Rapport Projet Raytracing Distribué

Noms *: Maxence Eva*

Ryan Korban

Baptise Delaborde

Baptise Henequin

GitHub : https://github.com/korban2u/DistRayTracer

Vidéo Présentation :
https://drive.google.com/file/d/1OJbf31njLzKKWrKGO7gqhcZ1dAbeOA7V/view?usp=sharing

Table des matières

Réponses aux Questions

1. ***<u>En utilisant votre meilleur outil, votre imagination, décrivez et illustrez comment cela pourrait être réalisé, sans rentrer dans les détails JAVA, que vous n'allez pas tarder à mettre en œuvre</u>.***

Nous allons découper l'image en plusieurs morceaux et répartir sur plusieurs machines le calcul des morceaux :

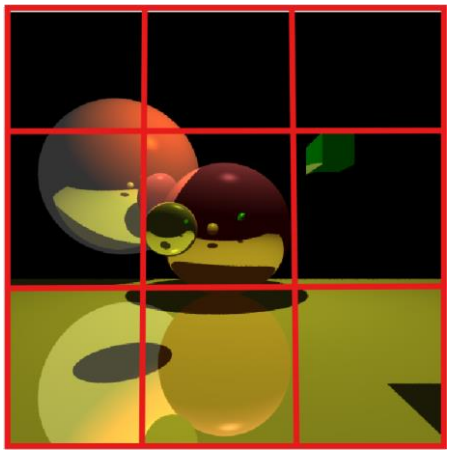

Figure 1 Exemple de découpage de l'image

On pourra aussi utiliser les Threads pour accélérer encore plus le calcul avec un traitement en simultané.

- Il faut donc un premier processus (p1) qui découpera l'image, enverra les parties aux machines qui calcul et puis affichera l'image avec les résultats.
- Un autre processus (p2) qui lui se chargera de gérer les différentes machines de calcul et de les communiquer à p1.
- Et enfin un dernier processus (p3) qui représentera les machines qui calculeront les morceaux envoyés par P1 et renverront le résultat.

2. ***Voici un graphique comparatif du temps de calcul (en ms) de l'image avec le Raytracer :***

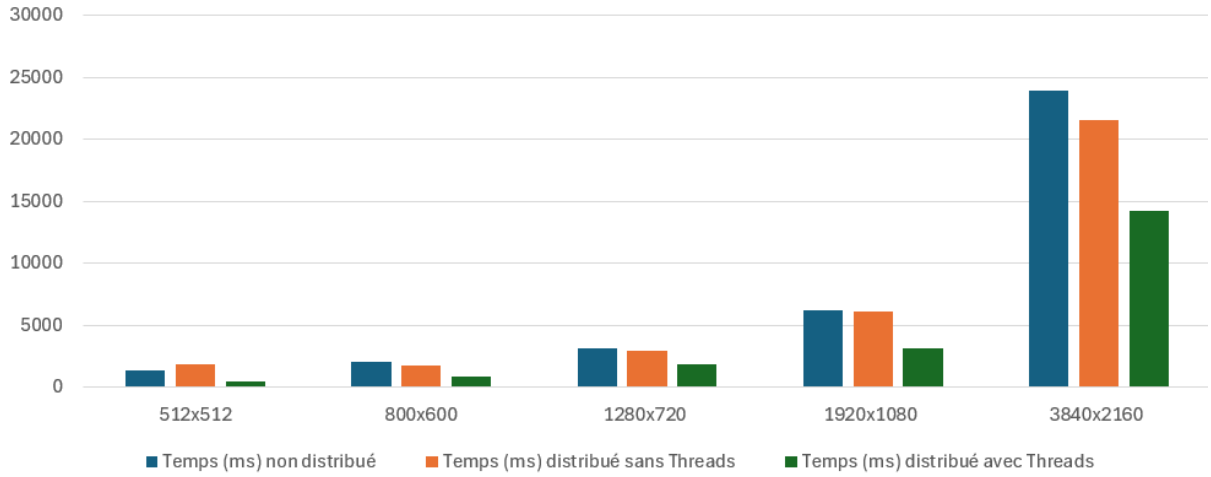

Figure 2 Histogramme groupé du temps de calcul (en ms) de l'image par rapport à la résolution (2 machines utilisé pour les calculs distribué)

3. ***Faire un petit schéma de cette architecture***

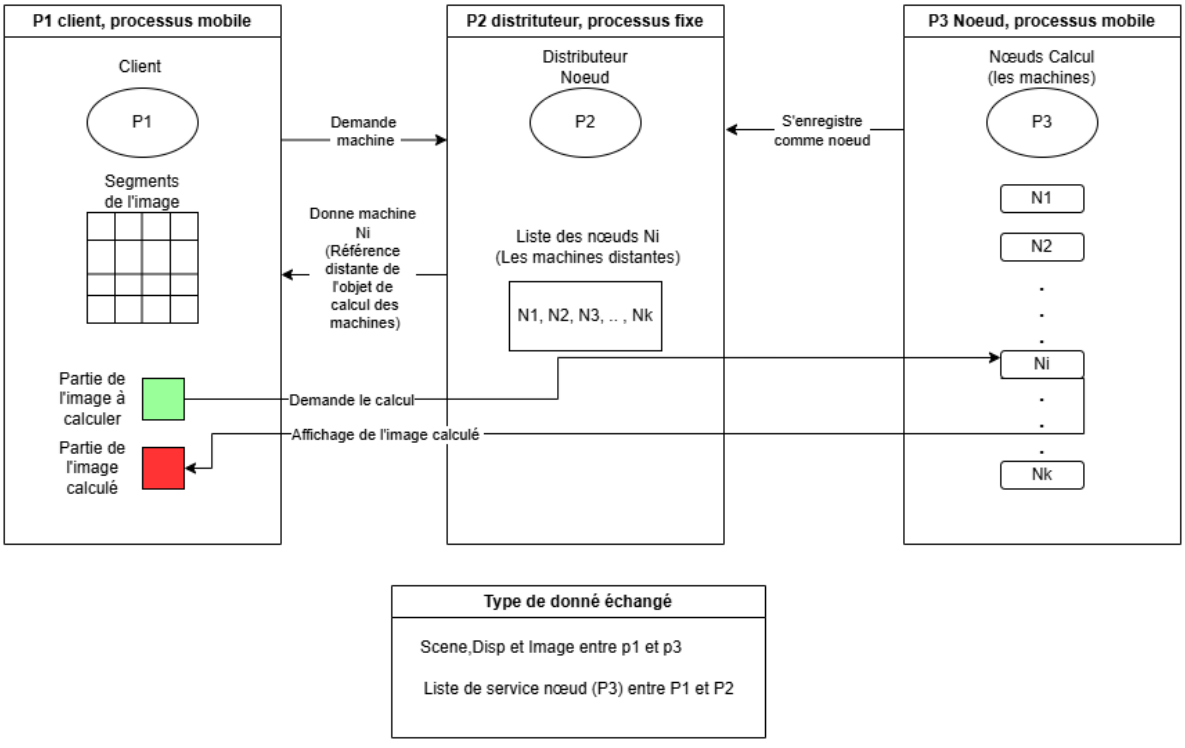

Figure 3 Premier schéma de l'architecture du projet

4. ***<u>Si on veut que les calculs se fassent en parallèle que faut-il faire ?</u>***

Pour que les calculs se fassent en parallèle il faut utiliser les Threads, comme ça chaque demande sera lancer sur un thread séparer permettant ainsi de calculer tous les morceaux d'image en même temps, évitant ainsi d'attendre qu'un calcul soit fini pour en commencer un autre.

Architecture complète et les composants

1. ***<u>Donnez une description de l'architecture complète (schéma) de votre application.</u>***

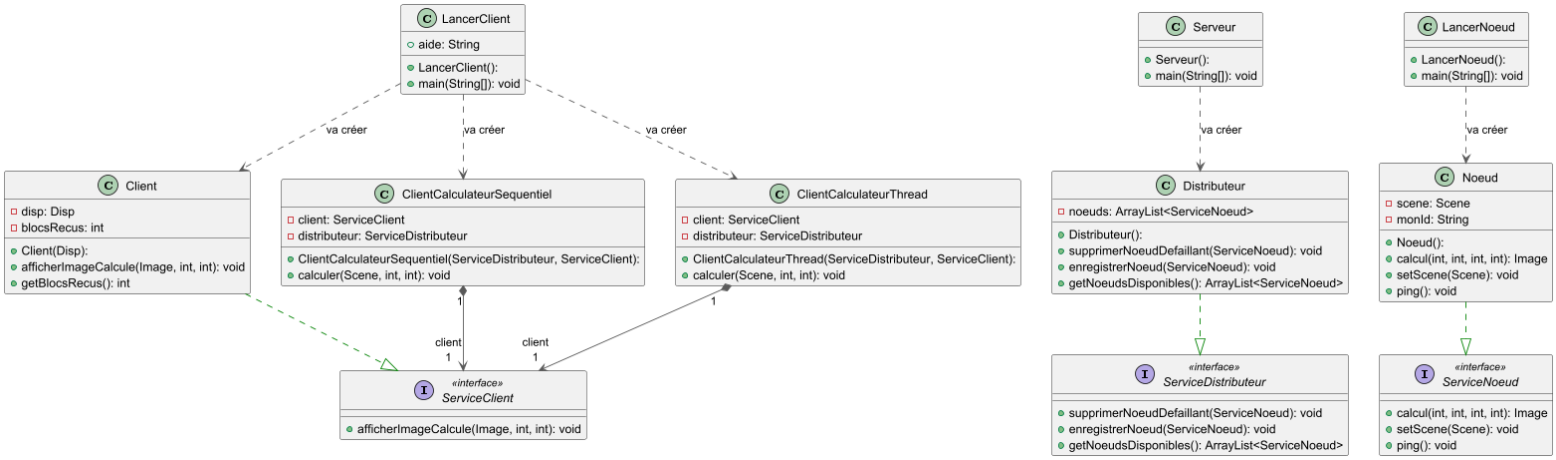

Figure 4 Digramme de classes du projet

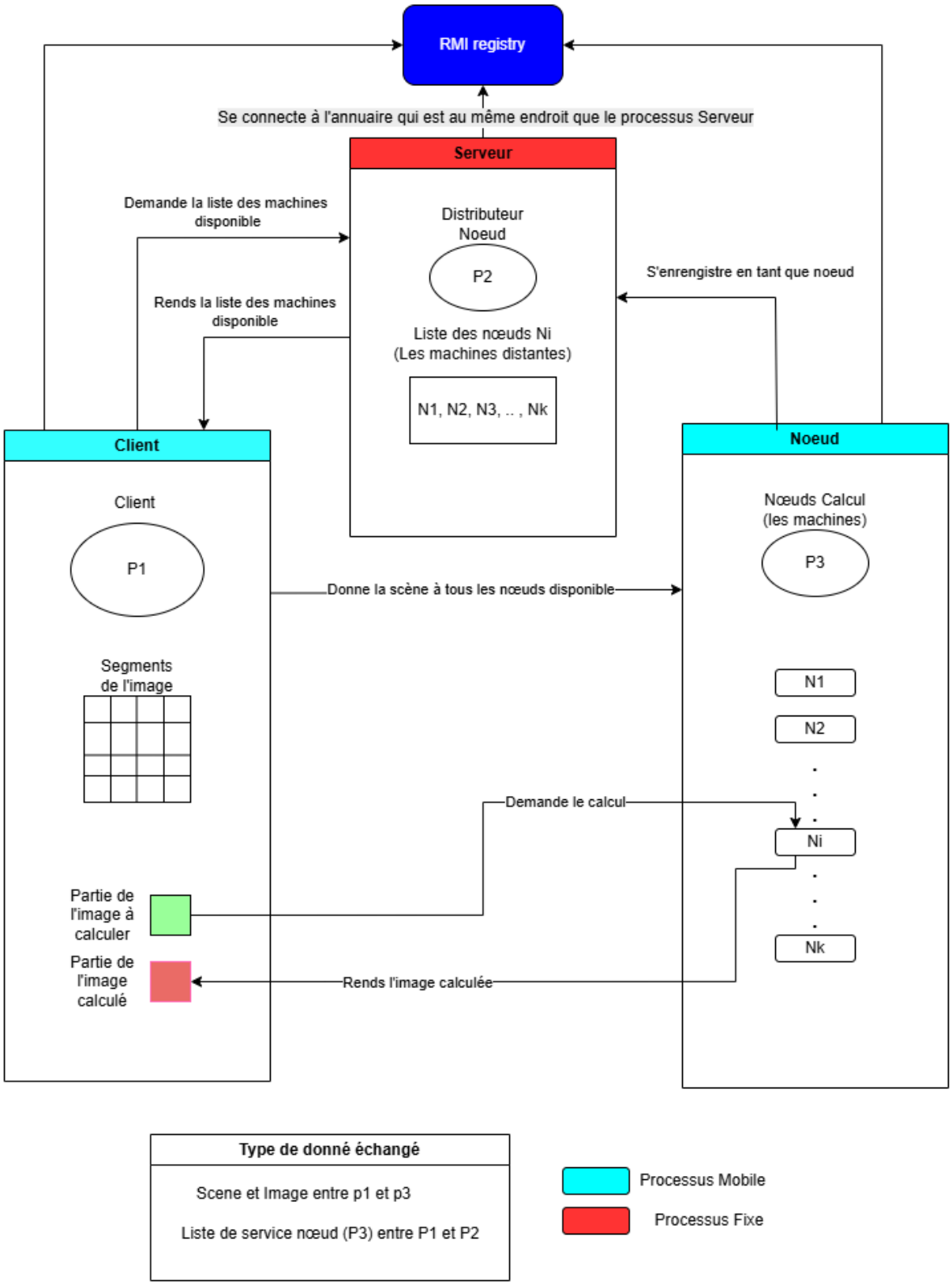

Figure 5 Architecture final du projet

2. ***Donnez les rôles de tous ses composants***

Serveur (Distributeur)

- Gère l'enregistrement des nœuds de calcul
- Maintient la liste des nœuds disponibles
- Détecte et supprime les nœuds déconnectés
- Utilise le registre RMI pour la communication

Nœud (Machine de calcul)

- Effectue les calculs de raytracing pour des blocs d'image
- S'enregistre automatiquement auprès du distributeur
- Reçoit la scène à calculer du client
- Peut être lancé sur plusieurs machines

Client

- Interface utilisateur pour lancer le calcul
- Découpe l'image en blocs et les distribue
- Reçoit et affiche les résultats en temps réel
- Propose deux modes : séquentiel et parallèle (threads)

RMI Registry

- Permet de trouver les services enregistrés

Diagramme de séquences

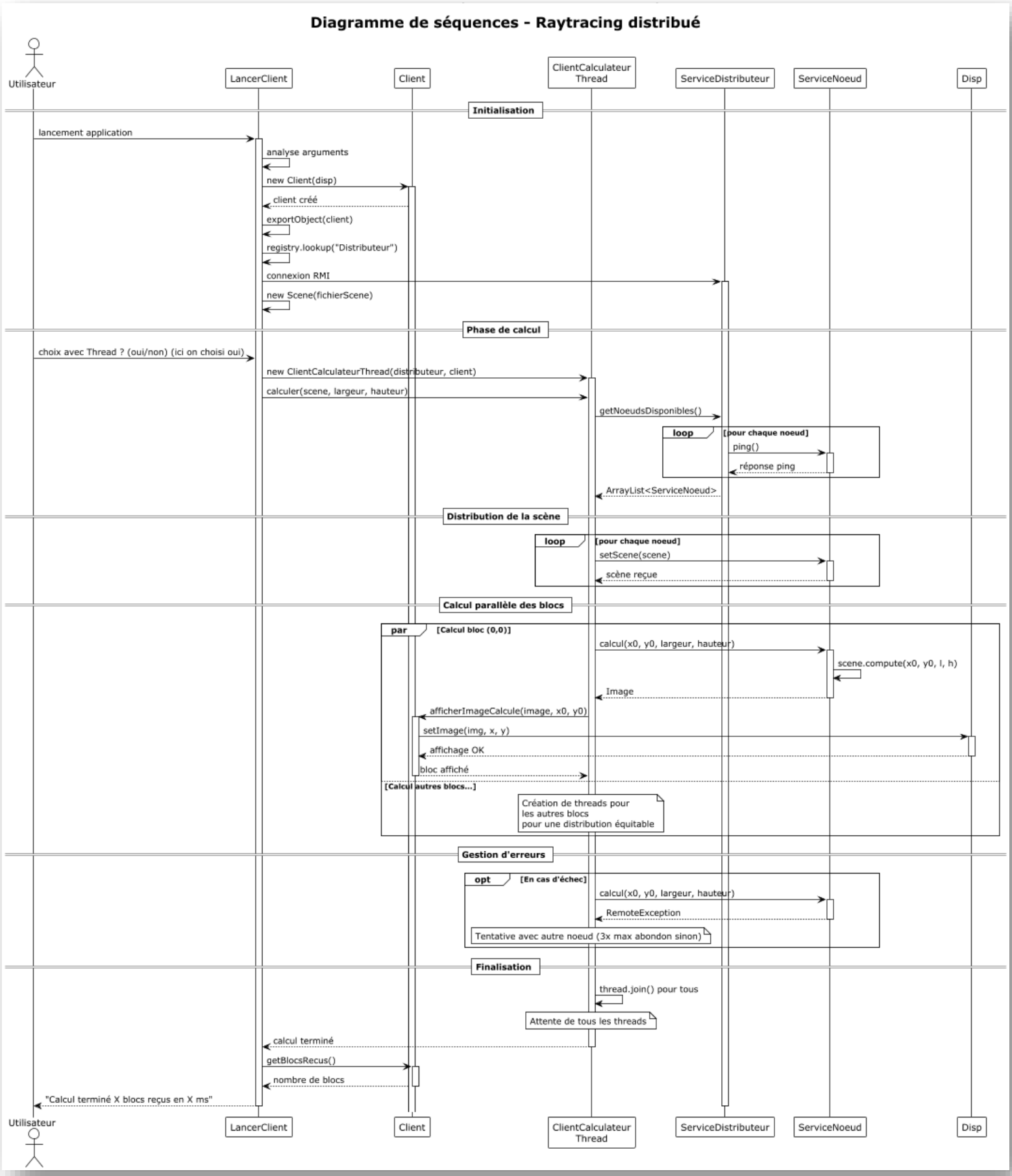

Figure 6 Digramme de séquence

Code des interfaces utilisées

```java
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.ArrayList;

/**
 * Interface Distributeur.
 */
public interface ServiceDistributeur extends Remote {

    /**
     * Enregistre un nouveau noeud chez le distributeur (serveur).
     *
     * @param noeud le noeud à enregistrer
     * @throws RemoteException si une erreur de communication RMI se produit
     */
    void enregistrerNoeud(ServiceNoeud noeud) throws RemoteException;

    /**
     * Récupère la liste des noeuds actuellement disponibles et supprime les noeuds déconnectés
     *
     * @return une nouvelle liste contenant tous les noeuds disponibles
     * @throws RemoteException si une erreur de communication RMI se produit
     */
    ArrayList<ServiceNoeud> getNoeudsDisponibles() throws RemoteException;

    /**
     * Supprime un noeud qui marche pas de la liste des noeuds disponibles.
     *
     * @param noeud le noeud à supprimer
     * @throws RemoteException si une erreur de communication RMI se produit
     */
    void supprimerNoeudDefaillant(ServiceNoeud noeud) throws RemoteException;
}
```

```java
import java.rmi.Remote;
import java.rmi.RemoteException;

import raytracer.Image;

/**
 * Interface du services client
 * Permet aux noeuds (les machines) d'envoyer leurs résultats au client.
 */
public interface ServiceClient extends Remote {

    /**
     * Reçoit et affiche une partie d'image calculée par un noeud distant (machines).
     *
     * @param img l'image calculée à afficher
     * @param x   coordonnée x de l'image dans l'interface grafique
     * @param y   coordonnée y (meme chose)
     * @throws RemoteException si une erreur de communication RMI se produit genre une deconnection
     */
    void afficherImageCalcule(Image img, int x, int y) throws RemoteException;
}
```

```java
import java.rmi.Remote;
import java.rmi.RemoteException;

import raytracer.Image;
import raytracer.Scene;

/**
 * Interface pour le noeud
 */ public interface ServiceNoeud extends Remote {
    /**
     * Définit la scène que ce noeud utilisera pour ses calculs
     *
     * @param scene la scène
     * @throws RemoteException si une erreur de communication RMI se produit
     */
    void setScene(Scene scene) throws RemoteException;


   /**
     * Fait le calcul de raytracing pour un petit bloc de l'image entière.
     *
     * @param x0 coordonnée X du coin supérieur gauche de la zone à calculer
     * @param y0 coordonnée Y du coin supérieur gauche de la zone à calculer
     * @param largeur largeur de la zone à calculer
     * @param hauteur hauteur de la zone à calculer
     * @return l'image calculée pour la zone spécifiée
     * @throws RemoteException si une erreur de communication RMI se produit
     * @throws IllegalStateException si aucune scène n'a été donnée au noeud
     */
    Image calcul(int x0, int y0, int largeur, int hauteur) throws RemoteException;

    /**
     * Méthode qui pemet de vérifier que le noeud est toujours disponible
     *
     * @throws RemoteException si une erreur de communication RMI se produit
     */
    void ping() throws RemoteException; // Pour tester la connectivité
}
```