جامعة عبد المالك السعدي
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵄⴰⴱⴷⵍⵎⴰⵍⵉⴽ ⵙⵙⴰⵄⴷⵉ
Université Abdelmalek Essaâdi

Département Informatique

Master : D'ingénierie informatique

Rapport du projet

Conception et Implémentation d'une Application Distribuée pour le Calcul Matriciel Parallèle en Java

Réalisé par : El Mghari Aya

El Afia Mohamed

Encadré par : Prof. Abdelaziz El Hibaoui

2024/2025

Table des matières

Table de figures

I. Introduction

Dans un contexte où les calculs matriciels jouent un rôle clé dans divers domaines scientifiques et techniques, optimiser leur traitement est une nécessité. Ce projet propose une solution distribuée en Java, basée sur RMI et des sockets, afin d'accélérer l'exécution des opérations matricielles en répartissant la charge sur plusieurs serveurs. En divisant les matrices en sous-parties et en distribuant les calculs, cette approche permet une exécution parallèle, réduisant ainsi le temps de traitement tout en exploitant efficacement les ressources disponibles.

L'application développée est conçue pour être facilement déployée sous forme d'un fichier **.jar**, garantissant une installation simple et rapide sur différentes machines. Ce rapport détaille la conception, l'implémentation et les défis techniques rencontrés, mettant en avant les avantages du calcul distribué pour le traitement de grandes matrices.

II. Résumé :

Ce projet consiste en la création d'une application distribuée en Java pour effectuer des opérations matricielles (addition, soustraction, multiplication) en parallèle. L'application est composée de trois parties principales : un client, un serveur principal, et plusieurs serveurs esclaves.

- **Client** : Le client charge les matrices à partir d'un fichier, envoie une requête au serveur principal via RMI, et écrit le résultat dans un fichier de sortie.
- **Serveur Principal** : Le serveur principal expose les opérations matricielles via RMI, divise les matrices en sous-matrices, et les envoie aux serveurs esclaves pour traitement.
- **Serveurs Esclaves** : Les serveurs esclaves effectuent les opérations matricielles sur les sous-matrices reçues et renvoient les résultats au serveur principal.

L'application utilise des threads pour gérer les communications et les calculs en parallèle, ce qui permet d'améliorer les performances pour les grandes matrices. Pour faciliter son déploiement et son utilisation, l'application a été empaquetée sous forme d'un fichier exécutable .jar, permettant une distribution et une exécution simplifiées sur différentes machines sans nécessiter de configuration complexe.

Le rapport détaille la conception de chaque composant, les choix techniques, et les résultats obtenus, en mettant en avant l'optimisation des performances et la simplicité d'utilisation grâce à l'empaquetage en fichier .jar.

III. Chapitre 1 : Problématiques, objectifs et produit final

1. Problématique

Pour répondre aux besoins de calculs matriciels sur des matrices de grande taille, il est essentiel de résoudre les problèmes suivants :

- **Temps de calcul élevé** : Les approches séquentielles sur une seule machine ne sont pas adaptées pour traiter des matrices de grande taille en un temps raisonnable.
- **Ressources limitées** : Une seule machine ne dispose pas toujours des ressources nécessaires (mémoire, puissance de calcul) pour effectuer des calculs complexes.
- **Gestion de la parallélisation** : Il est nécessaire de diviser les matrices en sous-matrices et de distribuer les calculs sur plusieurs machines pour améliorer les performances.

Ce projet vise à résoudre ces problèmes en proposant une solution distribuée et parallèle pour le calcul matriciel. L'application est fournie sous forme d'un fichier .jar, ce qui simplifie son déploiement et permet une utilisation sur plusieurs machines sans configuration complexe.

2. Objectifs

- Plusieurs serveurs, l'application permet de réduire

Ce projet vise à développer une application distribuée et parallèle pour le calcul matriciel en Java, permettant de résoudre les problèmes suivants :

- **Temps de calcul élevé** : Accélérer les opérations matricielles en divisant les matrices en sous-matrices et en répartissant les calculs sur plusieurs serveurs.
- **Ressources limitées** : Optimiser l'utilisation des ressources en permettant le traitement de matrices volumineuses sur des systèmes distribués.
- **Gestion de la parallélisation** : Améliorer la performance en gérant la parallélisation via des threads et une communication efficace entre les serveurs.

L'application propose une interface simplifiée pour :

- Charger les matrices,
- Spécifier les opérations (addition, soustraction, multiplication),
- Récupérer les résultats.

L'application est empaquetée sous forme de fichier .jar, facilitant son déploiement et son utilisation. Ce projet permettra également d'améliorer les compétences en développement distribué, notamment avec RMI, les sockets et la gestion des threads, tout en offrant une solution réutilisable pour d'autres besoins de calcul distribué.

3. Produit final

- Le produit final est une application distribuée en Java permettant d'effectuer des opérations matricielles en parallèle. Ses principales caractéristiques sont les suivantes :

- **Opérations matricielles** : Addition, soustraction et multiplication de matrices.
- **Communication distribuée** : Utilisation de RMI pour la communication client-serveur et de sockets pour la communication serveur-esclave.
- **Parallélisation** : Utilisation de threads pour répartir les calculs sur plusieurs serveurs esclaves.
- **Interface simple** : Chargement des matrices, spécification de l'opération et récupération des résultats.
- **Empaquetage en fichier .jar** : Simplification du déploiement et de l'utilisation sur diverses machines.

  Cette solution est réutilisable et peut être adaptée à d'autres besoins de calcul distribué.

IV. Chapitre 2 : Conception et modélisation de l'application

1. Introduction

Dans le cycle de vie de notre projet la conception représente une phase primordiale déterminante pour produire une application de haute qualité. C'est dans ce stade que nous devons clarifier en premier lieu l'étude technique réalisée, ou cette étude entame la projection des besoins en langage de modélisation UML à travers les différents diagrammes :Diagramme de séquence ,diagramme de cas d'utilisation et diagramme de classe.

2. Diagramme de cas d'utilisation :

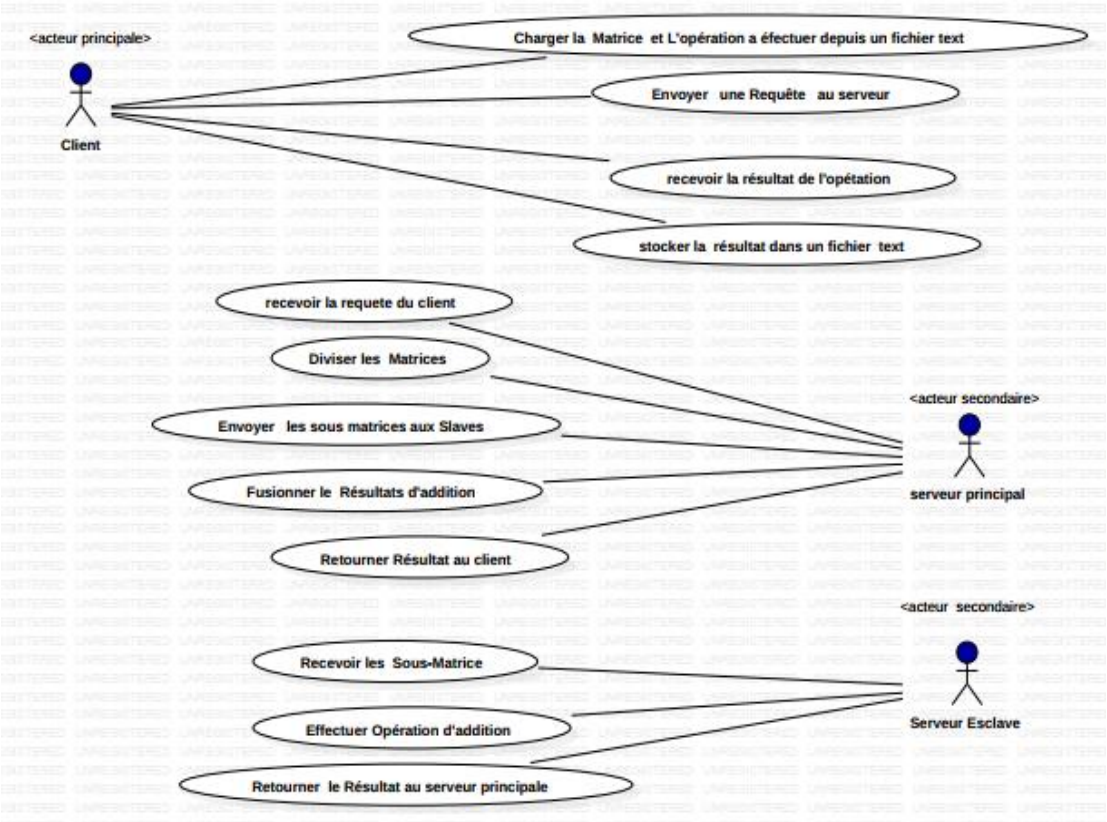

Figure 1:Diagramme de cas d'utilisation

Les Cas d'Utilisation Principaux :

1. **Client**
    - **Charger les données** : Lit les matrices et l'opération depuis un fichier texte.
    - **Envoyer la requête** : Transmet les données au serveur via RMI.
    - **Recevoir le résultat** : Récupère le résultat final du serveur.
    - **Sauvegarder le résultat** : Écrit le résultat dans un fichier texte.
2. **Serveur Principal**
    - **Réception de la requête** : Reçoit les matrices et l'opération via RMI.
    - **Division des matrices** : Décompose les matrices en sous-matrices.
    - **Distribution du calcul** : Envoie les sous-matrices aux serveurs esclaves via sockets.
    - **Fusion des résultats** : Assemble les résultats partiels.
    - **Retour au client** : Envoie le résultat final via RMI.
3. **Serveur Esclave**
    - **Réception des sous-matrices** : Reçoit les données du serveur principal.
    - **Exécution de l'opération** : Effectue l'addition, soustraction ou multiplication.
    - **Retour du résultat** : Envoie le résultat partiel au serveur principal.

3. Diagramme de Classe :

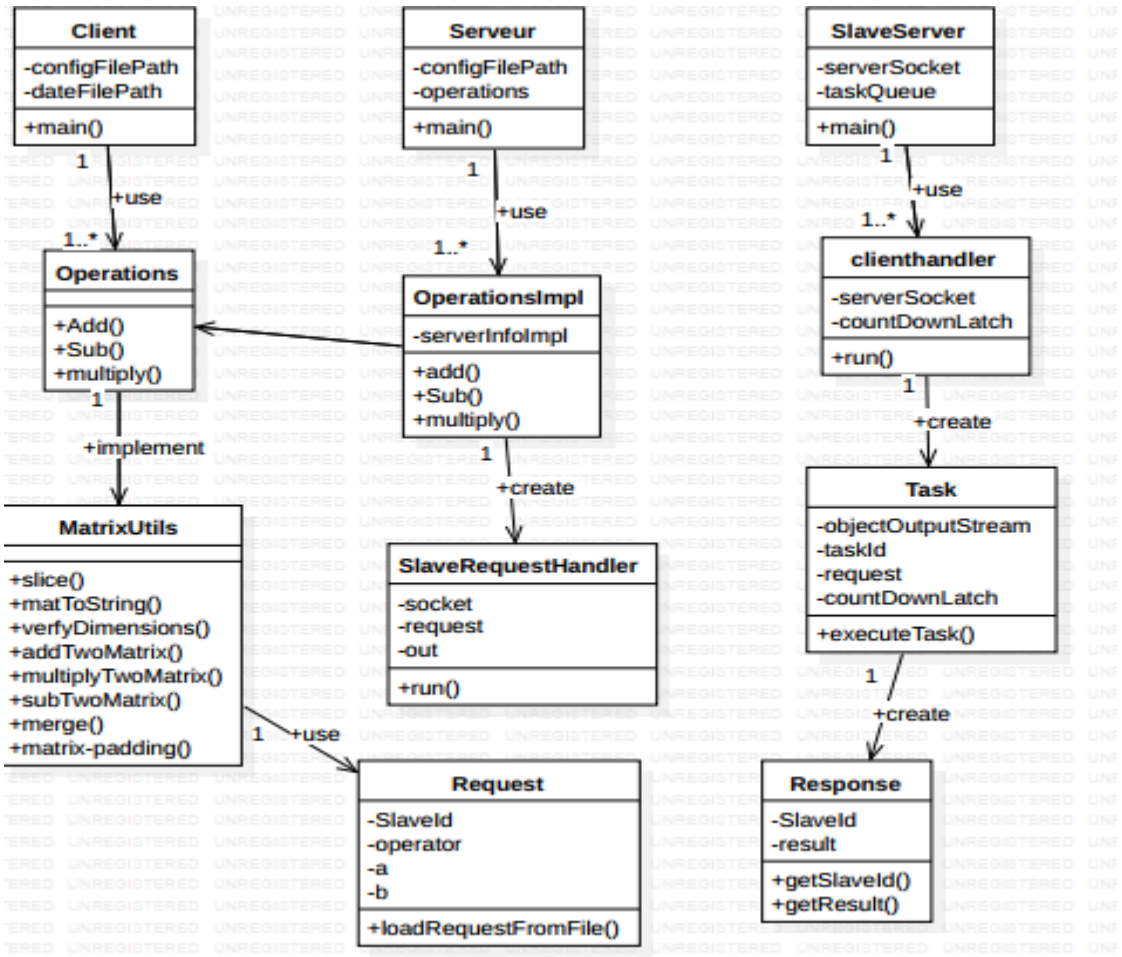

Figure 2:Diagramme de classe

La Description du Diagramme de Classe :

Ce diagramme modélise l'architecture de l'application distribuée pour le calcul matriciel. Il est divisé en trois parties principales : **Client, Serveur Principal et Serveurs Esclaves**.

1. Client

- **Client** : Charge les fichiers de matrices et envoie des requêtes au serveur principal via RMI.

- **Operations** : Interface définissant les opérations matricielles (+, -, *).
- **OperationsImpl** : Implémente les opérations définies dans Operations.

2. Serveur Principal

- **Serveur** : Coordonne le traitement en recevant la requête du client, en divisant la matrice et en distribuant les sous-tâches.
- **MatrixUtils** : Fournit des méthodes utilitaires pour la manipulation des matrices (division, fusion, vérification des dimensions).
- **SlaveRequestHandler** : Gère l'envoi des sous-tâches aux serveurs esclaves via des sockets.

3. Serveurs Esclaves

- **SlaveServer** : Reçoit les sous-tâches du serveur principal et les exécute.
- **ClientHandler** : Gère la communication entre le serveur principal et les esclaves.
- **Task** : Exécute les calculs sur les sous-matrices.
- **Request** : Représente une requête envoyée aux esclaves (matrices et opération à exécuter).
- **Response** : Stocke le résultat du calcul et l'identifiant du serveur esclave.

4. Diagramme de séquence:

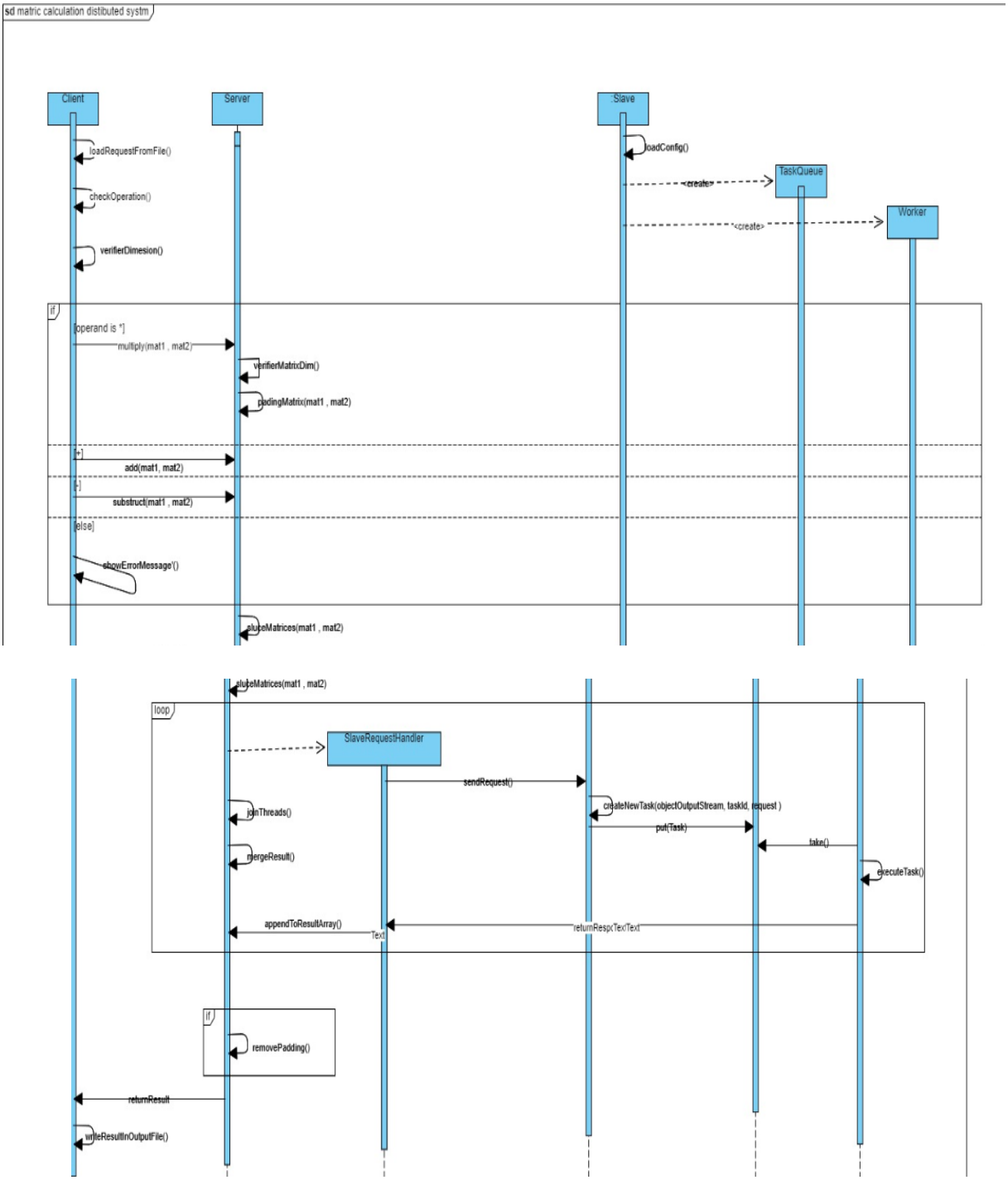

Figure 3:Diagramme de séquence

La descreption du Diagramme de Séquence :

Ce diagramme de séquence illustre le fonctionnement détaillé d'un **système de calcul matriciel distribué** basé sur **Java RMI et sockets**.

1. **Le client** envoie une requête au serveur contenant les matrices et l'opération à effectuer (addition, soustraction ou multiplication).
2. **Le serveur** :
    - Vérifie l'opération demandée.
    - Vérifie la compatibilité des dimensions des matrices.
    - En fonction du type d'opération, il répartit le travail en sous-tâches.
3. **Les serveurs esclaves** :
    - Chargent la configuration et préparent les files de tâches.
    - Créent des objets de tâches et les ajoutent à la file d'attente.
4. **Le gestionnaire de tâches (Task Queue)** :
    - Récupère les sous-tâches et les assigne aux **workers** pour exécution.
5. **Les workers** effectuent les calculs sur les sous-matrices.
6. Une fois les calculs terminés, **les résultats sont renvoyés aux serveurs esclaves**, puis au **serveur principal**.
7. **Le serveur principal** agrège les résultats et reconstruit la matrice finale.
8. **Le serveur renvoie le résultat final** au client.

V. Chapitre 3 : Implémentation

1. Introduction

Dans ce chapitre, nous détaillons la mise en œuvre de l'application distribuée pour le calcul matriciel parallèle. Nous expliquerons les technologies utilisées (RMI, sockets, threads), le rôle de chaque composant, et les interactions entre les classes. L'objectif est de montrer comment l'application fonctionne de manière concrète, en mettant l'accent sur la communication entre les différents modules et la gestion des calculs parallèles.

2. Structure du projet :

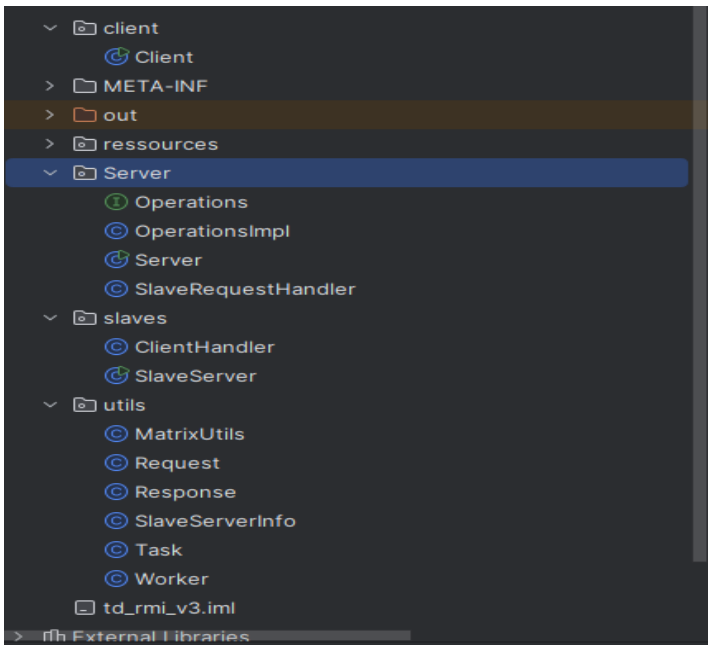

Figure 4:Structure du projet

3. Technologies et Concepts Clés

L'application utilise plusieurs technologies pour la communication et la parallélisation :

- **RMI (Remote Method Invocation)** :

  Permet au client de communiquer avec le serveur principal de manière transparente. Le client invoque les méthodes distantes via un registre RMI pour des opérations matricielles.

  Avantage : Simplifie la communication en masquant la complexité des appels distants.

- **Sockets** :

  Utilisés pour la communication entre le serveur principal et les serveurs esclaves. Le serveur principal envoie des sous-tâches aux serveurs esclaves via des sockets.

  Avantage : Permet une communication bidirectionnelle et asynchrone.

- **Threads** :

  Permettent de gérer les communications et calculs en parallèle. Le serveur principal crée des threads pour envoyer des sous-tâches, et les serveurs esclaves utilisent des threads pour exécuter les tâches.

  Avantage : Optimise les performances en exploitant les ressources distribuées

4. Architecture de l'Application

L'application est composée de trois principaux composants :

- **Client :**

  Chargé de charger les matrices, d'envoyer la requête au serveur principal, et d'afficher les résultats.

Fonctionnement : Charge les matrices, invoque le serveur via RMI et affiche le résultat.

- **Serveur Principal :**

  Coordonne les opérations en divisant les matrices et en envoyant les sous-tâches aux serveurs esclaves via des sockets.

  Fonctionnement : Divise les matrices, distribue les tâches et renvoie les résultats au client.

- **Serveurs Esclaves :**

  Effectuent les calculs sur les sous-matrices et renvoient les résultats au serveur principal.

  Fonctionnement : Écoutent les requêtes, effectuent les opérations et renvoient les résultats.

5. Relations entre les Classes

Les classes de l'application interagissent de la manière suivante :

❖ **Client ↔ Serveur Principal** :

- Le client invoque les méthodes distantes du serveur principal via RMI.
- Le serveur principal renvoie les résultats au client.

❖ **Serveur Principal ↔ Serveurs Esclaves** :

- Le serveur principal envoie des sous-tâches aux serveurs esclaves via des sockets.

- Les serveurs esclaves renvoient les résultats au serveur principal.

❖ **MatrixUtils** :

- Utilisée par toutes les classes pour manipuler les matrices (division, fusion, vérification des dimensions).

1. Tests et Résultats

Des tests ont été réalisés pour valider le bon fonctionnement de l'application. Ces tests vérifient la bonne exécution des opérations matricielles, la communication entre les composants, et la gestion optimale des matrices de grande taille dans un environnement distribué. Les résultats sont obtenus et affichés via l'application empaquetée sous .jar.

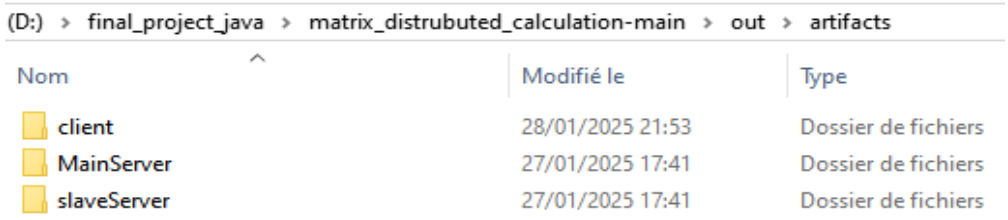

Figure 5:Structure des Artifacts du Projet

Le figure montre la structure des fichiers générés après la compilation et la construction du projet Java distribué. On observe trois dossiers principaux :

- **client/** : Contient les fichiers nécessaires à l'exécution du client.

- **MainServer/** : Regroupe les fichiers du serveur principal qui coordonne les opérations distribuées.
- **slaveServer/** : Stocke les fichiers des serveurs secondaires qui participent au calcul matriciel distribué.

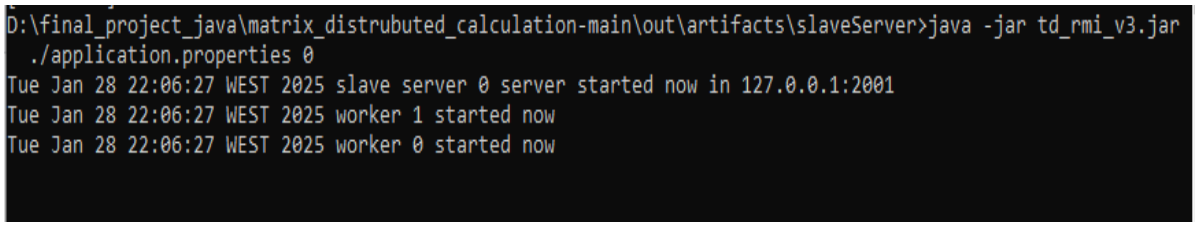

Figure 6:Démarrage du Serveur Secondaire(slave)

Ce figure montre l'exécution du serveur secondaire (**slave server**) dans l'architecture distribuée.

- Le fichier **td_rmi_v3.jar** est exécuté avec Java, en utilisant un fichier de configuration (**application.properties**).
- Le serveur secondaire démarre et écoute sur **127.0.0.1:2001**.
- Deux **workers** sont également lancés, marquant le début du traitement distribué.

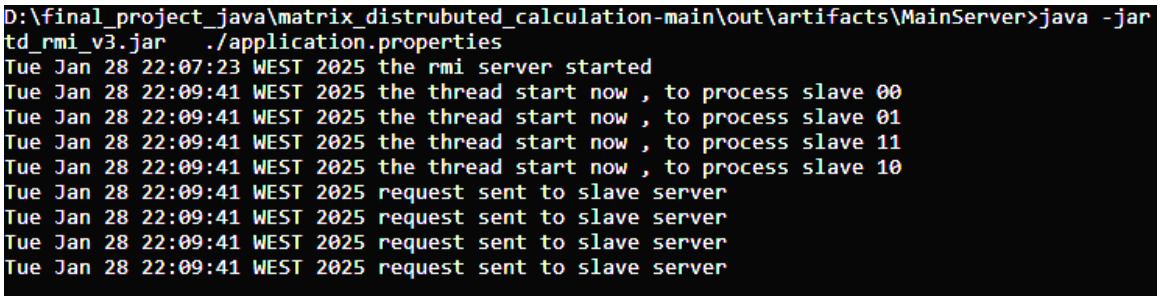

Figure 7:Démarrage du Serveur Principal

Ce figure illustre l'exécution du **serveur principal (MainServer)** dans le système de calcul distribué.

- Le **serveur RMI** démarre avec le fichier **td_rmi_v3.jar** et le fichier de configuration **application.properties**.
- Plusieurs threads sont lancés pour gérer différentes instances des serveurs secondaires (**slaves**), identifiés par des indices (00, 01, 10, 11).
- Le serveur principal envoie ensuite plusieurs **requêtes aux serveurs secondaires**, initiant ainsi le calcul distribué.

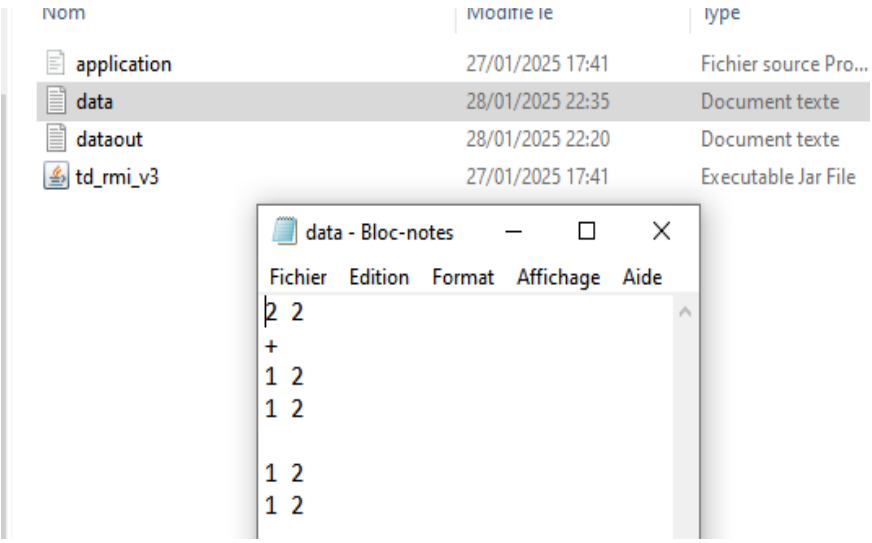

Figure 8:Fichier d'Entrée du Client

Cette figure illustre le fichier data.txt, utilisé comme entrée par le client dans le système de calcul distribué.

- Il contient une opération de somme (+) entre deux matrices de dimensions 2x2.
- Chaque matrice est définie par des lignes contenant des valeurs numériques.
- Ce fichier est traité par le client pour effectuer le calcul et générer un fichier de sortie.

```
D:\final_project_java\matrix_distrubuted_calculation-main\out\artifacts\client>java -jar td_rmi_v3.jar
./data.txt ./application.properties  ./dataout.txt
the operation was successful check the output file

D:\final_project_java\matrix_distrubuted_calculation-main\out\artifacts\client>
```

Figure 9:Exécution du Client

Cette figure montre l'exécution du client dans le système de calcul distribué.

- Le client exécute le fichier td_rmi_v3.jar en spécifiant les fichiers data.txt, application.properties et dataout.txt.
- Une fois le traitement terminé, un message de succès est affiché, indiquant que le résultat est stocké dans le fichier de sortie.

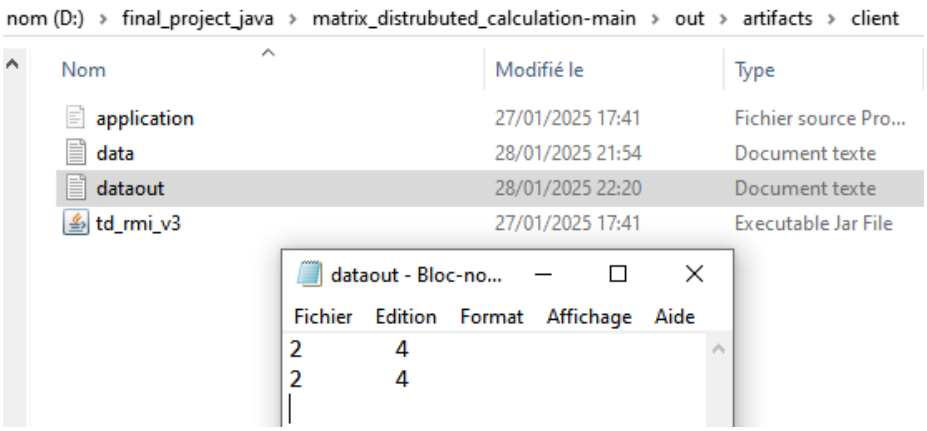

Figure 10:Fichier de Sortie du Client

Cette figure montre le fichier dataout.txt, qui contient le résultat du calcul distribué.

- Les valeurs résultantes correspondent à la somme des matrices d'entrée.
- Ce fichier confirme que l'opération a été effectuée correctement par le système.

VI. Conclusion

Ce projet a permis de concevoir et d'implémenter une application distribuée en Java pour effectuer des opérations matricielles en parallèle. Grâce à l'utilisation de RMI pour la communication client-serveur et de sockets pour la coordination entre le serveur principal et les serveurs esclaves, l'application optimise le temps de calcul et exploite efficacement les ressources disponibles sur plusieurs machines.

Les tests effectués ont démontré l'efficacité du système dans le traitement de matrices de grande taille, en réduisant considérablement le temps de calcul par rapport à une exécution séquentielle. L'empaquetage en fichier **.jar** facilite son déploiement et son utilisation, rendant l'application portable et accessible sans configuration complexe.

En plus des aspects techniques, ce projet nous a permis d'approfondir nos compétences en développement distribué, en gestion des threads, et en optimisation des performances pour des applications intensives en calcul. Cette solution pourrait être étendue à d'autres domaines nécessitant des calculs parallèles, ouvrant ainsi des perspectives pour des améliorations et des évolutions futures.