INF8480 - SYSTÈMES RÉPARTIS ET INFONUAGIQUE

TRAVAIL PRATIQUE 1
APPELS DE MÉTHODES À DISTANCE

PRÉSENTÉ À :

HOUSSEM **DAOUD**

ANAS **BALBOUL**

PAR :

1773922, ÉTIENNE **ASSELIN**

1744784, VINCENT **RODIER**

1734142, FRÉDÉRIC **BOUCHARD**

DATE : 13 FÉVRIER 2018

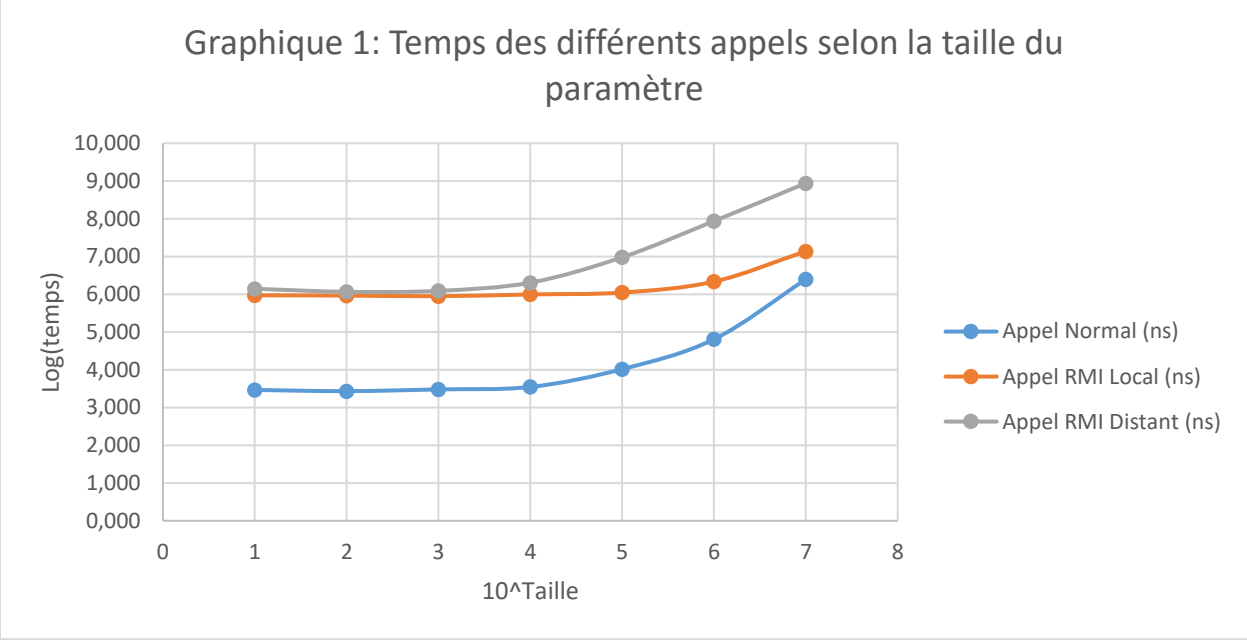
Question 1

À la suite de l'exécution de la commande nous permettant de calculer le temps des appels des différentes méthodes (Appel Normal, RMI Local & RMI Distant), nous avons obtenus ces différents résultats.

Tableau 1 : Temps des différents appels selon la taille du paramètre

Taille (Octet)	LOG (Taille)	Appel Normal (ns)	LOG (Appel Normal)	Appel RMI Local (ns)	LOG (Appel RMI Local)	Appel RMI Distant (ns)	LOG (Appel RMI Distant)
10	1	2921	3,466	939148	5,973	1395078	6,145
100	2	2714	3,434	923894	5,966	1160931	6,065
1000	3	3018	3,480	889285	5,949	1232144	6,091
10000	4	3510	3,545	990714	5,996	2032958	6,308
100000	5	10388	4,017	1111158	6,046	9541134	6,980
1000000	6	64321	4,808	2166311	6,336	86975704	7,939
10000000	7	2485560	6,395	13758649	7,139	877504684	8,943

Ce tableau nous a permis de confectionner le graphique suivant.



On remarque que le temps d'un appel normal est considérablement plus rapide que des appels RMI. Ce résultat est notamment dû au fait que l'appel normal ne passe par aucun proxy. Il est intéressant de comparer les appels RMI local et distant. Pour une taille inférieure à 1000 octets, ces deux appels prennent un temps sensiblement identique. Néanmoins, le temps d'un appel local est légèrement plus rapide en raison du temps de latence en moins entre le client et le serveur *Openstack*. Un phénomène particulier se

produit pour des tailles excédant 1000 octets. Le temps d'un appel distant augmente considérablement en raison de la taille maximale des paquets passant sur le réseau. En effet, la taille maximale d'un paquet IP est de 1500 octets. Ainsi, lors d'un appel excédant cette taille il faut fractionner les données et les insérer dans plusieurs paquets.

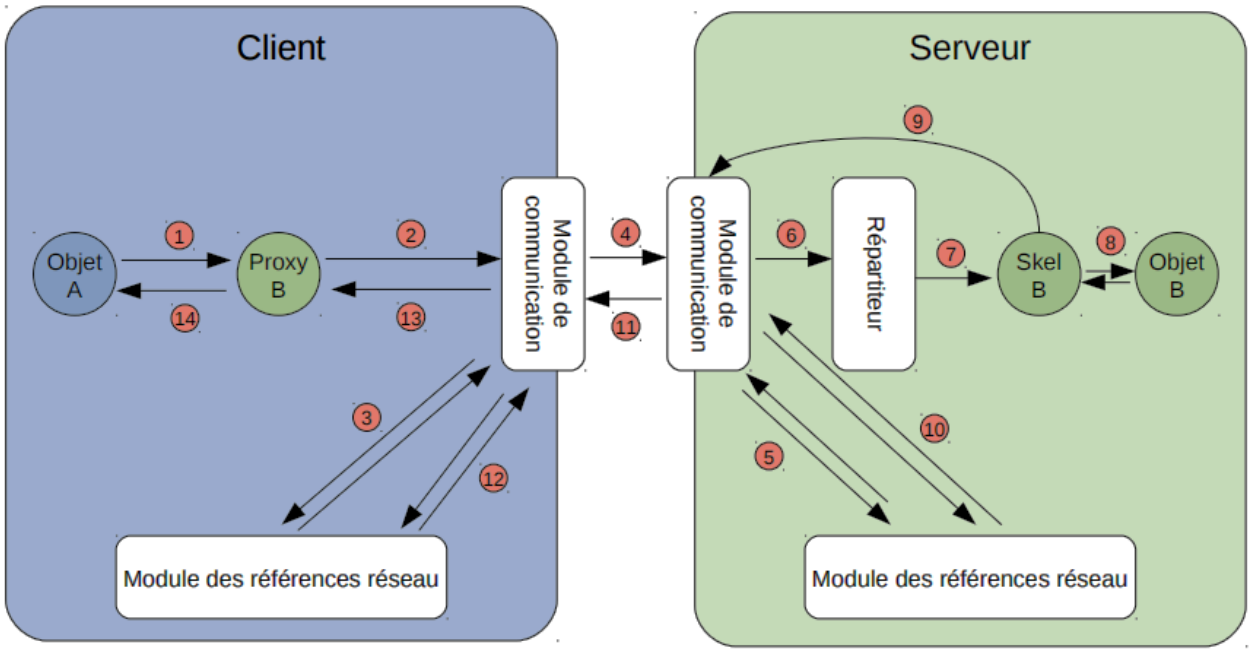
Il existe plusieurs avantages à utiliser Java RMI. En effet, il est facile de faire des appels car Java RMI s'applique à tous les types qui implantent l'interface *Remote*. De plus, le RMI *Registry* maintient une table pour obtenir une référence à un objet désirer qui se trouve sur un ordinateur donné (*//hostname :port/objectname*).

Cependant, Java RMI amène son lot d'inconvénients. D'une part, chaque objet passé en paramètre par une fonction Java RMI doit pouvoir être sérialisable. Ceci diminue la flexibilité des appels pouvant être fait. D'autre part, Java RMI fonctionne seulement pour un client et serveur Java. Cela restreint le développement d'un système sous d'autres langages de programmation.

Question 2

La figure suivante permet de mieux illustrer le cheminement des appels entre le client et le serveur.

Figure 1 : Fonctionnement de Java RMI pour un client et un serveur



Voici les 14 étapes lors d'un appel entre un client et un serveur.

1. L'objet A appelle une méthode locale sur le proxy de B.
2. Le proxy transmet au module de communication.
3. Le module de communication du client convertit la référence locale à B en une référence à l'objet distant grâce à la table des objets importés.
4. La requête est envoyée au serveur.
5. Le module de communication du serveur convertit la référence distante à B en une référence locale grâce à la table des objets exportés.
6. Le module de communication transmet la requête au répartiteur dans notre cas nous n'avons pas de répartiteur.
7. Le répartiteur identifie le bon squelette et lui transmet la requête.
8. Le squelette exécute la méthode de l'objet local et récupère la réponse.
9. Le squelette transmet la réponse au module de communication.
10. Le module de communication du serveur convertit la référence locale à B en une référence distante au proxy de B grâce à la table des objets exportés.
11. La réponse est retournée au client.
12. Le module de communication du client convertit la référence distante au proxy de B en une référence locale grâce à la table des objets importés.
13. Le module de communication du client retourne la réponse au proxy.
14. L'objet A reçoit finalement la réponse.

Voici les correspondances des différents objets selon notre implémentation.

- Objet A : Classe client
- Proxy B : Classe ServerInterface (distantServerStub)
- Module de référence réseau: Généré par l'appel de *rmiregistry*
- Module de communication : Socket pour TCP
- Skel B : Interface ServerInterface
- Objet B : Classe Server
- Répartieur : Inexistant