POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

INF8480

Systèmes répartis et infonuagique

Rapport de TP1

Présenté à

Houssem Daoud

Par

Anthony Abboud

Et

Luc Courbariaux

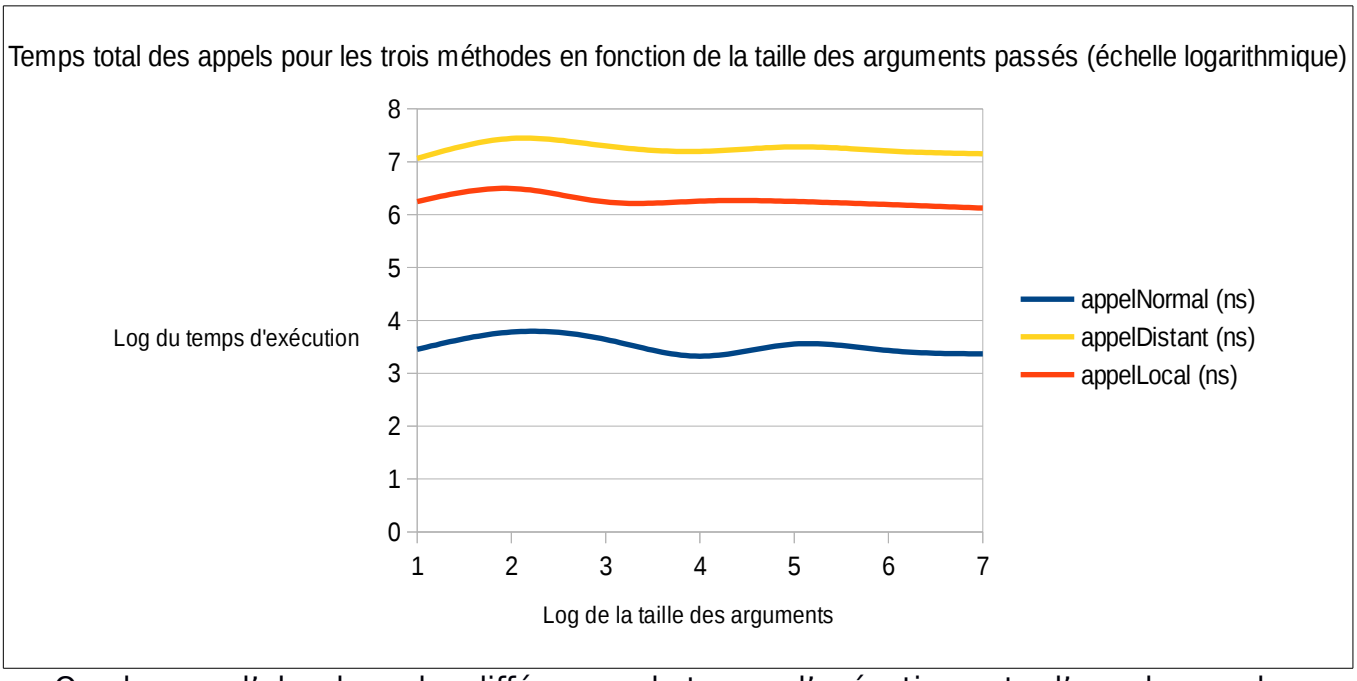
13 février 2018

Partie 1

Question 1

Taille des paramètres	appelNormal (ns)	appelLocal (ns)	appelDistant (ns)
10	2831	1766726	11576521
100	6021	3123583	27529702
1000	4370	1737149	19897993
10000	2104	1800506	15738295
100000	3563	1776311	19136476
1000000	2695	1553082	16014993
10000000	2327	1331211	14156513

Log(Taille)	Log(appelNormal)	Log(appelLocal)	Log(appelDistant)
1	3.4519398694	6.2471692004	7.0635780639
2	3.7796686272	6.4946530505	7.4398015103
3	3.640481437	6.2398370707	7.2988092737
4	3.3230457355	6.255394573	7.1969576815
5	3.5518158224	6.2495190052	7.2818619651
6	3.4305587695	6.1911943863	7.2045267532
7	3.3667963833	6.1242468976	7.1509562921



On observe d'abord que les différences de temps d'exécution entre l'appel normal et l'appel local sont très significatives (d'environ au moins 1000 fois). Le graphique logarithmique démontre que les résultats pour l'appel distant varient entre 10^3 et 10^4 tandis que les résultats pour l'appel distant varient entre 10^6 et 10^7.

Ceci est logique puisqu'on observera toujours un temps de latence supplémentaire lorsqu'on tente d'appeler une fonction qui n'est pas sur la même machine que le

client. Dans le cas d'appelLocal, la fonction est sur la même machine, mais c'est séparé par le fait que c'est dans un serveur.

Dans le même ordre d'idée un appel de fonction sur une machine distante prendra encore plus de temps que pour un appel sur machine local (la distance est plus grande et il y a un plus grand nombre d'étapes de protocole permettant l'envoi et la réception de données).

Enfin, on peut observer les avantages et désavantages suivants par rapport à Java RMI.

Avantages :

- Facile à implémenter
- Il y a un chargement de classes dynamique
- Le client peut être mis à jour de façon évidente par le serveur
- L'exécution distante de la méthode est masquée au code client; il y a donc une couche de sécurité déjà en place.
- C'est orienté objet

Désavantages :

- C'est utilisé strictement en Java; on ne peut pas l'utiliser avec un code d'un autre langage.
- On peut parfois observer de très grands temps de latence

Question 2

Selon ce qu'on a vu en classe, les notes de cours et les expérimentations du code :

1. Le serveur charge les classes Stub et Skel d'après java.rmi.server.codebase

   Dans notre code, cela correspond à la classe ServerInterfaceaux créations des variable de type ServerInterface (localServer, localServerStub et distantServerStub)

2. Le serveur enregistre l'objet distant : il communique une instance de Stub

   Dans le code, l'instance de Stub communiquée est fait avec ''ServerInterface Stub = (ServerInterface) UnicastRemoteObject.exportObject(this, 0);'' (voir fichier Server.java, ligne 28)

3. Le client réclame le Stub associé à ''HelloObject''

   Les Stub sont réclamés à l'aide de la méthode loadServerStub, dans Client.java.

4. Rmiregistry retourne le Stub au client

Rmiregistry est représenté par la classe Registry. Pour retourner le Stub au client, on utilise registry.rebind(serveur, stub); du côté du serveur et puis registry.lookup(serveur); du côté du client dans la méthode loadServerStub.

Le Stub est maintenant prêt à être utilisé par le client.

5. Le client invoque une méthode sur le Stub

   Dans notre cas, cela correspond à notre fonction de test.