Wissam Salamé 1488205

Laboratoire 1

Présenté à monsieur

Houssem Daoud

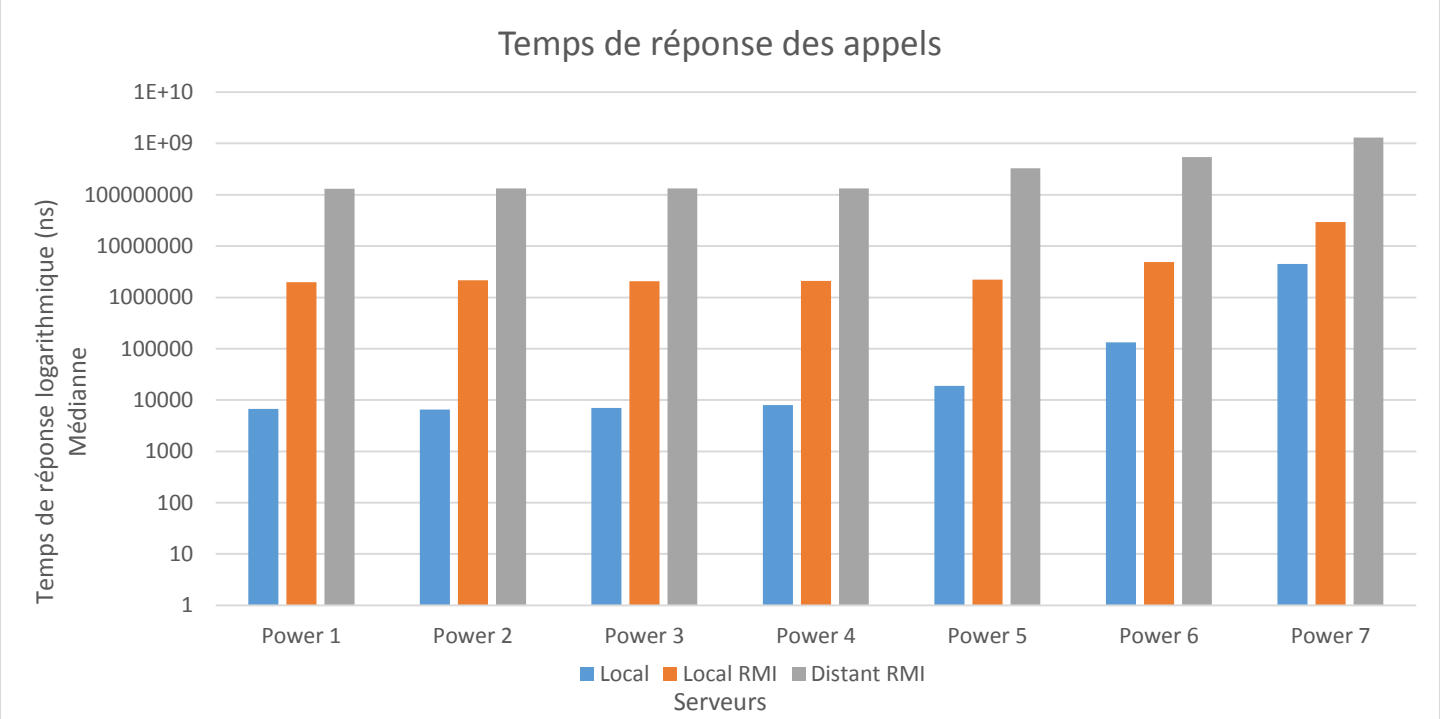
INF4410

École Polytechnique

19 octobre 2015

Partie 1

Question 1



Dans le graphique, on constate que pour une puissance inférieure ou égale à 4, les données reçus sont très semblable pour chacun des serveurs respectivement. En d'autres mots, on voit que pour un appel au *local* que les données changent très peu jusqu'à la puissance 4. Il en va de même pour les appels *local RMI* et *distant RMI*. C'est à partir de la puissance 5 que la taille du fichier commence à faire une différence, et le temps de réponse ne cesse d'augmenter exponentiellement à partir de cette valeur.

Comment se fait-il que c'est juste à partir d'une certaine puissance que la performance commence à varier ? Cela s'explique simplement par la puissance des ordinateurs. Jusqu'à la puissance 4, les plus grands facteurs semblent simplement être l'ordonnancemment/priorité que donne le processeur, et vu la petite taille du fichier, même si le processus doit attendre un peu plus, le temps pour traiter le fichier est assez petit pour ne pas voir de différence. À partir de 5, la taille du fichier est assez grande pour que le processeur ait de la difficulté à traiter le fichier.

Ensuite, comment se fait-il que la performance varie selon le serveur ? Pour le serveur *distant RMI*, il y a tout d'abord le temps de transfert du fichier en tant que tel. Donc, la connexion au réseau importe énormément. Par contre, les tests ont été fait sur un réseau local, donc la vitesse de transfert est excellente. C'est donc expliqué par autre chose, qui explique aussi la raison pourquoi le *local RMI* est plus lent que le *local*. Le fait que les appels sont fait à des processus externes ajoutent du *overhead*. En d'autres mots, lorsque le client *local* fait un appel, il y a tout plein d'instructions cachés qui sont exécutés automatiquement pour permettre de communiquer avec un processus externe, même si c'est sur la même machine. Il y a donc encore plus de *overhead* si c'est une machine distante, puisqu'il y a encore plus de trucs à initialiser.

Pour résumé en d'autres mots, le temps d'exécution varie seulement à partir d'une certaine taille assez grande pour donner de la difficulté au processeur (dans ce cas-ci, c'est à partir de la puissance 5), et s'il y a un appel à un serveur, qu'il soit local ou distant, l'appel doit passer les couches RMI (et même OSI) qui ralenti tout l'appel.

Une mauvaise utilisation d'appels RMI est pour exécuter des fonctions qui ne dépendent pas de d'autres données. Par exemple, faire un appel à un serveur pour additionner deux chiffres que le client envoie est une mauvaise idée, car le *latency* de l'appel à un

serveur sera beaucoup plus grand qu'un à un appel local. Par contre, dans un système où il y a plusieurs utilisateurs qui partagent des données, cela pourrait faire du sens. Une première personne pourra envoyer au serveur les chiffres à traiter. Une deuxième personne pourra demander au serveur la somme des chiffres envoyés par la personne, sans même que la deuxième personne connaissent les chiffres utilisés pour faire le calcul.

Question 2

Pour comprendre comment se font les appels, il faut étudier la structure des couches RMI.

Tout d'abord le serveur va démarrer et s'enregistrer dans un registre qui va permettre aux clients de se connecter. Lorsque le client démarre, il va consulter ce registre pour retrouver le serveur qui lui intéresse, qui sera alors conserver en tant que stub chez le client. Selon la commande tappé, le *switch* du code va invoquer la méthode correspondante du stub. S'il y a des paramètres, les paramètres vont être sérialisés (marshmallow) pour pouvoir être transportable jusqu'au serveur. Rendu au serveur, les paramètres vont être désérialisés (unmarshmallow) pour que le serveur puisse lire les paramètres et les valeurs correspondantes. Le serveur va alors faire le traitement qu'il à faire. Lorsque celle-ci est terminé, la même opération va être effectué dans le sens inverse. Le serveur va sérialisé l'objet de retour, et à la reception, le client va désérialisé la réponse, et traiter l'objet reçu au besoin.

Instructions

Pour exécuter la partie 1

1. Allez dans le dossier bin, et partez rmiregistry en tappant "rmiregistry" dans la console. Laissez la console ouverte.

2. (Optionnel) Le projet est déjà compilé, mais vous pouvez le recompiler en tappant "./ant" à la racine du projet

3. Partez le serveur en tappant "./server". Laissez la console ouverte

4. Partez le client en exécutant le fichier bash. Il faut aussi lui donner un paramètre indiquant lequel des serveurs on veut appeler

    3 choix :

        - local

        - localrmi

        - distantrmi

    Exemple : ./script.sh local

Pour exécuter la partie 2

1. Allez dans le dossier bin, et partez rmiregistry en tappant "rmiregistry" dans la console. Laissez la console ouverte.

2. (Optionnel) Le projet est déjà compilé, mais vous pouvez le recompiler en tappant "./ant" à la racine du projet

3. Partez le serveur en tappant "./server". Laissez la console ouverte

4. Partez le client en tappant "./client". Il faut aussi ajouter une commande en paramètre, et au besoin (selon la commande), un paramètre de plus

    Commandes :

        - create <nomDuFichier>

        - list

        - syncLocalDir

        - get <nomDuFichier>

        - lock <nomDuFichier>

        - push <nomDuFichier>

    Exemple : ./client list