REPUBLIQUE TUNISIENNE

Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique

Rapport de Projet

Travail de :

Fares Chachia

TABLE DES MATIÈRES

I. QU'EST CE QUE UN SYSTÈME DISTRIBUÉ ?

1) Définition :

Un système distribué est un environnement informatique dans lequel divers composants sont répartis sur plusieurs ordinateurs (ou autres dispositifs informatiques) appartenant à un même réseau. Ces appareils divisent le travail et coordonnent leurs efforts pour effectuer des tâches plus efficacement qu'un seul appareil.

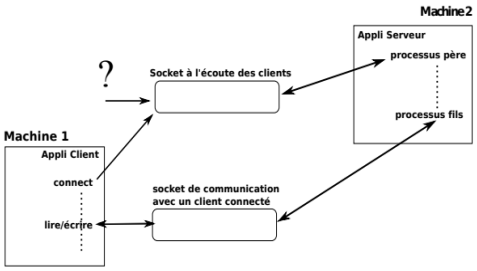
2) Caractéristiques :

- **l'évolutivité.** La capacité à croître à mesure que la taille de l'applicatif augmente est une caractéristique essentielle des systèmes distribués, qui s'obtient en ajoutant des unités de traitement ou des nœuds supplémentaires au réseau en fonction des besoins ;
- **la concurrence.** Les composants du système distribué s'exécutent simultanément. Ils se caractérisent également par l'absence d'« horloge globale », les tâches se produisant dans le désordre et à des rythmes différents .
- **la disponibilité/tolérance aux interruptions de service.** Si un nœud arrête de fonctionner, les nœuds restants peuvent continuer à fonctionner sans perturber le calcul global .
- **la transparence.** Un programmeur externe ou un utilisateur final voit le système distribué comme une unité de calcul unique et non comme ses parties sous-jacentes, ce qui lui permet d'interagir avec un seul périphérique logique sans avoir à se préoccuper de l'architecture du système .
- **l'hétérogénéité.** Dans la plupart des systèmes distribués, les nœuds et les composants sont souvent asynchrones et répartis sur des machines, du middleware, des logiciels et des systèmes d'exploitation différents. Cela permet d'élargir les systèmes distribués en ajoutant de nouveaux composants .
- **la réplication.** Les systèmes distribués permettent le partage des informations et des messages dans un objectif de cohérence des ressources redondantes telles que les composants logiciels ou matériels, ce qui améliore la tolérance aux interruptions de service, la fiabilité et l'accessibilité.

II. LES SOCKETS :

1) Définition :

Un socket constitue un mécanisme de communication entre processus du système Unix/Linux (mais pas exclusivement). Il sert d'interface entre les applications et les couches réseau. Il existe pour cela une bibliothèque système (ensemble de fonctions/primitives) permettant de gérer les services de communication offerts par les sockets. Le principe général de la communication entre applications (dites 'serveur' et 'client') est illustré dans la figure ci-joint



2) Implémentation des sockets : (Service de chat)

1. Serveur de Chat :

- Le serveur est le point central où les clients se connectent pour envoyer et recevoir des messages.
- Il écoute sur un port spécifique pour les connexions entrantes des clients.
- Une fois qu'un client se connecte, le serveur crée un socket dédié pour cette connexion.
- Le serveur gère les messages entrants et sortants des clients, les distribuant aux destinataires appropriés.
- Il peut également gérer la création et la suppression de salles de discussion, ainsi que la gestion des autorisations d'accès.

2. Client de Chat :

- Chaque client possède son propre socket pour se connecter au serveur.
- Lorsque le client se connecte, il peut envoyer des messages au serveur et recevoir des messages des autres clients.
- Le client peut également gérer les actions telles que la connexion, la déconnexion, la création de nouveaux messages, etc.

3.Communication via Sockets :

- La communication entre le serveur et les clients se fait via des sockets.
- Lorsqu'un client envoie un message, il l'envoie au serveur via son socket.
- Le serveur reçoit ce message et le redistribue à tous les autres clients connectés, à l'exception de l'expéditeur.
- De même, lorsque le serveur reçoit un message d'un autre client, il le transmet à tous les autres clients.

En somme, l'implémentation des sockets dans un service de chat repose sur la mise en place d'une architecture client-serveur robuste permettant la communication en temps réel entre les utilisateurs connectés.

3) Avantages et Limites des sockets :

+ Flexibilité et Contrôle : Les sockets offrent un haut degré de flexibilité et de contrôle sur la communication réseau. Les développeurs ont la possibilité de personnaliser et de contrôler les aspects de la communication tels que le protocole, le flux de données et la gestion des connexions.

+ Interopérabilité : Les sockets sont largement pris en charge par de nombreuses plateformes et langages de programmation. Cela permet une interopérabilité élevée entre les systèmes hétérogènes, ce qui facilite la communication entre différentes applications et environnements.

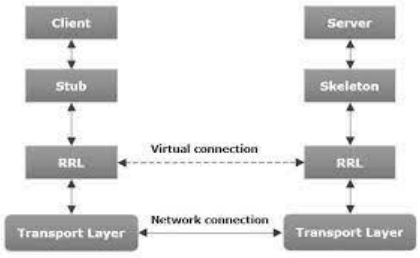
- Complexité : Implémenter une communication réseau basée sur des sockets peut être complexe, en particulier lorsqu'il s'agit de gérer des aspects tels que la synchronisation, la gestion des erreurs et la sécurité. Les développeurs doivent souvent écrire beaucoup de code pour gérer ces aspects, ce qui peut rendre le développement plus difficile et augmenter les risques d'erreurs.

- Niveau d'Abstraction Bas : Les sockets fournissent un niveau d'abstraction relativement bas pour la communication réseau. Cela signifie que les développeurs doivent souvent gérer des détails de bas niveau tels que la gestion des connexions, la sérialisation/désérialisation des données et la gestion des erreurs. Pour les applications nécessitant une communication réseau plus complexe, cette faible abstraction peut entraîner une augmentation de la complexité du code et une augmentation de la charge de travail du développement.

III. JAVA RMI :

1) Définition :

Java RMI (Remote Method Invocation) est une technologie de communication distribuée en Java qui permet à un programme Java d'appeler des méthodes sur des objets distants, comme s'ils étaient des objets locaux.



2) Implémentation de Java RMI (Gestion d'une liste de tâches)

1. Interface distante (`TaskList`) :

- Cette interface définit les méthodes que le client peut invoquer sur le serveur. Elle hérite de l'interface `Remote` pour indiquer qu'elle est destinée à être utilisée à distance.
- Trois méthodes sont définies : `addTask`, `removeTask` et `getAllTasks`, qui permettent respectivement d'ajouter une nouvelle tâche, de supprimer une tâche existante et de récupérer la liste complète des tâches.

2. Implémentation du serveur (`TaskListImpl`) :

- Cette classe implémente l'interface `TaskList`. Elle étend également `UnicastRemoteObject`, ce qui signifie qu'elle peut être exportée en tant qu'objet distant.
- La liste des tâches est stockée localement dans cette classe.
- Les méthodes `addTask`, `removeTask` et `getAllTasks` sont implémentées pour fournir les fonctionnalités définies dans l'interface distante.

3. Code du serveur RMI (`TaskListServer`) :

- Cette classe contient la méthode `main` qui démarre le serveur RMI.
- Dans cette méthode, une instance de `TaskListImpl` est créée et enregistrée auprès du registre RMI avec le nom symbolique "TaskListService".
- Le serveur RMI utilise le registre RMI pour permettre aux clients de rechercher et d'accéder au service.

4. Code du client RMI (`TaskListClient`) :

- Cette classe contient également la méthode `main`, qui est le point d'entrée pour le client RMI.
- Dans cette méthode, le client obtient une référence distante au service à l'aide du registre RMI en utilisant le nom symbolique "TaskListService".
- Le client peut ensuite appeler les méthodes de l'interface `TaskList` sur cette référence distante pour interagir avec le serveur RMI.

3) Avantages et Limites de Java RMI :

+ Simplicité d'Utilisation : Java RMI simplifie la communication entre des objets distribués en permettant aux développeurs de faire des appels de méthodes sur des objets distants de manière transparente, comme s'ils étaient des objets locaux. Cela permet de créer des applications distribuées de manière plus intuitive **.**

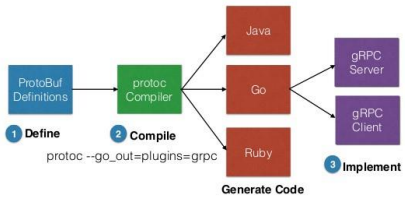
+ Intégration Transparente avec Java : Java RMI s'intègre de manière transparente avec le langage Java, ce qui signifie qu'il utilise les mêmes concepts et les mêmes outils que ceux utilisés pour le développement Java standard. Cela simplifie le processus de développement pour les développeurs Java et réduit la courbe d'apprentissage pour l'utilisation de la technologie.

- Interopérabilité Limitée : Java RMI est spécifique à la plateforme Java, ce qui signifie qu'il peut avoir des difficultés à interagir avec des systèmes développés dans d'autres langages ou des plates-formes non Java. Cela peut limiter l'interopérabilité avec les systèmes existants qui ne sont pas basés sur Java.

- Complexité de la Configuration : La configuration et le déploiement de Java RMI peuvent être complexes, en particulier pour les applications distribuées plus complexes. La nécessité de démarrer un registre RMI et de configurer correctement les autorisations de sécurité peut ajouter de la complexité au processus de déploiement et de gestion des applications.

IV. GRPC:

1) Définition :



gRPC est un Framework RPC universel Open Source qui offre de hautes performances et est développé par Google. Dans gRPC, une application cliente peut appeler directement des méthodes sur une application serveur située sur une machine différente, comme s'il s'agissait d'un objet local, facilitant ainsi la création d'applications et de services distribués.

2) Implémentation de gRPC(Service de messagerie)

1. Fichier de définition du service (`messaging.proto`) :

- Ce fichier définit le service de messagerie à l'aide de la syntaxe Protobuf. Il spécifie les méthodes du service, leurs paramètres et leurs types de réponse.
- Le service `MessagingService` comprend deux méthodes RPC : `SendMessage` pour envoyer un message et `GetMessagesForUser` pour récupérer les messages pour un utilisateur donné.
- Les messages `MessageRequest`, `MessageResponse`, `UserRequest` et `MessagesResponse` sont également définis pour décrire les données échangées entre le client et le serveur.

2. Implémentation du serveur (`MessagingServer`) :

- Cette classe contient la méthode `main` qui démarre le serveur gRPC.
- Le serveur écoute sur le port 9090 et utilise `MessagingServiceImpl` comme implémentation du service de messagerie.
- `MessagingServiceImpl` est une classe interne qui étend `MessagingServiceGrpc.MessagingServiceImplBase` et implémente les méthodes du service de messagerie définies dans le fichier `.proto`.

3. Implémentation des méthodes du service :

- Dans `MessagingServiceImpl`, les méthodes `SendMessage` et `GetMessagesForUser` sont implémentées pour fournir la fonctionnalité de messagerie.
- Pour chaque méthode, la logique métier correspondante peut être mise en œuvre. Par exemple, `SendMessage` pourrait envoyer le message à un destinataire spécifié et `GetMessagesForUser` pourrait récupérer les messages pour un utilisateur donné dans une base de données ou une autre source de données.

3) Avantages et Limites des sockets :

+ Haute Performance :gRPC est basé sur le protocole HTTP/2, qui offre des performances améliorées par rapport à HTTP/1.1. Il prend en charge la multiplexage, la compression, la priorisation des requêtes et d'autres fonctionnalités qui réduisent la latence et améliorent l'efficacité du réseau. Cela permet des échanges de données plus rapides et plus efficaces entre les clients et les serveurs.

+ Interopérabilité Multiplateforme : gRPC prend en charge plusieurs langages de programmation, ce qui permet aux clients et aux serveurs d'être implémentés dans différents langages. Google fournit des bibliothèques gRPC pour de nombreux langages, y compris Java, C++, Python, Go, JavaScript (Node.js), Ruby et bien d'autres. Cela favorise l'interopérabilité entre les différents composants d'un système distribué.

- Complexité de Configuration La configuration de gRPC peut être plus complexe que d'autres solutions de communication, en particulier pour les développeurs moins expérimentés. Configurer correctement les autorisations de sécurité, gérer les certificats TLS/SSL et optimiser les performances peuvent nécessiter une expertise supplémentaire. Cela peut rendre la mise en œuvre initiale et la maintenance plus complexes.

- Surcoût de la Sérialisation : Bien que Protobuf (le format de sérialisation par défaut de gRPC) soit plus efficace que JSON ou XML en termes de taille des données et de vitesse de sérialisation/désérialisation, il peut y avoir un surcoût initial de la sérialisation lors de l'utilisation de gRPC, en particulier pour les petites charges de travail où le surcoût de la sérialisation pourrait ne pas être justifié par les avantages de performance.

V. CONCLUSION :

En résumé, le choix entre les sockets, Java RMI et gRPC dépend des besoins spécifiques de votre application. Les sockets offrent une flexibilité maximale mais nécessitent une mise en œuvre détaillée. Java RMI offre une abstraction plus élevée, mais est limité à l'écosystème Java. gRPC offre une performance élevée et une interopérabilité multiplateforme, mais peut nécessiter une configuration plus complexe.