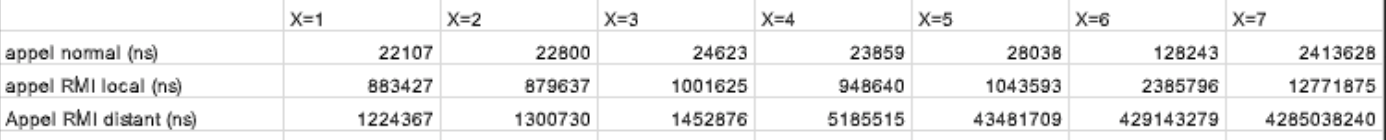
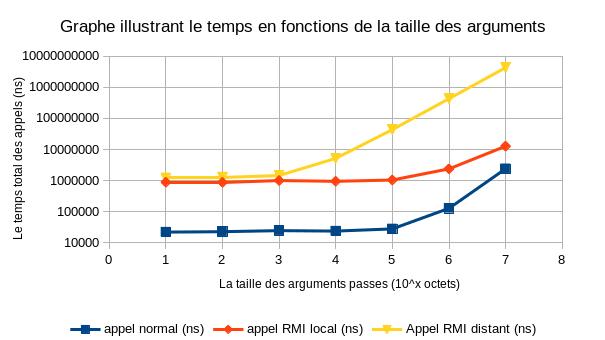
**Question 1:**

****

Le tableau représente le temps total en fonction de la taille de l’argument passé à méthode. le temps est en nanosecondes et la taille est 10^x octets.



**Commentez et expliquez les résultats obtenus:**

On voit dans le graphe (échelle logarithmique) le temps total des appels de fonctions (appel normal, appel RMI local, appel RMI distant). entre 10 et 1000 les 3 graphes sont presque parallèle, et on que l’appel normal a le moins duree apres on trouve l’appel RMI local et appel RMI distant qui sont très proches mais qui prennent une durée très grande par rapport l’appel normal, et ce résultat est a cause de l’appel de RMI(Romote Method Invocation). L’’appel RMI distant prend plus du temps pour s'exécuter puisque il demande d’utilisation du réseau qui lui aussi prend du temps. Entre 1000 octets et 100000 octets on voit que l’appel RMI distant diverge rapidement alors que l’appel RMI local et l’appel normal restent parallèle. Transmettre un argument gourmand (100000 octets) va prendre plus de temps pendant la transmission, alors que pour l’appel normal et l’appel local ne demande pas d’utilisation du réseau, donc on ls trouvent encore parallèle. l’envoi de plus de 100000 octets a influence sur l’appel normal et local en termes de durée, et comme on voit sur le graphe la durée augmente rapidement, et que la durée de l’appel normal augmente avec plus d’acceleration.

**Discutez les avantages et les inconvénients de Java RMI:**

**Avantages:**

Le programmeur est capable d’utiliser tout le pouvoir expressif de la programmation orientée objet dans le développement de systèmes distribués, y compris l’utilisation d’objets, de classes et d’héritage, et peut également utiliser des méthodologies de conception et outils associés.

S'appuyant sur le concept d'identité d'objet dans les systèmes orientés objet, tous les objets d'un système basé sur RMI ont des références d'objet uniques (qu'ils soient locaux ou distants). De telles références d'objet peuvent également être transmises en tant que paramètres, offrant ainsi des paramètres beaucoup plus riches.

Autres avantages:

1. Gère les threads.
2. Gère les sockets.
3. Marshall les objets.
4. Load dynamique des classes.
5. Client et serveur sont indépendants.

**Inconvénients:**

l’un des inconvénients de java RMI est qu’il ne peut fonctionner que de Java à Java. Impossible de garantir qu'un client utilisera toujours le même thread dans des appels consécutifs. Cela signifie que vous devez écrire un mécanisme pour identifier vous-même le client.

**Question 2:**

Le serveur implémente l’interface ServerInterface (la méthode execute()). Dans le serveur on trouve la fonction main() qui crée un objet qui peut être appelé à distance à partir d’un autre host. On trouve aussi l’implémentation des méthodes de l’objet RMI ‘server’. Dans la méthode run() on trouver un gestionnaire de sécurité pour permettre à java d'appliquer la protection appropriée pour un serveur RMI. Un gestionnaire de sécurité par défaut appelé RMISecurityManager est fourni. Il protège les ressources locales pour assurer que les classes chargées à partir de sites distants ne peuvent avoir aucun effet sur

des ressources telles que les fichiers.

la méthode exportObject (définie sur UnicastRemoteObject) est utilisée pour mettre cet objet à la disposition du RMI, le rendant ainsi disponible pour recevoir les appels entrants. Le deuxième paramètre de exportObject spécifie le port TCP à utiliser pour les appels entrants.

Après, le RMIregistry met la référence de l’objet (“server”) dans le registre pour que le client peut le trouver.

le client commence par installer un gestionnaire de sécurité. Cette étape est nécessaire car le processus de réception du stub de l'objet distant peut nécessiter le téléchargement de définitions de classe à partir du serveur. Pour que RMI puisse télécharger des classes, un gestionnaire de la sécurité doit être en vigueur. Après avoir installé un gestionnaire de sécurité, le client construit un nom à utiliser pour rechercher un objet distant server, en utilisant le même nom que celui utilisé par le serveur pour lier son objet distant. De plus, le client utilise l'API LocateRegistry.getRegistry pour synthétiser une référence distante au registre sur le serveur. La valeur du premier argument de ligne de commande, args [0], est le nom de l'hôte distant sur lequel l'objet server est exécuté. Le client appelle ensuite la méthode de recherche sur le registre pour rechercher l'objet distant par son nom dans le registre du serveur. LocateRegistry.getRegistry est utilisée, et qui possède un paramètre String unique, renvoie une référence à un registre de l'hôte nommé et au port de registre par défaut, 1099. Après, le client invoque la méthode execute() du serveur.

les interactions entre les différents acteurs sont cachée et le programmeur ne voit pas toutes les communications entre le serveur, registre RMI et le client.

1. le serveur charge les classes Stub et Skel d ’après java.rmi.server.codebase
2. le serveur enregistre l’objet distant dans le registre RMI : il communique une instance de stub (rebind(“server”)).
3. le client réclame le stub associé à "server", lookup(“server”).
4. rmiregistry retourne le stub.
5. le client charge la classe Stub d ’après java.rmi.server.codebase.
6. le client invoque une méthode sur le stub, Stub.execute().
7. le stub prend les paramètres de la méthode et les envoie au serveur.
8. le skeleton invoque la méthode sur l’objet et retourne le résultat.

le photos en bas illustrent les différentes communications :

