Mise en situation

Suite au premier travail pratique, lequel visait à faire l'implémentation d'un communication client-serveur en utilisant Java RMI, le travail pratique 2 nous a mené à implanter un système distribué ayant pour fonctions le calcul de plusieurs opérations mathématiques de types *Pell* (nombre de Pell) et *Primary* (plus grand facteur premier).

Nous avons donc eu à construire un système distribué constitué d'un répartiteur qui a eu la responsabilité de distribuer le flot d'opérations aux serveurs disponibles, d'un service de répertoire des noms qui donne la liste des serveurs disponibles au répartiteur et authentifie ce même répartiteur pour les serveurs, et enfin de serveurs (2 et plus) qui s'occupent de traiter les opérations et de retourner le résultat.

Pour bien assimiler les capacités des systèmes répartis et les contraintes qui s'y rattachent, nous avons démarré nos serveurs en limitant leurs ressources de calcul, nous avons simulés les pannes intempestives, ainsi que la présence de serveurs dit malicieux, qui ont pour but de retourner de faux résultats.

Finalement, la robustesse du système, par rapport aux résultats retournés par les serveurs, étant un aspect impossible à négliger pour implanter tout système réparti, nous avons simulés deux modes de fonctionnement. Le premier étant le mode « sécurisé », un mode par lequel les résultats sont acceptés par le répartiteur, et ce, sans vérification. Le deuxième est le mode « non sécurisé », un mode qui nécessite que le résultat fourni le soit au moins de la part de deux serveurs, pour ainsi le considérer comme résultat correct.

Structure du projet

La présente section représente l'architecture du système implanté dans le but d'expliquer les fonctionnalités des classes et méthodes impliquées ainsi que les interactions qu'elles entretiennent.

Script de lancement

Afin de libérer, d'initialiser et de lancer les composants du système réparti d'une manière automatisée, un script bash s'avère très utile. Dans note cas, nous avons écrit un script ./

bootstrap.sh qui lit la configuration dans *config.txt*, puis démarre automatiquement le système décrit dans ce dernier fichier.
Ce dernier fonctionne de la manière suivante:

1. Récupérer les informations de configurations, telles que les addresses IP des serveurs, du répertoire de noms, des ports, du fichier d'opérations, etc…
2. Compiler le serveur, le répartiteur et le répertoire de noms, ainsi que le module shared
3. Arrêter les processus *rmiregistry* et *java*
4. Rafraîchir la liste des serveurs
5. Exécuter *rmiregistry* selon la configuration (ports)
6. Démarrer le répertoire de noms, les serveurs puis le répartiteur

---

Répartiteur

Le répartiteur (*LoadBalancer*) est notre point d'entrée. Il prend 4 arguments, lesquels sont le nom d'utilisateur, le mot de passe, le chemin vers le fichier contentant les opérations et le port.
Ce dernier fonctionne comme suit:

1. Découper le travail sur les serveurs disponibles
2. Communiquer aux serveurs les informations nécessaires d'une manière parallèle (threads)
3. Rassemble les résultats des serveurs tout en gérant les pannes pouvant survenir
4. Retourner le résultat ainsi que certaines statistiques (affichage sur la console)

Il est démarré avec la commande:
> bash loadBalancer <username> <password> <filePath> <port>

Répertoire des noms

Le service de répertoire des noms (*NameRepository*) agit comme un intermédiaire ou une station obligatoire à laquelle le répartiteur et les serveurs devraient faire une escale pour assurer le bon fonctionnement du tout. Comme cité plus haut, il devra maintenir une liste des serveurs disponibles et devra offrir une fonctionnalité d'authentification du répartiteur aux serveurs.
Il est démarré avec la commande:
> bash nameRepository <port>

Serveurs de calculs

Les serveurs (Server) implémentent les fonctions de traitement des opération *Pell* et *Primary* contenus dans les fichiers reçus, sans oublier le calcul du taux de refus à cause des ressources
manquantes. Ils démarrent avec 3 arguments lesquels sont : le q, le m et le port sur lequel ils écoutent (le même utilisé pour démarrer le *rmiregistry*).
On crée un serveur de la manière suivante:
> bash server <malice> <capacité> <port>

Flot d'exécution

1. On démarre un le service de répertoire des noms.
2. On démarre ensuite les serveurs qui devront s'enregistrer au service (fourniront leurs adresses IP et le port sur lequel ils écoutent).
3. On démarre ensuite le répartiteur qui ira récupérer la liste des serveurs disponibles du service de répertoires des noms.
4. Le répartiteur communique par après avec les serveurs son nom et son mot de passe (aucune emphase sur la sécurité n'a été considérée).
5. Le serveur récupère le nom et le mot de passe et fait appel à la fonction d'authentification du *NameRpository*. Si le résultat est positif, ils seront disposés à traiter les blocks d'opérations qui leur sont envoyés, en autant que ceux-ci respectent les contraintes qui leur aient propres (capacité q).

Gestion de pannes intempestives

La gestion des pannes est une propriété très importante des systèmes répartis. Par conséquent, le système que nous avons produit implémente cette fonctionnalité.
La façon dont ce mécanisme fonctionne peut se formuler comme suit:

1. Envoyer les requêtes de calculs aux serveurs d'une manière parallèle (thread)
2. Détecter si un des threads lance une exception ou tout simplement est détruit
3. Utilisation des objets *Runnable*, *Future* et *ExecutorService*
4. En cas de panne d'un serveur, le répartiteur marque ce dernier
5. Donner la tache bloquée au prochain serveur disponible

Ainsi nous pouvons garantir un bon niveau de disponibilité, vu que le système s'auto-régule si une panne se produis.
Une exécution contenant une panne simulée par un serveur qui s'auto-détruit est illustrée par la figure 5 dans l'annexe.

Tests de performance

Mode sécurisé

On remarque clairement que lorsqu'on augmente le nombre de serveurs, le temps d'exécution diminue. Cependant plus on ajoute des serveurs, et plus cette diminution se ralenti.
Le fait que le temps d'exécution diminue s'explique tout simplement par l'augmentation de la capacité de traitement du système lorsqu'on lui ajoute des serveurs de calculs.
En effet, le système utilise l'avantage de l'exécution parallèle des calculs répartis sur les serveurs, et cela diminue le temps.
D'un autre coté, le ralentissement de cette diminution s'explique par le coût qu'on rajoute au système lorsqu'on augmente le nombre de serveurs. En effet, dans cette situation la latence du réseau devient un facteur considérable. De plus le fait de changer de contexte entre les threads, ajoute un autre cout au répartiteur qui est l'unique gestionnaire du système.
Les résultats expliqués sont illustrés par le graphe dans la figure 1, ainsi que les captures d'écrans dans les figures 2, 3 et 4 de l'annexe.

Question 1

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?

Afin d'améliorer la résilience du système, il faudra une gestion rigoureuse des pannes du répartiteur pour garantir le flot des opérations si le travail s'interrompt à cause d'un répartiteur hors-service.

Une solution possible serait de définir un protocole qui permette de transmettre les taches du répartiteur à une successeur capable de reprendre les rennes. Il faudrait d'abord conserver l'état du répartiteur après chaque opération envoyée, afin de permettre au successeur de continuer là où le dernier a arrêté.

Cela serait avantageux dans la mesure où le protocole serait tout à fait autonome de gérer la panne tout en assurant la continuité du travail. Néanmoins, la résilience vient à un prix, celui d'une allocation de ressources importantes (mémoire et temps), car comme cité plus haut, il faudra sauvegarder l'état du répartiteur à chaque opération.

Puisqu'il est rare, voire impossible de concevoir un système parfait. Celui-ci n'en fait pas exception. En effet, une panne matérielle pourrait corrompre la sauvegarde de l'état du répartiteur. Le successeur héritera donc d'un état faux ou illisible.
De plus, nous assumons qu'il peut y avoir des serveurs malicieux. Si l'un d'entre eux roule le répartiteur, il faudra s'assurer que les erreurs de calculs sont gérées adéquatement.

Annexe

Images ici