on reste dans les mêmes ordres de grandeurs. Et ce que ce soit avec une taille de tâche de 5 ou 10.

Les différences de valeurs sont dues à divers facteurs comme des moments de ralentissement sur le réseau, un utilisation de cette machine précise à un moment, problème de connexion, …

B. Mode non-sécurisé



On remarque qu'en moyenne les temps sont au moins deux fois plus long qu'en mode sécurisé. Ce qui est compréhensible de par le fait que l'on a au moins deux fois plus de calculs à faire. On a choisi ici des tâches de taille 5.

II. Question de réflexion

Le système distribué tel que présenté dans cet énoncé devrait être résilient aux pannes des serveurs de calcul. Cependant, le répartiteur demeure un maillon faible. Présentez une architecture qui permette d'améliorer la résilience du répartiteur. Quels sont les avantages et les inconvénients de votre solution? Quels sont les scénarios qui causeraient quand même une panne du système?

Une première idée serait pour chaque serveur de calcul de garder pendant un temps les valeurs qu'il aura calculé pour tel serveur d'un côté le temps que le serveur répartiteur confirme qu'il a bien fini avec cette tâche. Cela pourrait permettre si le serveur répartiteur tombait de recommencer et de ne pas recalculer ce qui l'a déjà été. Le problème majeur avec cela est que l'on fait occuper des ressources inutiles sur les machines de calcul qui ne sont pas forcément optimisé pour cette tâche.

Une autre solution pour assurer que l'on puisse en tout temps répondre au client serait d'avoir une élection parmi les serveurs de calculs et avoir au cas où le serveurRépartiteur tombe un serveur de Calul qui se transforme en serverRépartiteur. Le problème avec cette solution est d'une part toujours la même que pour la première solution à savoir ne pas avoir une machine optimisé pour son job. L'autre problème demeure en la sauvegarde des données.

Au final, un bon compromis pourrait être de redonder le serveur répartiteur en lui associant une machine esclave qui suit tout ce qu'il fait et réplique sur son propre journal les opérations identiques à celle du serveur maitre. Si jamais le serveur maitre ne répond plus, le serveur Esclave après s'être assuré que son maître est vraiment disparu, il va par précaution lui couper le courant pour être vraiment sûr qu'il sera le seul maître. Au retour de l'ancien serveur maître, on pourra alors manuellement ou par une procédure définie refaire passer l'ancien maître maître.

III. Retour sur le développement :

La structure est toujours similaire à celle du premier TP avec trois package :

- serverCalcul où l'on retrouve la classe pour le serveurCalcul.
- serverRepartiteur où l'on retrouve la classe pour la gestion de la répartition du travail en tâches d'opérations à envoyer, via la classe gérant les stubs, aux serveurs de calcul. On y retrouve aussi deux objets Callable pour gérer la gestion de l'envoie des tâches au serveur en parallèle en utilisant l'objet ExecutorService.
- Sharedoù l'on retrouve les classes qui seront partagées entre le serveur de Calcul et le Répartiteur.

A. serverRepartiteur

serverRepartiteur est l'entrée du système pour l'utilisateur. serverRepartiteur expose trois opérations à l'utilisateur :

- la liste des serveurs enregistrés aux différents registres RMI connus
- secureCompute une fonction qui ve permettre de réaliser le calcul séurisé de tâches passées en paramètre.
- nonSecureCompute une fonction qui ve permettre de réaliser le calcul en mode non séurisé de tâches passées en paramètre.
- Sont contenus dans cette classe essentielement le code pour faire fonctioner le serveur répartiteur et la logique des commandes exposées à l'utilisateur.

La gestion des serveurs de calculs disponibles et des stubs propre à chacun d'eux est gérée par la classe ServerCalculStubsManager. Pour les fonctions de calcul des opérations contenues dans les fichiers d'opérations, nous allons utiliser la classe Travail. Celle-ci aura extrait les Operations du fichier passé en paramètre et les aura distribués en Taches

La méthode nonSecureCompute est plus particuliaire car elle mettra à jour la tache de l'object Travail que lorsqu'elle sera sur que le résultat a été voté par une majorité d'au moins deux ServerCalcul.

1. Gestion traitement asynchrone des tâches auprès des serveurs de calcul

Pour trouver comment réaliser les opérations de mise à jour en simultané, le tutoriel http://tutorials.jenkov.com/java-util-concurrent/executorservice.html sur l'objet ExecutorService a été d'une grande aide pour structurer les bases de la classe serverRepartiteur.

2. Gestion des déconnexions intempestives, des tâches échouées

Nous avons trois types d'erreurs qui peuvent survenir qui ont leur état propre dans chaque tâche :

- hasStateRefused : le serveur de calcul a refusé la tâche donné. Auquel cas on essaye de lui retourner la même tache. L'idée est que s'il refuse il se peut qu'il accepte la prochaine fois. Après X essais (5 pour l'instant) on va diviser la tâche en deux et ainsi délester cette tâche d'une de ses moitiés et retourner de

nouveau cette tâche au même serveur mais avec une tâche deux fois plus petite. La seconde moitié est envoyée à un serveur au hasard. On pense à marquer à chaque fois l'état, le lien entre les tâches, de la tâche (futureTask)

- hasStateNotDelivered : la tâche n'a pas été livrée au serveur. Il a été déconnecté entre temps et n'est plus joignable.
- Autre : Autre erreur qui fait que le résultat de l'opération est null. Cela arrive surtout quand un serveur venait d'accepter une tâche et qu'il était en train de la compléter mais qu'il a été interrompu avant la fin.

Ces erreurs sont les mêmes en mode secure et nonSeccure.

A cela va se rajouter la vérification à chaque tour du nombre de serveur connecté et si ils sont en nombre suffisant (checkHasServersAndMaybeWaitForMore). Le programme peut être interrompu à cause du fait qu'il n'y ait plus de serveurs disponibles. Auquel cas, concernant cette tâche, nous serons contraint de passer la main à un autre serveur. S'il n'y a pas d'autres serveurs disponibles pour traiter cette tâche, nous afficherons alors un message demandant de reconnecter un serveur ou d'en connecter un nouveau.

- Si jamais on ne retrouve plus le premier Serveur, on peut passer à un second en le piochant dans la liste des serveurs.

Le mode sécurisé est plus simple car on peut directement récupérer le résultat retourné par le serveur de calcul et l'intégré dans les documents

Pour le mode non sécurisé, un grand nombre d'options en plus ont été ajoutées.

- On a mis en place un système de vote avec au départ seulement 2 votants que l'on élargie au besoin si l'on n'arrive pas à se mettre d'accord avec d'avoir au moins 50% exclu de serveur honnêtes.
- Le but est que si une tache est rejetée par un serveur de calcul on reste dans tous les cas le plus longtemps avec lui pour obtenir le résultat à moins d'être déconnecté. On va donc retourner au même serveur les tâches qui lui sont attribué aléatoirement au tout début mais une fois défini, on se tient à avoir les deux premiers serveurs choisis qui traiteront toute la Tache chacun de leur côté et mettront en commun par la suite pour élire le résultat. (première partie de la grosse condition, c'est-à-dire quand le résultat retourné est toujours null.
- Si jamais nous étions tombé sur un couple de serveurs honnête / malicieux, on va faire appel à un nouveau serveur pour essayer de départager chacun. Si ce nouveau ne suffit pas car il est lui aussi malicieux, le système va alors passer dans un mode où il demande à l'utilisateur de connecter un nouveau serveur de calcul honnête. (deuxième partie du else de la fonction nonSecureCompute lorsque le résultat retourné par l'Object Callable est non null.)

Ces fonctions du serverRepartiteur sont appuyées par toutes celles des package serverCalcul et shared.

B. serverCalcul

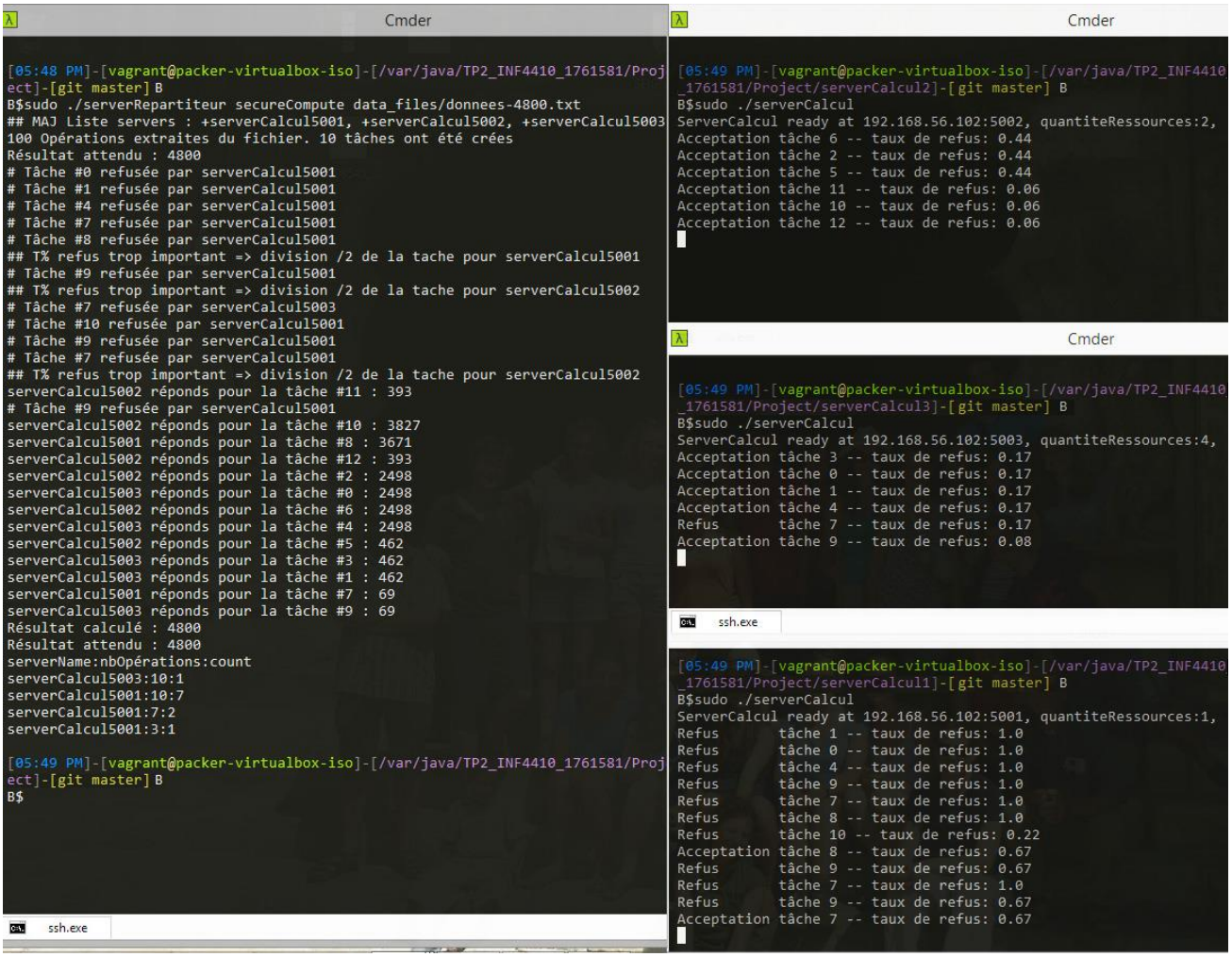
Concernant le serveur de calcul, il n'y avait que très peu de libertés à prendre. En effet, tout était bien défini dans le sujet et la FAQ du TP.

C. Cas d'utilisation

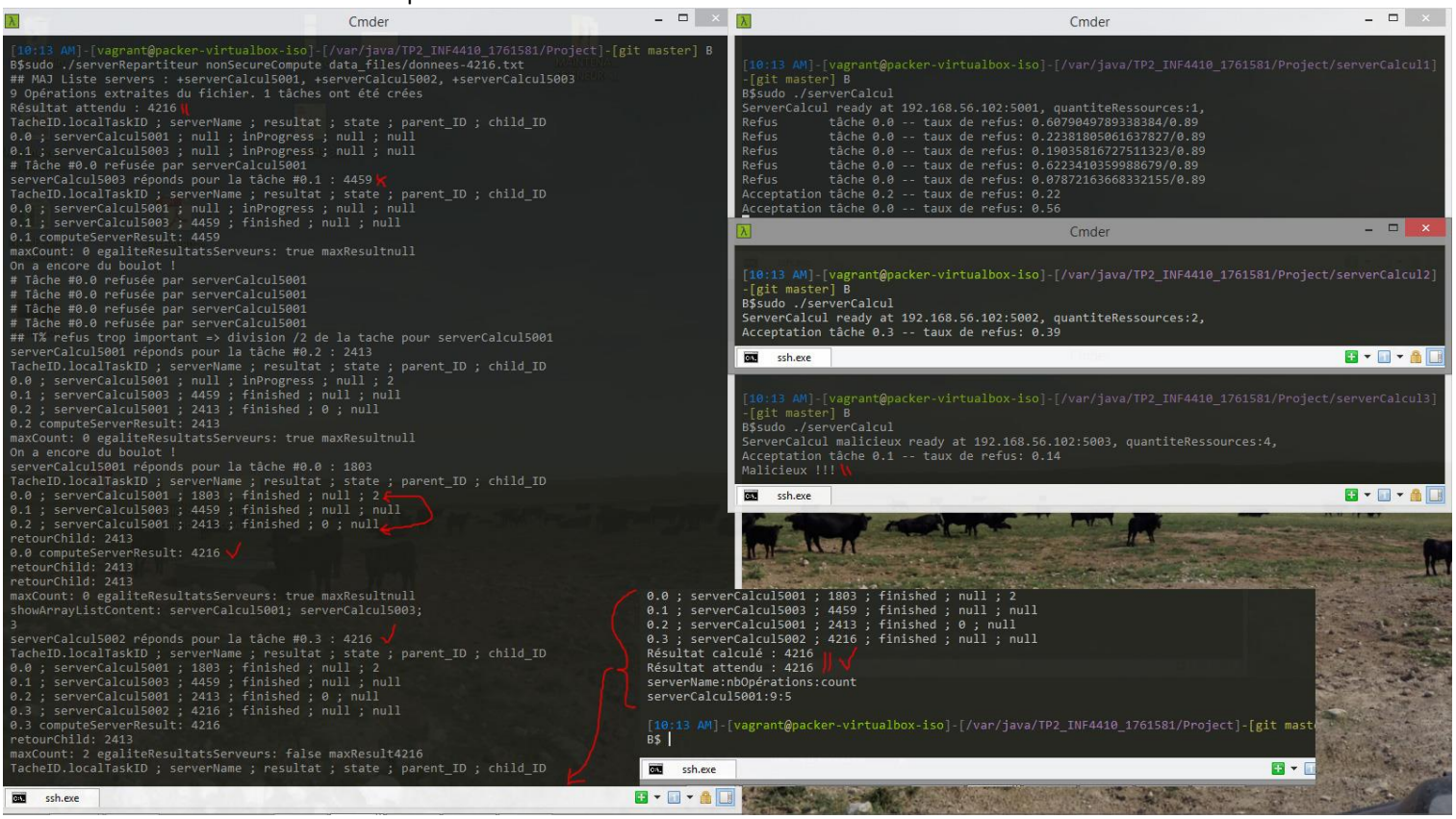
Voici quelques cas d'utilisation pour mieux comprendre le fonctionnement du programme.

1. secureCompute

Nous retrouvons le fonctionnement à trois serveurs de calculs lancés sur une machine différentes de celle sur laquelle nous avons le serverRepartiteur de lancé. Nous retrouvons aussi la division des tâches en deux si l'on a atteint ici 5 refus de suite pour une taille de tâche donné pour un serveur donné.



2. nonSecureCompute



Ici, nous avions utilisé un petit fichier de donné pour voir tout sur le même écran.

Nous pouvons constater le fonctionnement pour le mode non sécurisé de calcul :

- Nous avons eu dans un premier temps pour la tâche #0 les serveurs #5001 et #5003.
- Le serveur #5003 est malicieux, il a retourné 4459 pour la tache #0.1 au lieu de 4216.
- Le serveur #5001 ayant un qi de 1 est obligé de couper en deux la tâche #0.0 en #0.0 et #0.2
    - On obtient alors 1803 pour #0.0 et 2413 pour #0.1 qui sommés donnent bien 4216.
- On n'a pas le même résultat. On est obligé d'interroger un nouveau serveur différent des deux premiers pour départager le vote. C'est le serveur #5002 qui est choisi.
    - Celui-ci créé la tâche #0.3 qui a pour parent null, il est lui aussi lié à la racine de l'arbre.
    - Éventuellement, la tâche #0.3 est retournée et vaut 4216.
- Le résultat calculé est bien 4216 qui correspond au vote des serveurs #5001, #5002 et #5003.