Université Mohammed V – Rabat
Ecole Mohammadia d'Ingénieurs
Département Génie Informatique

Projet

Sujet

Système de calcul distribué

Réalisé par :
BENHLIMA Douae

Encadré par :
Mme. Dalila Chiadmi

ANNÉE UNIVERSITAIRE 2022-2023

Table des matières

1 Introduction

Dans le contexte actuel d'évolution rapide de la technologie et d'augmentation constante de la quantité de données à traiter, la nécessité de mettre en œuvre des systèmes de calcul distribué efficaces devient de plus en plus évidente. Ces systèmes sont capables de décomposer une tâche complexe en sous-tâches plus gérables qui peuvent être exécutées simultanément sur un ensemble de nœuds, ce qui permet de réduire considérablement le temps nécessaire pour accomplir la tâche.

C'est dans ce contexte que j'ai été chargé de concevoir et de développer un système de calcul distribué pour l'exécution parallèle de tâches spécifiques, en particulier la multiplication de matrices. Le système que j'ai développé est capable de distribuer les tâches entre les nœuds de calcul et de collecter les résultats. Il fournit également une interface utilisateur simple et intuitive par laquelle les utilisateurs peuvent soumettre leurs tâches.

Le but de ce rapport est de présenter l'architecture de ce système, les choix de conception que j'ai faits lors de sa création, ainsi que les motivations qui ont guidé ces choix.

Pour ce projet, j'ai utilisé le langage de programmation Java, en raison de ses nombreuses bibliothèques intégrées pour la programmation concurrente, de son support de la programmation orientée objet qui facilite la modélisation des composants de mon système, et de sa portabilité, ce qui permet à mon système de fonctionner sur une variété de plateformes.

2 Présentation de l'architecture du système

Le système que j'ai développé est composé de plusieurs parties qui interagissent entre elles pour accomplir la tâche de la multiplication des matrices de manière distribuée. Il est basé sur une architecture client/serveur, avec un serveur central qui coordonne la distribution des tâches et la collecte des résultats, et plusieurs nœuds qui exécutent les tâches assignées.

CentralServer : Le serveur central est l'entité qui coordonne la distribution des tâches entre les différents nœuds. Il crée un nombre de nœuds équivalent à la dimension de la matrice en utilisant un ExecutorService. Lorsqu'une demande de multiplication de matrices est reçue, il répartit les lignes de la première matrice entre les différents nœuds, soumet ces tâches à l'ExecutorService et attend que les résultats soient renvoyés.

Node et NodeImpl : La tâche de calcul est encapsulée dans les objets Node. Chaque Node est capable d'exécuter la multiplication d'une ligne de la première matrice avec la seconde matrice. L'implémentation de Node (NodeImpl) est où le calcul réel est effectué.

MatrixInput : Il s'agit de la classe qui gère l'interface utilisateur. Elle crée une fenêtre où l'utilisateur peut entrer les matrices à multiplier, initier le calcul, et voir le résultat. Les matrices sont entrées comme des champs de texte organisés en grille, la taille de la grille étant définie par l'utilisateur. Une fois les matrices entrées, l'utilisateur peut demander la multiplication, qui est alors effectuée par le serveur central.

En résumé, l'utilisateur interagit avec le système à travers l'interface MatrixInput. Lorsqu'il demande une multiplication, les matrices sont envoyées au serveur central qui les distribue entre les nœuds. Chaque nœud effectue sa partie du calcul et renvoie les résultats au serveur central, qui les assemble et les renvoie à l'interface utilisateur pour affichage.

3 Détails de l'architecture et choix de conception

3.1 CentralServer

Le serveur central, implémenté dans la classe CentralServer, est le composant central du système qui gère la coordination des tâches entre les différents nœuds. Le nombre de nœuds est équivalent à la dimension de la matrice à multiplier. J'ai utilisé la classe ExecutorService de Java pour gérer un pool de threads, chaque thread exécutant une tâche de calcul sur un nœud. Voici un extrait de code qui montre comment cela est fait :

```
public CentralServer(int numNodes) {
    executor = Executors.newFixedThreadPool(numNodes);
    nodes = new ArrayList<>();
    for (int i = 0; i < numNodes; i++) {
        nodes.add(new NodeImpl());
    }
}
```

La fonction multiplyMatrices est la fonction principale qui est appelée pour multiplier deux matrices. Cette fonction crée des tâches calculables qui sont ensuite soumises à executor pour être exécutées parallèlement.

```java
1 usage
public int[][] multiplyMatrices(int[][] matrix1, int[][] matrix2) throws InterruptedException, ExecutionException {
    int[][] result = new int[matrix1.length][matrix2[0].length];

    // Create a list to hold the Future object associated with Callable
    List<Future<int[]>> futures = new ArrayList<>();

    for (int i = 0; i < matrix1.length; i++) {
        Node node = nodes.get(i % nodes.size());
        int rowIndex = i;
        Callable<int[]> callable = () -> node.calculate(rowIndex, matrix1, matrix2);
        // Submit Callable tasks to be executed by thread pool
        Future<int[]> future = executor.submit(callable);
        // Add Future to the list, we can get return value using Future
        futures.add(future);
    }

    for (int i = 0; i < matrix1.length; i++) {
        result[i] = futures.get(i).get();
    }

    return result;
}
```

3.2 NodeImpl

Le nœud, implémenté dans la classe NodeImpl, est l'entité qui exécute la tâche de calcul. Chaque nœud est capable de calculer une ligne du produit de deux matrices. Voici un extrait de code qui montre comment cela est fait :

```java
public class NodeImpl implements Node{
    1 usage
    @Override
    public int[] calculate(int rowIndex, int[][] matrix1, int[][] matrix2) throws InterruptedException {
        int[] result = new int[matrix2[0].length];
        for (int i = 0; i < matrix2[0].length; i++) {
            for (int j = 0; j < matrix1[rowIndex].length; j++) {
                result[i] += matrix1[rowIndex][j] * matrix2[j][i];
            }
        }
        return result;
    }
}
```

3.3 MatrixInput

La classe MatrixInput est l'interface utilisateur du système. Elle permet aux utilisateurs de définir la taille de la matrice, d'entrer les valeurs de la matrice et d'initier le calcul. L'interface est construite en utilisant le JFrame de Java Swing, avec des champs de texte pour entrer les valeurs de la matrice.

Lorsque l'utilisateur appuie sur le bouton "Calculer la multiplication", la fonction calculateMultiplication est appelée. Cette fonction récupère les valeurs de la matrice à partir des champs de texte, les convertit en tableaux 2D d'entiers, puis les envoie au serveur central pour le calcul.

```
private void calculateMultiplication() {
    System.out.println("calculateMultiplication: start");

    int[][] matrix1 = getMatrixFromFields(matrix1Fields);
    int[][] matrix2 = getMatrixFromFields(matrix2Fields);

    try {
        int[][] result = server.multiplyMatrices(matrix1, matrix2);
        System.out.println("calculateMultiplication: result = " + Arrays.deepToString(result));

        StringBuilder resultString = new StringBuilder();
        for (int i = 0; i < result.length; i++) {
            for (int j = 0; j < result[i].length; j++) {
                resultString.append(result[i][j]).append("\t");
            }
            resultString.append("\n");
        }
        resultArea.setText(resultString.toString());
    } catch (InterruptedException | ExecutionException e) {
        System.out.println("calculateMultiplication: exception = " + e.getMessage());
        resultArea.setText("Erreur lors du calcul de la multiplication: " + e.getMessage());
    }

    System.out.println("calculateMultiplication: end");
}
```

4 Mécanisme de distribution des tâches

Dans le système que j'ai conçu, la distribution des tâches se fait de manière parallèle grâce à l'utilisation d'un ExecutorService. C'est un mécanisme de gestion de thread pool fourni par Java qui permet de répartir efficacement les tâches sur plusieurs threads. L'ExecutorService est initialisé avec un nombre fixe de threads, correspondant à la dimension de la matrice dans ce cas.

```
public CentralServer(int numNodes) {
    executor = Executors.newFixedThreadPool(numNodes);
    nodes = new ArrayList<>();
    for (int i = 0; i < numNodes; i++) {
        nodes.add(new NodeImpl());
    }
}
```

Les tâches sont ensuite soumises à l'ExecutorService sous forme de Callables, qui sont des tâches qui renvoient un résultat et qui peuvent lever une exception. Chaque Callable est une tâche de multiplication d'une ligne de la première matrice par la seconde matrice, exécutée par un Node.

```
Callable<int[]> callable = () -> node.calculate(rowIndex, matrix1, matrix2);
// Submit Callable tasks to be executed by thread pool
Future<int[]> future = executor.submit(callable);
// Add Future to the list, we can get return value using Future
futures.add(future);
```

La classe CentralServer est responsable de ce mécanisme. Elle crée et gère l'ExecutorService, qui est utilisé pour répartir les tâches de calcul entre les différents nœuds (instances de NodeImpl).

5 Mécanisme de communication

La communication entre les différentes parties du système est gérée de manière implicite grâce à l'utilisation de l'ExecutorService et de la classe Future de Java. Les futures sont utilisés pour récupérer le résultat des tâches exécutées de manière asynchrone. Cela permet de ne pas bloquer l'exécution du programme en attendant que toutes les tâches soient terminées, ce qui est crucial pour un système distribué où les tâches peuvent prendre un certain temps à s'exécuter.

```
for (int i = 0; i < matrix1.length; i++) {
    result[i] = futures.get(i).get();
}
```

La communication est gérée implicitement par l'ExecutorService, et les Futures sont utilisés pour récupérer les résultats des tâches de manière asynchrone. Encore une fois, c'est la classe CentralServer qui coordonne ces opérations, en soumettant les tâches à l'ExecutorService et en récupérant les résultats via les Futures.

6 Collecte des résultats intermédiaires, intégration et retour à l'utilisateur

La collecte des résultats intermédiaires est effectuée à l'aide de la méthode get() de la classe Future. Cette méthode bloque jusqu'à ce que le résultat de la tâche soumise soit disponible, puis renvoie ce résultat. Cela garantit que tous les résultats intermédiaires sont collectés avant de procéder à l'étape suivante.

Ensuite, ces résultats intermédiaires sont intégrés dans une matrice finale qui est renvoyée à l'utilisateur. Cette matrice est construite ligne par ligne à partir des résultats intermédiaires. Enfin, cette matrice de résultats est affichée à l'utilisateur via l'interface utilisateur, en utilisant la méthode setText() de la classe JTextArea.

```
StringBuilder resultString = new StringBuilder();
for (int i = 0; i < result.length; i++) {
    for (int j = 0; j < result[i].length; j++) {
        resultString.append(result[i][j]).append("\t");
    }
    resultString.append("\n");
}
resultArea.setText(resultString.toString());
```

La collecte des résultats intermédiaires et leur intégration dans une matrice finale est également gérée par la classe CentralServer. Le retour des résultats à l'utilisateur est effectué par l'interface utilisateur, qui est gérée par la classe MatrixInput. Une fois que CentralServer a assemblé la matrice de résultats, celle-ci est transmise à MatrixInput, qui l'affiche à l'utilisateur via un JTextArea.

7 Choix de Conception motivation et argumentation

Voici une représentation détaillée des choix de conception que j'ai utilisés pour développer ce système, ainsi que la motivation et l'argumentation pour chacun.

- **Choix 1: Utilisation de Java Swing pour l'interface utilisateur**

Motivation et argumentation: Le choix d'utiliser Java Swing pour l'interface utilisateur était basé sur l'objectif d'offrir une expérience utilisateur interactive et conviviale. Swing est largement utilisé et dispose d'une large gamme de composants d'interface utilisateur qui ont permis de créer une interface intuitive pour entrer les données de la matrice. De plus, l'abondance de documentation et de ressources communautaires pour Swing facilite l'implémentation et la résolution des problèmes.

- **Choix 2: Architecture distribuée basée sur le client/serveur**

Motivation et argumentation: L'architecture distribuée basée sur le client/serveur a été choisie en raison de sa capacité à gérer efficacement l'exécution parallèle de tâches. Dans ce cas, chaque nœud client est responsable de calculer une ligne du produit de deux matrices, permettant ainsi une distribution efficace du travail. Cela optimise l'utilisation des ressources et augmente la performance globale du système, en particulier pour les grandes matrices.

- **Choix 3: Utilisation de l'ExecutorService de Java**

Motivation et argumentation: L'ExecutorService de Java a été utilisé pour gérer un pool de threads, permettant ainsi une exécution simultanée des tâches. Ce choix était motivé par la nécessité d'améliorer l'efficacité de l'opération de multiplication des matrices. En permettant l'exécution concurrente des tâches, l'ExecutorService accélère le processus de calcul, en particulier pour les grandes matrices.

- **Choix 4: Conception modulaire**

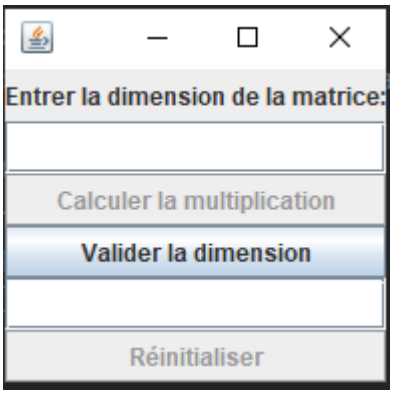
Motivation et argumentation: La conception modulaire a été choisie pour faciliter la maintenance et l'évolution du système. En séparant la logique de l'application en différents modules (l'interface utilisateur, le serveur central, les nœuds clients), chaque partie du système peut être développée, testée, modifiée ou étendue indépendamment des autres. Cela augmente la lisibilité et la flexibilité du code, rendant le système plus facile à maintenir et à étendre à l'avenir.

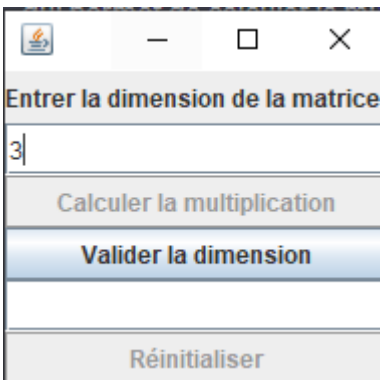
- **Choix 5: Gestion des erreurs robuste**

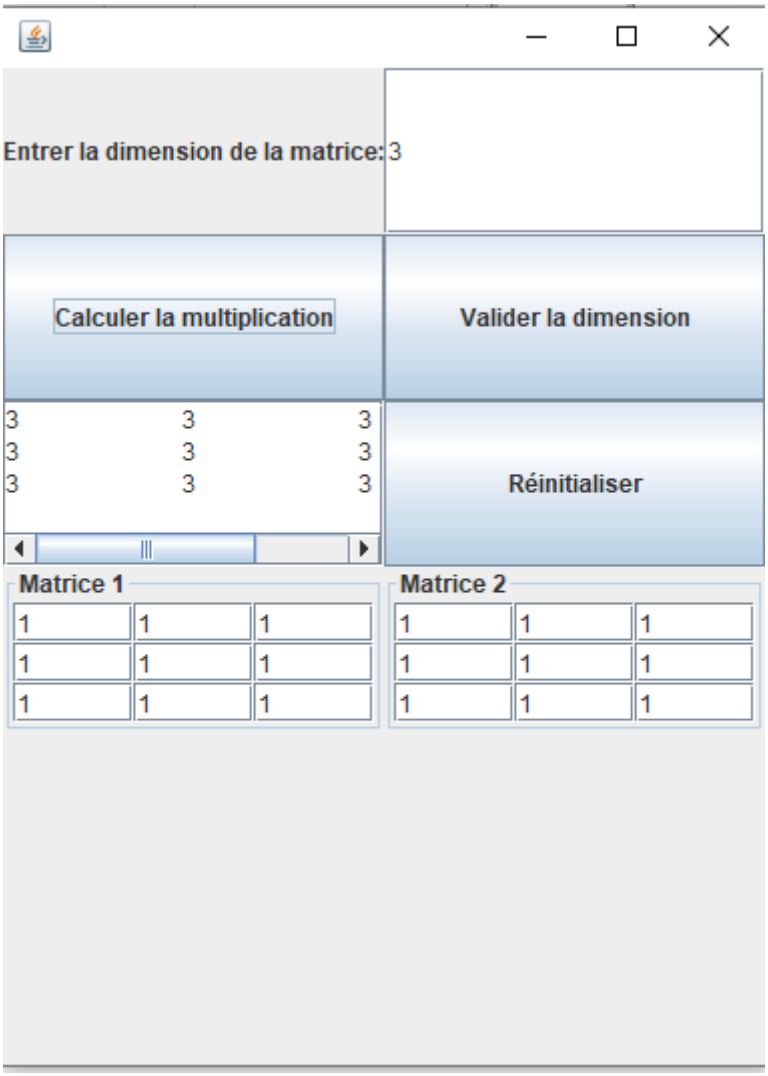
Motivation et argumentation: Une gestion robuste des erreurs a été intégrée dans le système pour assurer sa fiabilité. Cela a été motivé par la nécessité d'assurer que le système peut gérer et signaler les erreurs de manière appropriée lors de l'exécution des tâches. Par exemple, si une exception est levée lors du calcul, l'erreur est capturée et un message approprié est affiché à l'utilisateur. Cette approche renforce la robustesse du système et assure une meilleure expérience utilisateur.

8 Démonstration

L'application démarre et présente une interface utilisateur qui permet de déterminer la dimension de la matrice et de calculer la multiplication de deux matrices carrées saisie par l'utilisateur.







Si l'utilisateur souhaite effectuer une nouvelle opération de multiplication, il peut cliquer sur le bouton "Réinitialiser" pour vider tous les champs de texte et réinitialiser la dimension de la matrice. Il peut ensuite recommencer le processus.