INF4410 – Travail pratique 1

Automne 2017

Appels de méthodes à distance

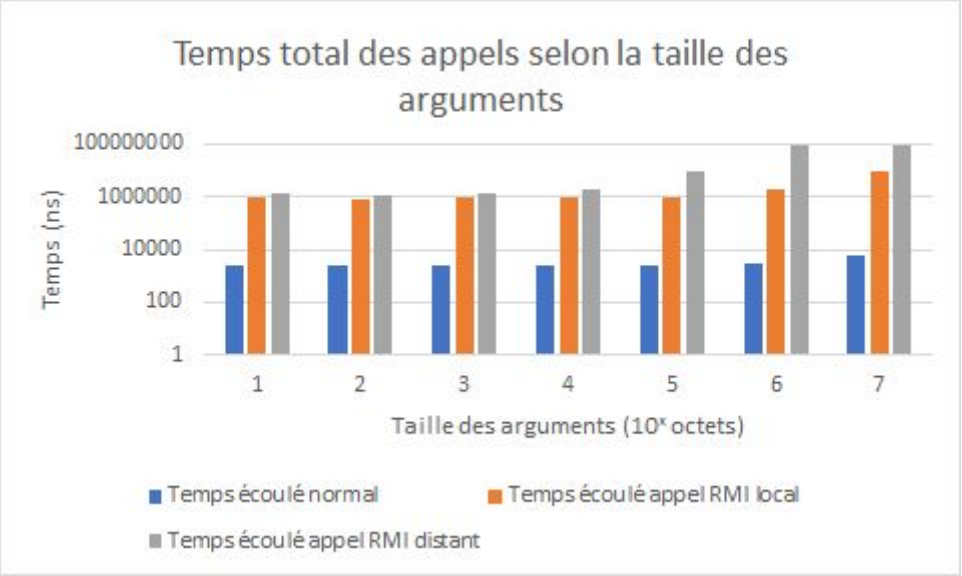
[1740216] - Ulric Villeneuve

[1739351] - Simon-Pierre Desjardins

3 octobre 2017

Partie 1

Question 1:



Nous remarquons que nos résultats semblent être logique. En effet, plus la taille des arguments augmentent, plus nous obtenons un temps élevé pour chacun des appels. On remarque que l'augmentation se fait graduellement pour un appel normal et donc avec une certain linéarité. Pour ce qui est des appels RMI, on peut remarquer qu'avec les appels distants plus particulièrement, l'augmentation ne semble pas linéaire et aurait plus tendance à devenir exponentiel si on continuait l'expérience. Cela est surement dû à l'étape du "marshalling" et du "demarshalling" qui prend plus de temps avec le serveur distant contrairement au cas avec l'exécution normal dans le cas où il y a de plus en plus d'arguments (ou taille des arguments augmentent).

Pour ce qui est de Java RMI, plusieurs avantages nous penchent à choisir ce protocole pour certaines situations. En effet, ce protocole s'occupe de la gestion des threads et des sockets. De plus, il est possible de changer l'implémentation du côté serveur sans nécessairement avoir besoin de toucher au côté client. Il est également possible de charger de manière dynamique des classes.

Par contre, il y a aussi des désavantages à utiliser cette méthode. La première est que nous pouvons qu'utiliser le langage java dans les deux sens de la communication ce qui peut limiter son utilité. Il peut également y avoir des problèmes de sécurité et il est important de bien s'assurer que le protocole est sécurisé pour éviter toutes failles. Finalement, il n'est pas possible de garantir que le client va utiliser le même "thread" dans des appels consécutifs, il est donc important de contourner ce potentiel problème en ayant un mécanisme qui permet d'identifier nous même le client.

Question 2:

Tout d'abord, JavaRMI se charge de générer les classes souche et squelette correspondant à la classe ServerInterface dans le cas de ce travail. Le serveur va ensuite enregistrer l'objet distant et communiquer une instance de la souche (ServerInterface) au registre RMI. Par la suite, le client réclame (classe Client) la souche associé à son instance. Le registre RMI retourne ensuite la souche. Le client charge ainsi la classe souche d'après java.rmi.server.codebase. De plus, le client va invoquer une méthode sur la souche, par exemple notre fonction vide. La souche va ensuite prendre les paramètre de la méthode (le tableau de la taille en bytes entrée, dans notre cas) et l'envoyer au serveur. Le serveur va ensuite lire les paramètres de la méthode et invoque la méthode sur l'objet (fonction vide du serveur). Il va ensuite retourner la valeur de retour à la souche qui va la lire et la transmettre au client directement.