1 Introduction

Ce projet implémente un système de chat distribué en Java, utilisant RMI (Remote Method Invocation) et l'algorithme d'horodatage logique de Lamport. L'objectif est de permettre à plusieurs clients de communiquer en temps réel tout en garantissant la cohérence et l'ordre causal des messages, même dans un environnement distribué.

2 Présentation du Projet

Le système se compose de plusieurs clients (interface graphique ou ligne de commande) et d'un serveur centralisé, pouvant être déployé dans un conteneur Docker. Les clients se connectent au serveur, envoient et reçoivent des messages, et chaque événement (envoi/réception) est horodaté grâce à l'algorithme de Lamport.

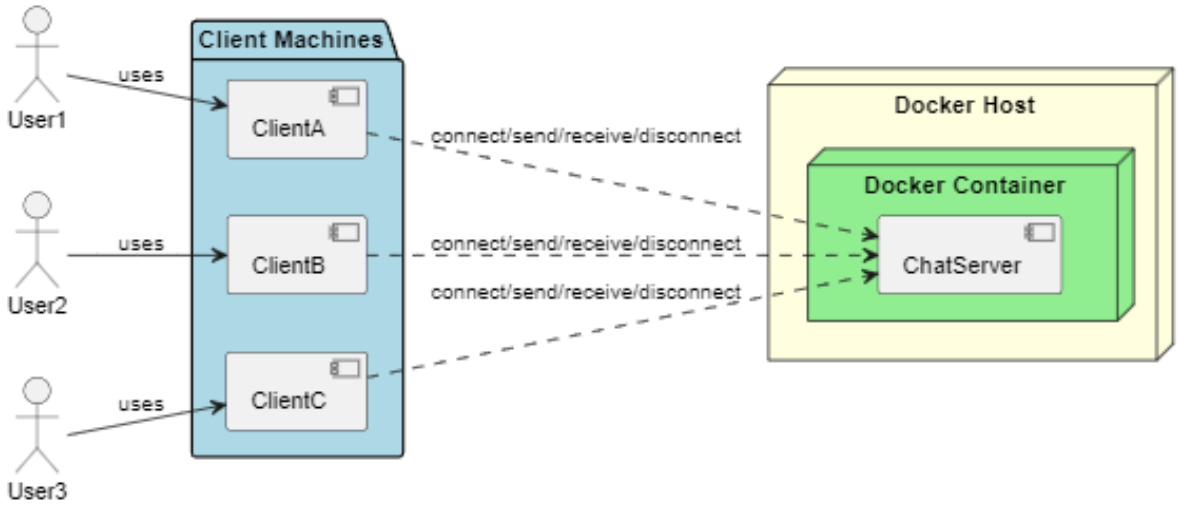

FIGURE 1 – Architecture générale du système distribué

L'architecture comprend :

— Des clients (A, B, C) utilisés par différents utilisateurs.
— Un serveur de chat centralisé, déployé dans un conteneur Docker.
— Des échanges de messages via RMI, avec gestion des connexions, envois, réceptions et déconnexions.

Contraintes d'implémentation :

— L'historique des messages sur le serveur est limité à 50 messages pour garantir de bonnes performances ; lorsqu'un nouveau message arrive alors que la limite est atteinte, le plus ancien des 50 messages est supprimé pour faire de la place au nouveau.
— Deux utilisateurs ne peuvent pas se connecter avec le même nom : le serveur impose l'unicité des noms d'utilisateur.

3 Diagramme de Classes

Le diagramme suivant présente les principales classes et interfaces du projet, incluant la gestion des clients, du serveur, des messages et de l'horloge de Lamport.

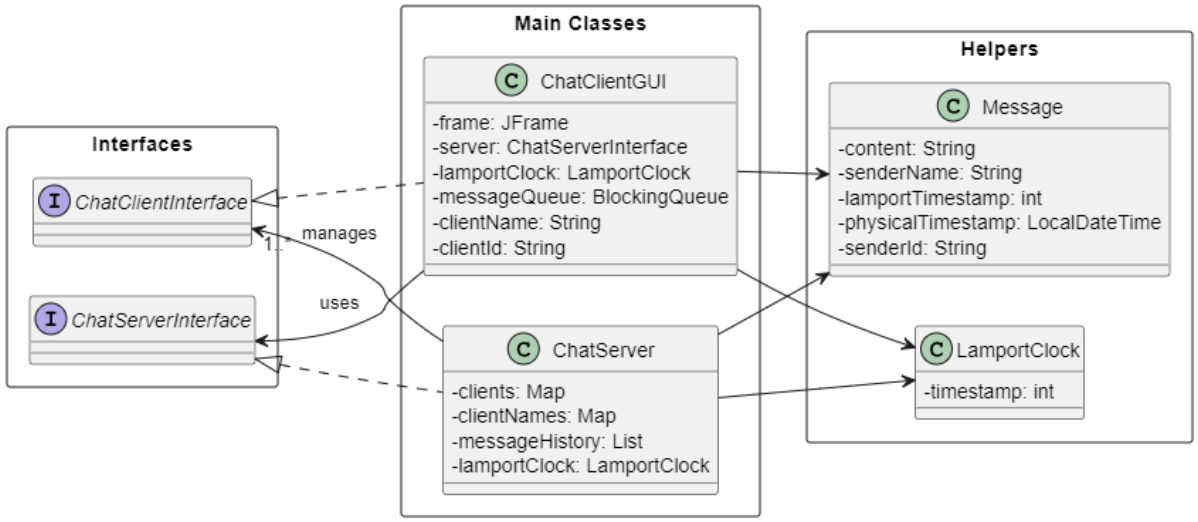

FIGURE 2 – Diagramme de classes du système de chat distribué

4 Algorithme de Lamport : Présentation et Fonctionnement

L'algorithme de Lamport permet d'ordonner les événements dans un système distribué à l'aide d'horodatages logiques. Chaque processus (client ou serveur) possède une horloge Lamport (un entier). Les règles sont :

1. Avant chaque événement local (envoi de message), le processus incrémente son horloge.
2. Lorsqu'un message est envoyé, il est accompagné de la valeur courante de l'horloge.
3. À la réception d'un message, le processus met à jour son horloge à `max(son_horloge, horloge_reçue) + 1`.

4.1 Présentation de l'algorithme

L'algorithme de Lamport, proposé par Leslie Lamport en 1978, vise à résoudre le problème de l'ordre des événements dans un système distribué où il n'existe pas d'horloge physique globale. Dans un tel environnement, il est difficile de déterminer l'ordre exact des événements (comme l'envoi ou la réception de messages) car chaque machine possède sa propre horloge locale. L'algorithme introduit le concept d'horodatage logique : chaque événement se voit attribuer un numéro croissant, garantissant ainsi un ordre partiel cohérent entre tous les nœuds du système. Cela permet d'assurer la cohérence causale des messages et d'éviter les conflits liés à la simultanéité des actions dans le système distribué.

Listing 1 – Horloge logique de Lamport (pseudo-code)

```
class LamportClock {
    int compteur = 0;

    // Incrementer pour chaque evenement local (envoi de message)
    synchronized int tick() {
        return ++compteur;
    }

    // Mettre a jour lors de la reception d'un message
    synchronized void update(int horlogeRecue) {
        compteur = max(compteur, horlogeRecue) + 1;
    }

    // Obtenir la valeur courante
    synchronized int get() {
        return compteur;
    }
}
```

4.2 Étapes de l'algorithme dans le chat

1. Le client A veut envoyer un message : il incrémente son horloge, puis envoie le message avec l'horodatage.
2. Le serveur reçoit le message, met à jour sa propre horloge, puis diffuse le message aux autres clients.
3. Le client B reçoit le message, met à jour son horloge selon la règle de Lamport.
4. À chaque nouvel événement (envoi/réception), l'horloge est ajustée pour garantir l'ordre causal.

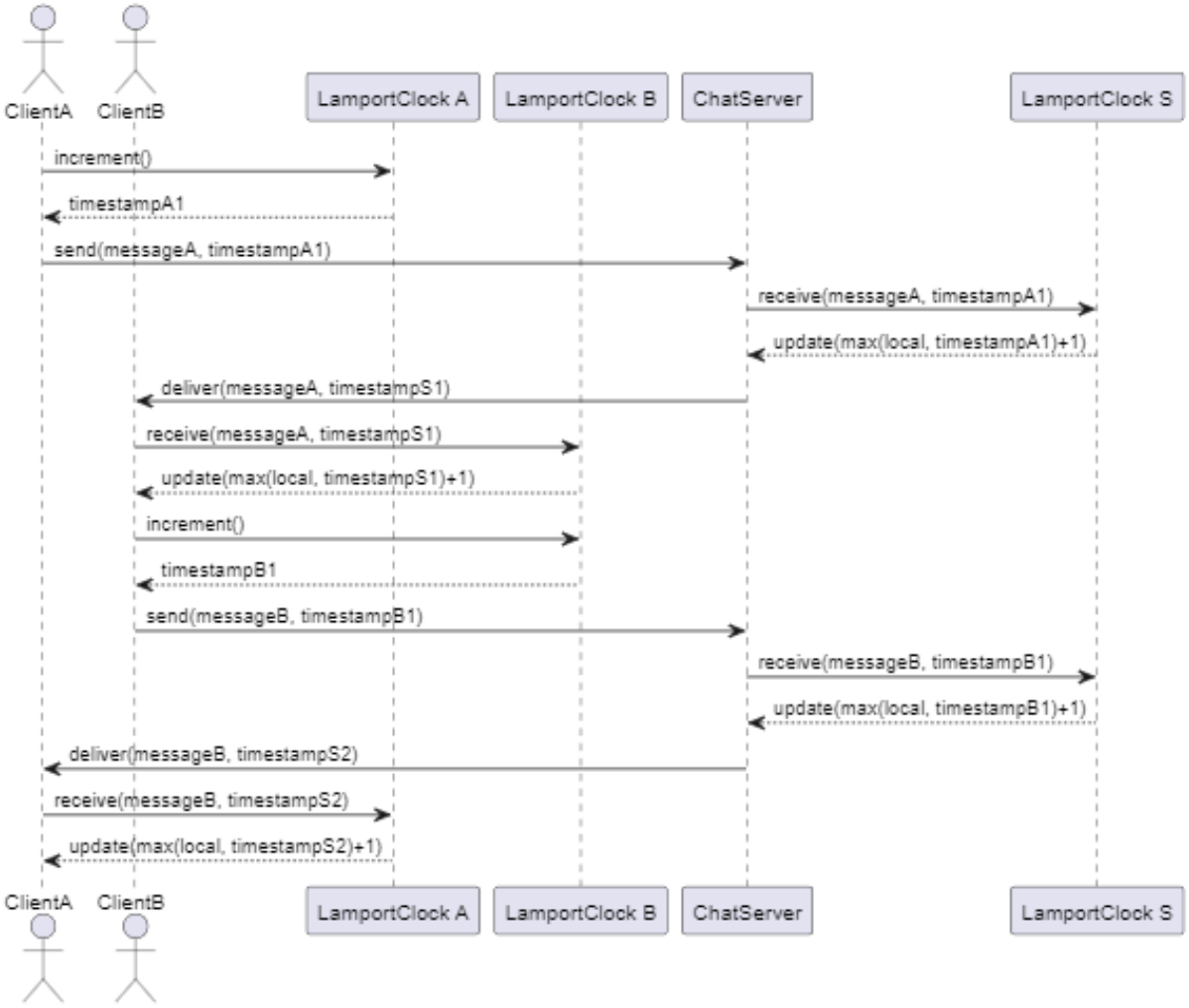

FIGURE 3 – Diagramme de séquence illustrant l'algorithme de Lamport dans le chat

5 comment executer le projet

5.1 Lancer le serveur avec Docker

1. Assurez-vous d'avoir Docker et Docker Compose installés sur votre machine.
2. Ouvrez un terminal dans le dossier du projet contenant le fichier `docker-compose.yml`.
3. Lancez le serveur avec la commande suivante :

Listing 2 – Démarrage du serveur avec Docker

```
docker-compose up -d
```

4. Pour vérifier que le serveur fonctionne, vous pouvez consulter les logs :

Listing 3 – Afficher les logs du serveur

```
docker-compose logs -f
```

5. Le serveur de chat est maintenant prêt à accepter des connexions clients.

5.2 Lancer un client GUI (Windows)

1. Ouvrez un terminal PowerShell dans le dossier du projet.
2. Exécutez le script PowerShell suivant pour démarrer l'interface graphique du client :

Listing 4 – Démarrage du client GUI sous Windows

```
./start-client-gui.ps1
```

3. Suivez les instructions à l'écran pour vous connecter au serveur. L'utilisateur doit uniquement saisir son nom ; l'adresse et le port du serveur sont automatiquement récupérés depuis le fichier `config.properties`.
4. Vous pouvez ouvrir plusieurs terminaux et lancer plusieurs clients pour simuler plusieurs utilisateurs.

6 Conclusion

Ce projet démontre l'utilisation de l'algorithme de Lamport pour assurer la cohérence des messages dans un système de chat distribué. Grâce à l'architecture modulaire et à l'intégration de Docker, le système est facilement déployable et extensible.