INF8480 – Systèmes répartis et infonuagique

**Travail Pratique 1**

**Appels de méthodes à distance**

Rachel **Côté Bessette** 1721384

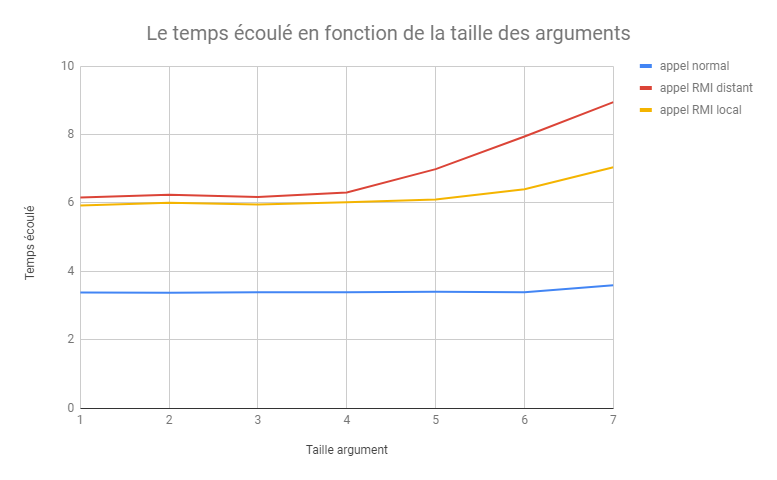
Clarence **Desmarais** 1762064

Remis à Houssem Daoud

Polytechnique Montréal

2 octobre 2018

**Question 1: On se propose d'étudier l'impact de la taille des arguments sur la performance des appels RMI ?**

**Tableau 1 : Le temps écoulé en fonction de la taille des arguments **

**Analyse des résultats obtenus :**

En analysant les graphiques, nous remarquons que pour l’appel normal l’augmentation du temps d’exécution varie assez peu (légère hausse) lorsque la taille des arguments grandit. Ensuite, concernant les appels RMI Local, on remarque que le temps d’exécution augmente un peu plus lorsque la taille des arguments grandit. C’est dans le cas de l’appel RMI distant que l’on remarque une plus grande hausse dans le temps d’exécution lorsque la taille des arguments augmente. Notons également que les hausses du temps d’exécution sont beaucoup plus fortes lorsque la taille des arguments est élevée, particulièrement pour les appels RMI.

D’abord, la performance nettement moins bonne des appels RMI distants peut être expliquée par l’utilisation du réseau pour l'accès aux objets distants. De plus, il y a aussi le temps pris pour les sérialisations et désérialisations. Ensuite, pour les appels RMI locaux, leur performance se situe entre les appels normaux et appels RMI distant. En effet, puisque les appels sont fait sur localhost, on évite les délais du réseau.

**Discutez les avantages et les inconvénients de Java RMI :**

**Avantages :**

Une des fonctionnalités et bénéfices importants lié à la RMI (Remote Method Invocation) est que cela permet d’appeler une méthode sur une machine distante afin de performer une opération et ensuite de retourner le résultat à la machine. Aussi, il est possible d’avoir accès à la définition d’une classe d’objet (remote) sur une machine distante.

Une telle fonctionnalité permet de découpler certains systèmes et permet plus de flexibilité lors de la conception des applications. Ainsi, lorsque les applications sont découplées, ils sont souvent plus facile à maintenir.

**Désavantages :**

Avec cette méthode, il est impossible d’utiliser du code qui n’est pas Java. En effet, la communication doit se faire entre des machines java exclusivement.

De plus, l’invocation d’une méthode sur une machine distante peut être sujette à des problèmes de latences notamment dû au réseau.

Également, il peut y avoir plus facilement des problèmes de sécurité puisqu’il y a communication entre deux machines différentes.

**Question 2 : Expliquez l'interaction entre les différents acteurs (client, serveur et registre RMI) à partir du tout début de l'exécution. Ainsi, à partir du moment où on lance le serveur jusqu'à l'appel de la fonction à distance par le client, décrivez toutes les communications qui ont lieu entre ces acteurs. Faites le lien entre vos explications et le code de l'exemple fourni. Il faut citer les classes et les méthodes responsables de chaque interaction.**

Avant de démarrer le serveur ou le client, il faut s’assurer de démarrer le registre RMI.  Le registre RMI peut être vu comme un entrepôt d’objets distants.

Le serveur, lorsqu’exécuté, crée un objet contenant la nouvelle méthode que nous avons créé (*testArrayLengthImpact*) et l’entrepose dans le registre RMI à l’aide d’une chaîne de caractères servant de clé (ex: “server”).

Pour l’appel local, le client crée un objet *localServer*, qui lui vise à évaluer l’appel de fonction en local (appel de fonction normal). L’exécution de l’appel de la fonction se déroule normalement, comme toute fonction appelée dans un même programme.

Pour les appels RMI, le client crée deux autres objets : *localServerStub* et *distantServerStub*. Pour créer ces objets, le client doit d’abord avoir accès au registre RMI. Ainsi, en appelant *LocateRegistry.getRegistry(hostname),* il est possible pour le client de récupérer un stub vers le registre RMI. Ensuite, une fois le stub obtenu, on récupère un stub vers l’objet distant à l’aide du stub vers le registre RMI grâce à la clé ‘’server’’.

Dans le cas du serveur local, on récupère le registre RMI sur l’adresse locale (127.0.0.1), tant dit que dans le cas du serveur distant, on récupère le registre RMI sur l’adresse distante.

L’objet distant (stub) associé à la clé ‘’server’’ permet d’accéder à la nouvelle méthode que nous avons créée. Lorsque l’on appelle cette méthode (ex : *localServerStub.testArrayLengthImpact(bytes)* ou *distantServerStub.testArrayLengthImpact(bytes)*), plusieurs actions ont lieu. D’abord, le client ouvre une connexion vers le serveur via le stub de l’objet distant et sérialise les informations de l’appel pour l’envoi. Ensuite, le serveur intercepte l’appel, attribue l’appel à l’objet distant et sérialise le résultat pour le renvoyer au client. Finalement, le client reçoit le résultat, le désérialise et retourne le résultat.

La différence entre les deux appels RMI est que sur l’un, l’objet partagé se trouve sur la même machine (en local), tant dit que dans l’autre, l’objet se trouve sur une machine distante.

**https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html**