# INF4410 – Systèmes répartis et infonuagique

# TP2 – Services distribués et gestion des pannes

**Chargé de laboratoire:**

Houssem Daoud

17 Octobre 2016

École Polytechnique de Montréal

# Introduction

Le second travail pratique vous fera prendre conscience de plusieurs problèmes reliés à la gestion des pannes à l'intérieur d'un système réparti sur plusieurs machines. Les services distribués permettent de répartir le risque dû à un bris sur l'ensemble des machines plutôt que sur un seul point de rupture. Dans un système distribué, des pannes peuvent survenir à tout moment. Il faut donc que le service soit assuré même quand une anomalie survient. De plus, il faut parfois prévoir des mécanismes qui vérifient les résultats calculés par des machines non fiables. Par exemple, le projet Folding@home, qui permet à des ordinateurs personnels de contribuer à une grappe de calcul, doit s'assurer que les résultats ne sont pas entachés par des utilisateurs malicieux ou du matériel défectueux.

**Prenez le temps de lire l'énoncé une fois attentivement avant de commencer, il y a beaucoup de petits détails à ne pas négliger.**

# Remise

- Méthode: Par Moodle, un seul membre de l'équipe doit remettre le travail, mais assurez vous d'inclure les deux matricules dans le rapport et le nom du fichier remis.
- Échéance: vendredi le 11 novembre avant 16h00, voir le plan de cours pour les pénalités de retard.
- Format: Une archive au format .tar.gz nommée *[matricule 1]-[matricule 2]-TP2-INF4410.tar.gz*. L'archive doit contenir votre code et un rapport avec les réponses aux questions. Ne remettez que les fichiers sources, pas les fichiers compilés. Pour les projets Eclipse, ne remettez que le projet, pas le workspace en entier.

Le laboratoire doit être fait en binôme.

On vous demande d'inclure un *README* expliquant en détails comment exécuter chacun des tests de performance demandés dans ce TP. Après la remise, il se peut que le chargé de laboratoire vous demande de faire une démonstration de votre travail, donc gardez votre code.

# Barème

**Respect des exigences et bon fonctionnement:** 10 points
**Rapport et réponses aux questions:** 6 points
**Clarté du code et commentaires:** 4 points
**Total:** 20 points, valant pour 10% de la note finale du cours.

Jusqu'à 2 points peuvent être enlevés pour la qualité du français.

# Spécifications

## *Généralités*

On vous demande de développer un système distribué qui a comme tâche de calculer le résultat d'une série d'opérations mathématiques. Un répartiteur devra s'occuper de distribuer le travail à faire entre les différents serveurs de calcul ainsi que de récupérer et colliger les résultats. Le répartiteur est également celui qui s'occupe de gérer la défaillance d'un des serveurs.

## *Spécification des opérations*

Le système réparti reçoit une liste d'opérations à effectuer sous forme de fichier texte. Il doit calculer la somme des résultats des opérations. Afin d'éviter un débordement d'entier, un modulo 4000 est appliqué à chaque étape du calcul.

Chaque ligne du fichier texte d'entrée respecte le format:

```
opération opérande
```

Où `opération` est le nom de l'opération et `opérande` est la valeur numérique de l'opérande.

Les opérations possible sont `pell` ($n^e$ nombre de Pell) et `prime` (plus grand facteur premier). Pour calculer le résultat d'une opération, vous devez obligatoirement utiliser les méthodes fournies dans la classe `Operation`.

Exemple de fichier d'entrée:

| | |
|---|---|
| `pell 5` | `pell`(5) = 29 |
| `prime 10` | prime(10) = 5 |
| `pell 13` | `pell`(13) = 33461 |
| `prime 6500` | `prime`(6500) = 13 |

Résultat attendu:

( ( ( 29 % 4000 + 5 ) % 4000 + 33461 ) % 4000 + 13 ) % 4000 = 1508

Notons que ce résultat aurait aussi pu être obtenu ainsi:

( ( ( 29 % 5000 + 5 ) % 4000 ) + ( ( 33461 % 4000 + 13 ) % 4000 ) ) % 4000 = 1508

## *Répartiteur*

Le répartiteur est le point d'entrée du système. C'est lui qui se connecte aux serveurs de calcul et soumet le travail que chacun d'eux doit effectuer. Il prend en paramètre le chemin du fichier d'entrée qui contient les opérations à effectuer. À la fin du calcul, il affiche le résultat sur la sortie standard.

## *Modes de fonctionnement*

Le système a la possibilité de travailler selon deux modes de fonctionnement.

- En mode sécurisé, le résultat des serveurs de calcul est considéré bon et valide. Le répartiteur n'a donc besoin que d'une seule réponse de la part d'un serveur.
- En mode non-sécurisé, le système ne fait pas confiance aux serveurs de calcul. Le répartiteur considère alors une réponse comme étant valide que si deux serveurs de calcul sont d'accord sur la même réponse à un même travail.

## *Serveurs de calcul*

Les serveurs de calcul sont responsables de recevoir des tâches du répartiteur, d'effectuer le calcul et de retourner la réponse au répartiteur. Une tâche est une partie des opérations à effectuer. Lors de la

réception d'une tâche, le serveur tente d'allouer les ressources nécessaires. Si les ressources sont insuffisantes, le serveur doit indiquer au répartiteur qu'il n'est pas en mesure d'effectuer le calcul.

### *Simulation des ressources*

Afin d'offrir un service équitable à tous leurs utilisateurs, les calculateurs limitent le temps de calcul qui peut être alloué à chaque tâche. Ainsi, chaque calculateur *i* acceptera toutes les tâches contenant au plus $q_i$ opérations mathématiques (paramètre configurable par serveur). Les tâches contenant davantage d'opérations pourront être acceptées seulement si le calculateur n'est pas surchargé. Pour les fins du travail pratique, on simulera le taux de refus des tâches à l'aide d'une fonction mathématique simple. Le taux de refus croit linéairement de $q_i$ (0%) à 5*$q_i$ (100%). L'équation suivante permet de calculer ce taux de refus (*T*) en fonction du nombre d'opérations pour lequel l'acceptation de la tâche est garantie ($q_i$) et le nombre d'opérations de la tâche soumise ($u_i$) :

$$T = \frac{u_i - q_i}{4 q_i} * 100\,\%$$

Le répartiteur ne connait pas les capacités de chaque serveur de calcul. Il doit donc s'assurer de distribuer des tâches de la bonne taille afin d'éviter de recevoir des échecs trop souvent. Notez également que les serveurs de calcul peuvent avoir une valeur $q_i$ qui varie d'un serveur à l'autre.

## Pannes intempestives

En plus des tâches qui ne se réalisent pas à cause d'un manque de ressources, le système doit pouvoir réagir correctement si jamais un serveur de calcul est tué au beau milieu de l'exécution d'une tâche. Le répartiteur doit alors redistribuer le travail vers les autres serveurs toujours disponibles.

## Serveur de calcul malicieux

En mode non-sécurisé, les serveurs de calcul peuvent retourner des résultats erronés. Parmi tous les serveurs de calcul, certains devront pouvoir être configurés de manière à agir malicieusement une fraction du temps. En autant qu'un nombre suffisant de serveurs agissent toujours de bonne foi (>50%), le répartiteur devrait toujours être en mesure de récupérer le bon résultat. La fréquence à laquelle un serveur donne un faux résultat est configurable lors de son lancement : le paramètre **m** allant de 0% (serveur de bonne foi) à 100% (serveur toujours malicieux).

## Travail demandé

Pour ce travail pratique, vous devez utiliser la technologie de communication Java RMI. Votre application doit pouvoir s'exécuter sur les ordinateurs du `L4714`.

### *Programme répartiteur*

Vous devez implémenter le programme qui fait la répartition du travail entre les différents calculateurs et produit le résultat final. Vous êtes libres de choisir comment découper le travail.

### *Programme serveur de calcul*

Vous devez implémenter l'algorithme qui permet d'effectuer les opérations demandées contenues dans une tâche soumise par le répartiteur. Assurez-vous que chaque serveur puisse être configuré indépendamment quant à la taille des tâches pour lesquelles l'acceptation est garantie ($q_i$) et son taux de réponse erronée (en mode non sécurisé).

### *Tests de performance – mode sécurisé*

On vous demande de créer une instance du système en mode sécurisé. Choisissez un fichier d'entrée suffisamment gros pour que le calcul prenne au moins quelques secondes à effectuer.

On va exécuter le même calcul en variant le nombre de serveurs :

- 2 serveurs ($q_1 = 3$; $q_2 = 6$)

- 3 serveurs ($q_1 = 3$; $q_2 = 6$; $q_3 = 9$)

- 4 serveurs ($q_1 = 3$; $q_2 = 6$; $q_3 = 9$ ; $q_4 = 12$)

Présentez le graphique du temps d'exécution en fonction du nombre de serveurs. Expliquez les résultats en faisant des liens avec vos choix d'implémentation.

### *Tests de performance – mode non-sécurisé*

Instanciez le système mais cette fois en mode non-sécurisé. Dans cette expérience, on va fixer le paramètre **q=5** pour trois serveurs de calculs et varier le paramètre ***m***.

Exécutez les trois cas suivants tout en mesurant le temps écoulé :

- Trois serveurs serveurs de bonne foi.
- Un serveur malicieux 50% du temps, deux autres serveurs de bonne foi.
- Deux serveurs malicieux 50% du temps, un autre serveur de bonne foi.

Présentez et expliquez les temps obtenus.

**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?

## Rapport

Puisque vous avez beaucoup de liberté pour ce travail pratique, chaque équipe devrait obtenir un résultat assez différent. Votre court rapport devrait donc mentionner et expliquer les différents choix que vous avez faits pendant la conception: la façon dont le répartiteur divise les tâches et gère les tâches échouées, comment le système détecte une panne intempestive, etc. Pourquoi avez-vous fait ces choix de préférence à d'autres ?

# Annexe – Conseil pour le travail au laboratoire

Entre les ordinateurs du laboratoire, seuls les ports de 5000 à 5050 sont ouverts pour la communication entre postes. Vous devez donc vous assurer que les processus qui écoutent sur le réseau écoutent sur un port dans cet intervalle.

Vous devrez d'abord modifier le port d'écoute de *rmiregistry* en le lui fournissant comme paramètre en ligne de commande, par exemple:

```
/opt/java/jdk8.x86_64/bin/rmiregistry 5001
```

L'appel à getRegistry devra refléter ce changement de port. Puis, en exportant un objet avec exportObject, un port entre 5000 et 5050 devra être fourni afin que le client puisse se connecter.

Pour faire vos tests avec plusieurs instances, utilisez SSH entre les postes. Lorsque vous êtes sur un poste du labo, vous n'avez qu'à taper `ssh L4714-xx` pour vous connecter à un autre poste (par exemple, `L4714-01`). De plus, vos sessions sur les différents postes partagent le même système de fichiers. Vous n'avez donc pas à transférer vos fichiers entre les postes, ils seront là dès que vous vous connecterez par SSH.

Vous aurez plusieurs variables de configuration à modifier afin de connecter les différents serveurs entre eux : adresses des serveurs, numéro de port, mode de fonctionnement, quantité de ressources, etc. Vous pouvez utiliser un ou plusieurs fichiers de configuration pour contenir ces variables afin de facilement les modifier pour réaliser vos différents tests.