INF4410 : Systèmes répartis et infonuagique

TP1 : Java RMI

Rapport

Présenté à :

M. Houssem Daoud

Travail réalisé par :

Félix Pelletier – 1581243

Jean-Frédéric Faust – 1583921

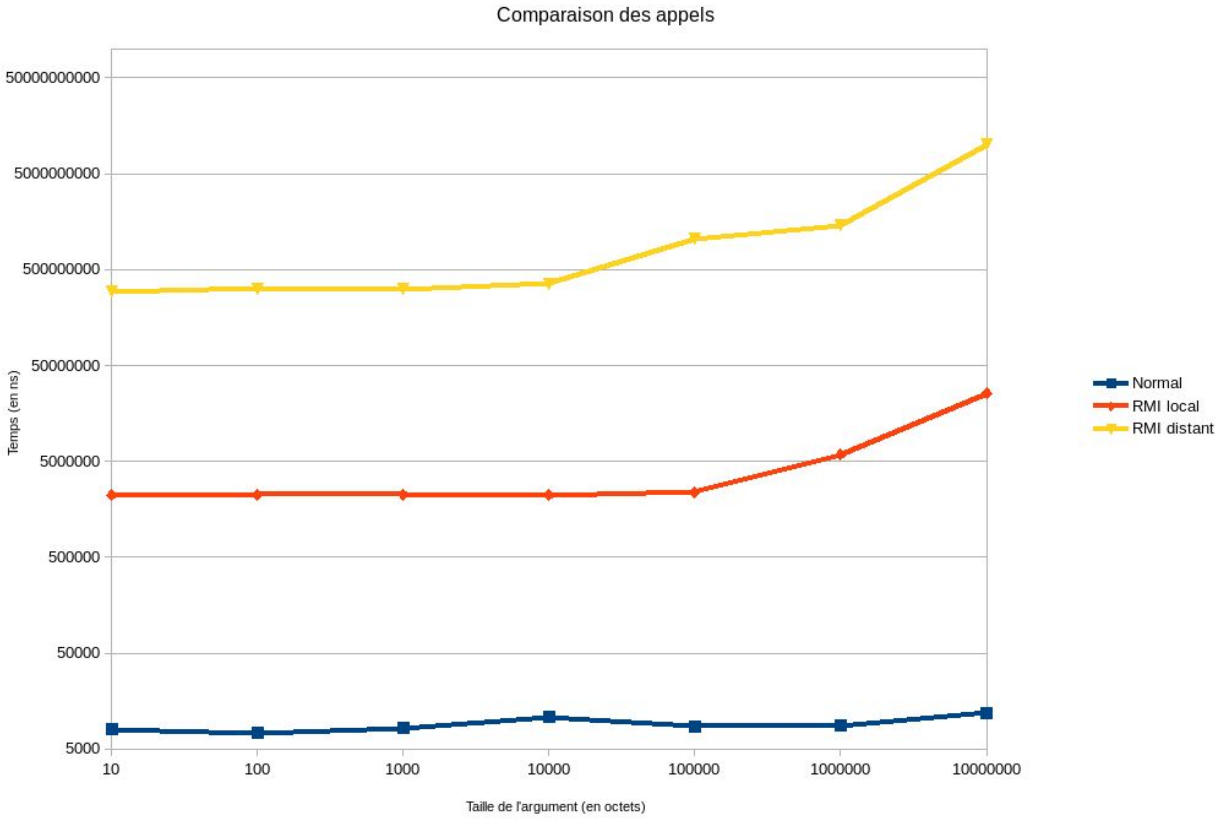
École Polytechnique de Montréal

22 février 2015

Partie 1

Question 1:

Tableau : Comparaison du temps requis pour effectuer les différants appels de fonction



Tout d'abord, nous tenons à spécifier que nous avons recommencer l'expérience plusieurs fois afin d'obtenir des résultats qui nous semblent normaux.

Pour les appels de fonctions normaux, on ne peut pas dire qu'il y a une croissance logarithmique. Il y a tout de meme des petites variations entre les temps obtenus mais cela est négligable et peut être causé par des éléments impondérables du système. Cependant, on peut observer une croissance logarithmique pour les appels RMI locaux et distants. On peut supposer que cela est causé par les traitements requis par RMI.

On remarque que les courbes RMI (rouge et jaune) sont pratiquement parallèles. Le seul endroit où on retrouve une différence par rapport aux autres points est pour une taille d'argument de 10^5.

On retrouve une certaine différence de temps presque constante entre chacune des courbes de temps. La différence entre les appels normaux et les appels RMI locaux (bleu et rouge) est principalement causé par l'addition de RMI dans le processus d'échange d'informations. Pour ce qui est de la différence entre les appels RMI locaux et distants (rouge et jaune), on suppose que cela est causé par le temps de propagation des informations sur le réseau pour les appels RMI distants. Finalement, la différence entre les appels normaux et les appels RMI distants est simplement l'addition des deux délais mentionnés précédemment.

Par rapport au cas d'utilisation de Java RMI, une bonne utilisation serait dans un contexte où il est peu probable qu'on doit développer d'autres applications nécéssitant un accès au service. De plus, si on considère rester dans un environnement de développement Java, c'est idéal de travailler avec Java RMI. Cependant, il serait peu approprié d'utiliser cette technologie si on doit maintenir une interopérabilité entre plusieurs services qui ne sont pas nécessairement en Java ou si l'on prévoit élargir nos horizons technologiques à l'extérieur de l'univers Java.

Question 2:

a) Appel normal (sans RMI)

Il n'y a pas d'intergiciel (*middleware)* qui se retrouve entre les deux composants (*client* et *localServer*). Dans la fonction *run* de l'objet *Client*, on fait appel à la fonction *appelNormal* du même objet. Au début de la fonction *appelNormal*, étant donné que l'objet *localServer* est de type *FakeServer*, il n'y a pas réellement de communication réseau puisqu'il s'agit d'un objet faisant partie du même domaine d'application. La fonction *execute(int a, int b)* est appelée sur l'objet *localServer* et retourne le résultat de l'addition entre les deux entiers fournis par paramètres. Le fil d'exécution revient à la fonction appelante (*appelNormal*) et le temps d'éxecution ainsi que le résultat sont affichés.

b) Appel RMI local

On démarre le registre RMI (*rmiregistry*), qui sera utilisé plus tard autant par les processus *client* que *server*. On lance le serveur (*server*) et celui-ci fait appel à la méthode *exportObject* de l'instance statique *UnicastRemoteObject* pour créer le composant qui permettra de communiquer avec notre objet distant qu'est le serveur. Ce composant sera considéré comme le squelette du serveur et du stub pour le client. Par la suite, on récupère l'instance du registre pour ensuite assigner le squelette en tant que valeur pour la clé "server".

Par la suite, lorsqu'on lance un client, ce dernier recherche dans le registre le stub associé à la clé mentionnée précédemment. Ensuite, la fonction *run* est appelée et celle-ci appelera à son tour *appelRMILocal*. Cette dernière va appeler la fonction *execute* du stub obtenu à partir du registre.

Le stub se chargera d'initialiser la connection avec la machine virtuelle Java qui contient le serveur local. Il s'occupe d'écrire et de transmettre les paramètres à la machine virtuelle et attend le résultat de l'invocation de la méthode. À ce stade, le rôle de squelette prend le relai, effectue la lecture des paramètres et invoque la méthode sur l'implémentation réelle de l'objet. Lorsque l'exécution de la méthode est complétée, il se charge d'écrire et de transmettre le résultat, que celui-ci soit une valeur ou une exception.

L'exécution reprend au niveau du stub et ce dernier lit a valeur retournée si aucune exception n'est retournée. Finalement, la fin est identique à celle de l'appel normal, soit l'affichage à la console sera mise à jour avec les dernières données.

c) Appel RMI distant

Pour ce dernier cas, il est pratiquement identique au cas précedent. Les seules différences résident au niveau de la localisation du registre et du serveur. Plus précisément, le registre est maintenant situé sur le serveur distant. Pour le reste, il y a des communications réseaux entre les processus *client* et *server,* incluant tout ce qui rapporte à RMI.