

**INF8480 – Systèmes répartis et infonuagique**

**Hiver 2019**

**TP1**

**Appels de méthodes à distance**

**Groupe 2**

**1791314 – Simon Nadeau**

**1846732 – Mathieu Châteauvert**

**Soumis à : Daniel Capelo & Adel Belkhiri**

**11/02/2019**

Q1. Commenter et expliquer les résultats obtenus.

Les résultats obtenus pour l’appel normal sont irréguliers. En effet, en regardant le graphique 1, on remarque clairement qu’il n’y a pas de tendance particulière, mais que les appels s’effectuent tous relativement rapidement. Les latences se situent tous autour de 2500 nanosecondes.

Les résultats obtenus pour les appels RMI local et distant suivent une tendance claire. En effet, la latence des appels est de plus en plus élevée au fur et à mesure que la taille des arguments augmente. Bien sûr, l’échelle logarithmique utilisée fait en sorte qu’il est difficile de déterminer avec précision la relation entre la taille des arguments et la latence des appels. Comme on peut le voir plus clairement sur le graphique 4, la courbe suit une tendance qui est plutôt linéaire.

Avantages et inconvénients

Un des principaux avantages de Java RMI est la capacité de télécharger la définition de la classe d’un objet même si cette classe n’est pas disponible localement. Il est donc possible de profiter des types et du comportement de l’objet même si celui n’est pas défini localement, mais seulement dans une autre machine. De cette façon, le comportement des objets n’est pas modifié même s’ils sont utilisés sur des machines distantes ce qui permet d’étendre facilement le comportement d’une application dans un programme distribué. De plus, cette méthode est sécuritaire et facile d’utilisation, car elle rend les aspects réseau transparents.

L’inconvénient principal du RMI est le même que dans plusieurs systèmes distribués : la latence est plus grande plus la taille des paquets augmente ce qui peut causer des problèmes de fiabilité. Un autre inconvénient est que cette technologie n’est supportée que par Java.

Q2.

Tout d’abord, il faut mettre en place le registre RMI sur chaque machine pour avoir accès à un annuaire des objets serveurs disponibles. Ensuite, du côté serveur, on crée une instance serveur, on recherche le registre RMI et on enregistre le serveur dans celui-ci afin que ses méthodes soient disponibles pour le client. Du côté client, on localise le registre RMI pour ensuite rechercher l’objet serveur dans celui-ci et créer une interface qui permettra au client d’accéder aux méthodes de l’objet serveur. Enfin, le client appel les méthodes du serveur.

1. **rmiregistry &** : Mettre en place un registre RMI offrant un annuaire des objets serveurs disponibles.

Coté Serveur

1. **UnicastRemoteObject.exportObject()** : Exporte l’objet (le serveur lui-même dans l’exemple fourni) pour faire en sorte qu’il puisse recevoir les appels entrants.
2. **LocateRegistry.getRegistry()** : Localise le registre RMI sur la machine.
3. **Registry.rebind()**: Remplacement de l’objet (le serveur dans l’exemple fourni) dans le registre RMI en lui accordant un nom.

**8. Server.execute()** : méthode ne faisant rien et prenant en argument un tableau de taille variable.

Côté Client

1. **LocateRegistry.getRegistry()** : Localise le registre RMI sur la machine par son nom (Adresse IP dans l’exemple fourni)
2. **Registry.lookup()** : Recherche d’un objet distant dans le registre RMI. Dans notre exemple, lookup recherche l’objet serveur pour en faire un interface serveur permettant de négliger les aspects réseaux. Nous avons donc ensuite accès aux méthodes de l’objet en question. Dans notre exemple, nous appelons la fonction *execute*.
3. **Server.execute()** : méthode du serveur.

Annexe

Tableau 1 des temps en nanosecondes selon la grosseur des paramètres passés en paramètre.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Taille des paramètres (byte) | Appel normal  (ns) | Appel RMI local  (ns) | Appel RMI distant  (ns) |
| 1 | 10 | 2399 | 1241602 | 3930195 |
| 2 | 100 | 2483 | 1676915 | 2007237 |
| 3 | 1000 | 2512 | 1345852 | 2346956 |
| 4 | 10000 | 2590 | 1332690 | 3796175 |
| 5 | 100000 | 2459 | 1435107 | 9910951 |
| 6 | 1000000 | 2546 | 3565611 | 88354654 |
| 7 | 10000000 | 2501 | 12170760 | 865108170 |