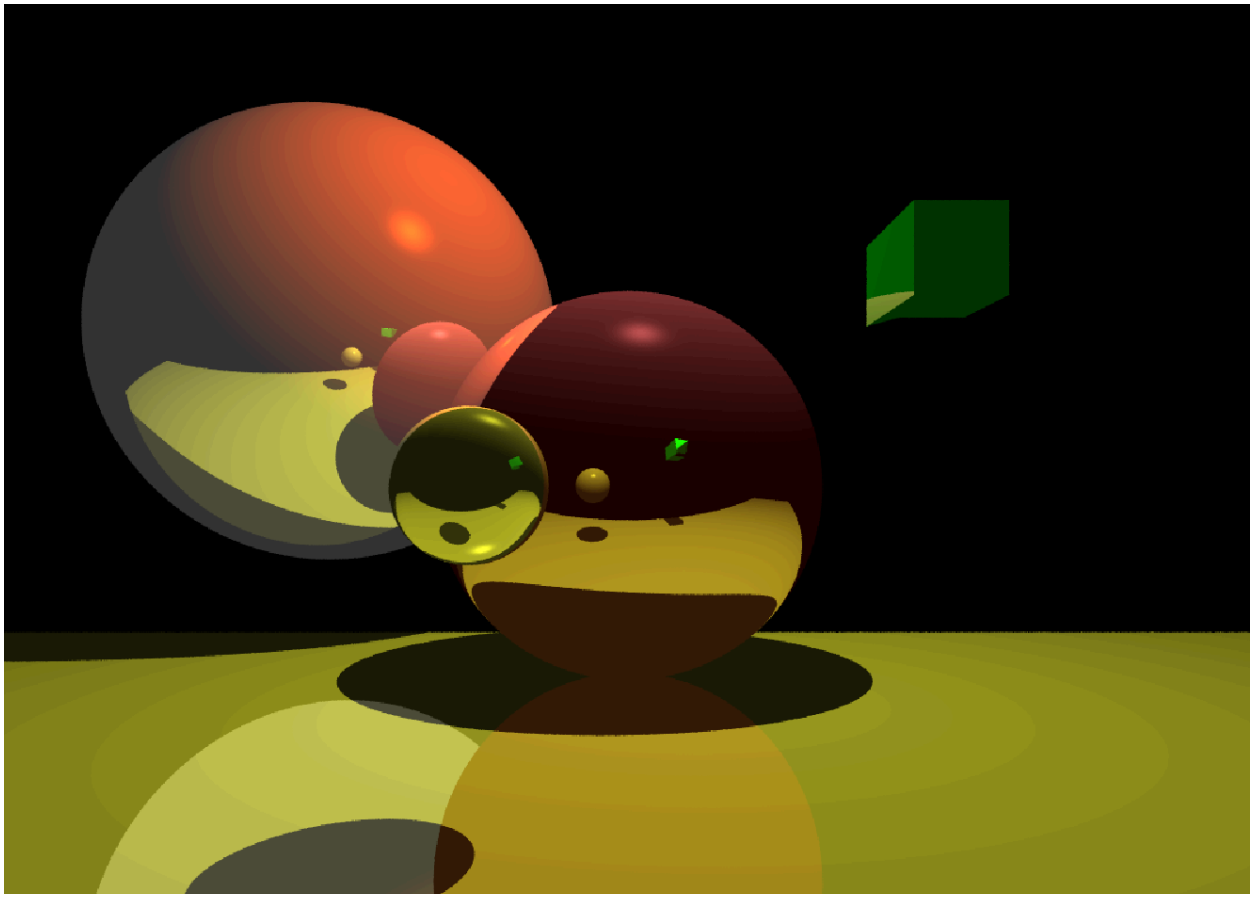
PROGRAMMATION RÉPARTIE

PROJET NOTÉ: CALCUL PARALLÈLE

S4 RA-IL 1 Dimanche 09 Juin 2024



KOMODZINSKI JAWAD
LOPPINET STÉPHANE
RYSAK HUGO
TROHA STANISLAS

TABLE DES MATIÈRES

Préambule :

Question

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.

Après nous être concertés, nous avons voulu recréer quelque chose de semblable au projet Tableau Blanc avec une topologie en étoile. C'est-à-dire un service central auprès duquel les machines de calculs pourraient s'enregistrer. Nous avons donc identifié trois acteurs. Un premier acteur étant le service central. Le deuxième étant les machines "esclaves" chargées des calculs des images. Et le troisième acteur étant le client, qui contacte le service central pour obtenir son image.

L'objectif étant que le client n'effectue pas les calculs de rendu de l'image sur sa propre machine, mais sur plusieurs machines mises à disposition par le service central. Le service central s'occupe de traiter la requête du client qui lui dit, je veux telle image, avec telles dimensions. Le service central s'occupant après de :

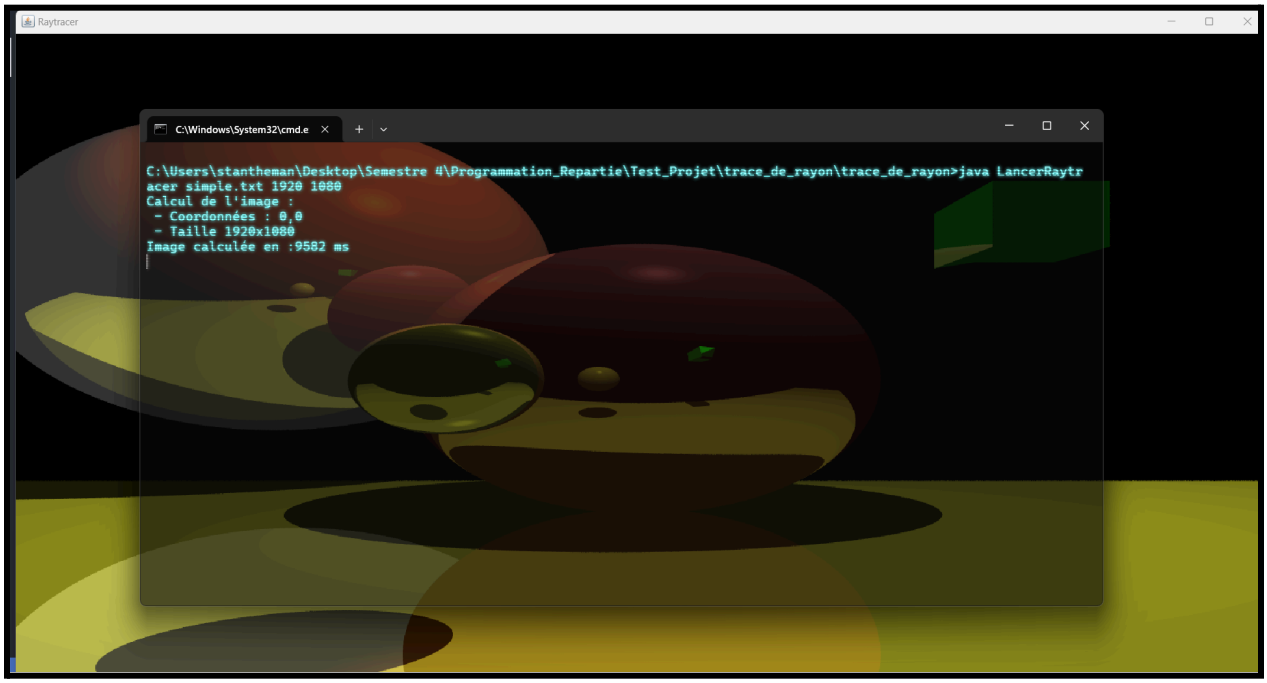
1) Découper l'image demandée par le client en plusieurs fragments en fonction du nombre de machines qui se sont enregistrées auprès de lui.
2) Envoyer à chaque machine les différents fragments d'image qu'elle doit générer/calculer
3) Puis pour chaque fragment, le retourner à la machine client pour que cette dernière l'affiche.

Le tracé de rayon (raytracing)

Questions

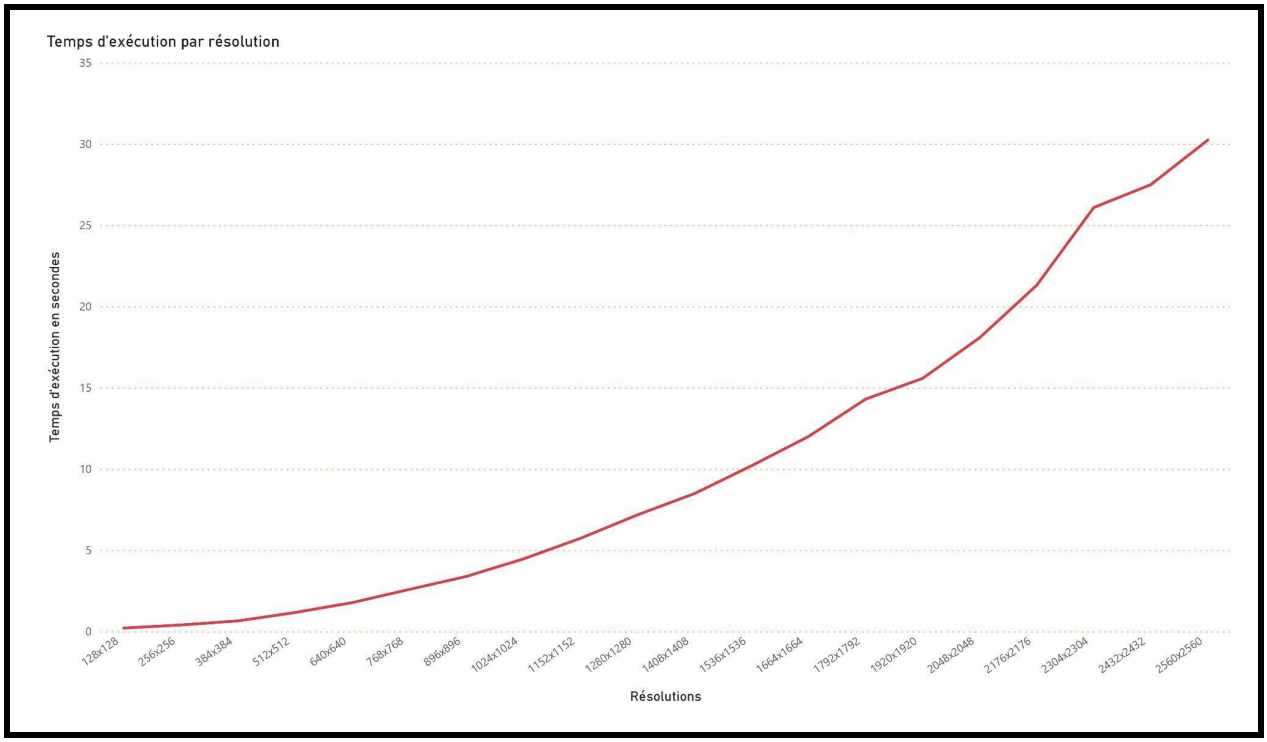
1) Tester le programme en modifiant ses paramètres (sur la ligne de commande).

En prenant des valeurs différentes pour la hauteur et la largeur, l'image s'étire selon la dimension dont on fournit la plus petite valeur. Plus on spécifie des dimensions importantes, plus le rendu de l'image prend de temps. (9,5 secondes pour une image HD contre 1,5 seconde pour une image de taille 512*512).

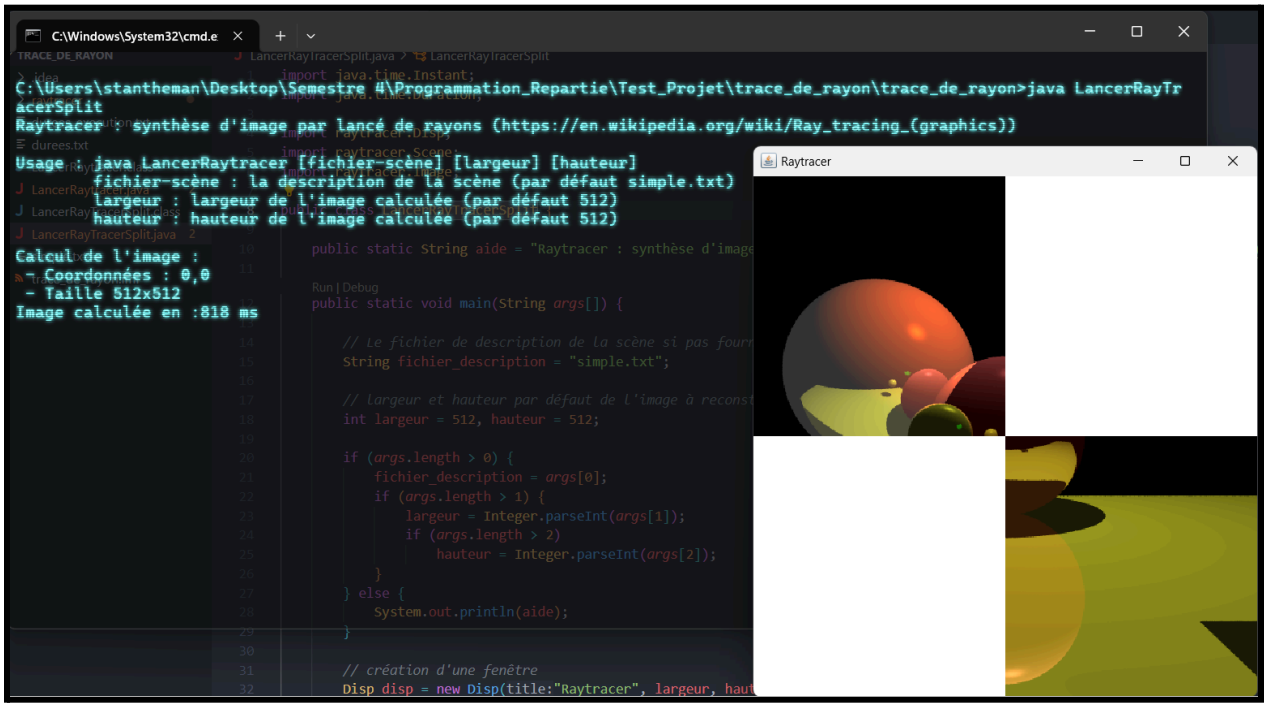


2) Observer le temps de d'exécution en fonction de la taille de l'image calculée. Vous pouvez faire une courbe (temps de calcul et tailles d'image).

Afin de réaliser le graphique ci-dessous, nous nous sommes basés sur des images de taille carrée. C'est-à-dire que pour chaque image, la largeur et la longueur sont similaires (Ex: 512x512, 1280x1280, etc.). Le programme a été lancé 5 fois afin de faire la moyenne des 5 exécutions pour chaque taille (meilleure fiabilité des résultats)
Voici le graphique obtenu (Power BI) :



*3) En ne modifiant **que** le fichier LancerRaytracer.java, reproduire l'image suivante :*



Pour ce faire, nous avons modifié la partie du code qui s'occupe de créer l'image à partir de la scène. Plutôt que de créer une seule image comme le faisait initialement le programme, nous avons créé deux images, disposées par la suite en haut à gauche et en bas à droite. Nous sommes donc passé de ça :

```
int x0 = 0, y0 = 0;

int l = largeur, h = hauteur;

Image image = scene.compute(x0, y0, l, h);

disp.setImage(image, x0, y0);
```

à ça :

```
int x0 = 0, y0 = 0;

int l = largeur, h = hauteur;

Image image = scene.compute(x0, y0, 256, 256);

Image image2 = scene.compute(256, 256, 256, 256);
```

```
disp.setImage(image, x0, y0);

disp.setImage(image2, 256, 256);
```

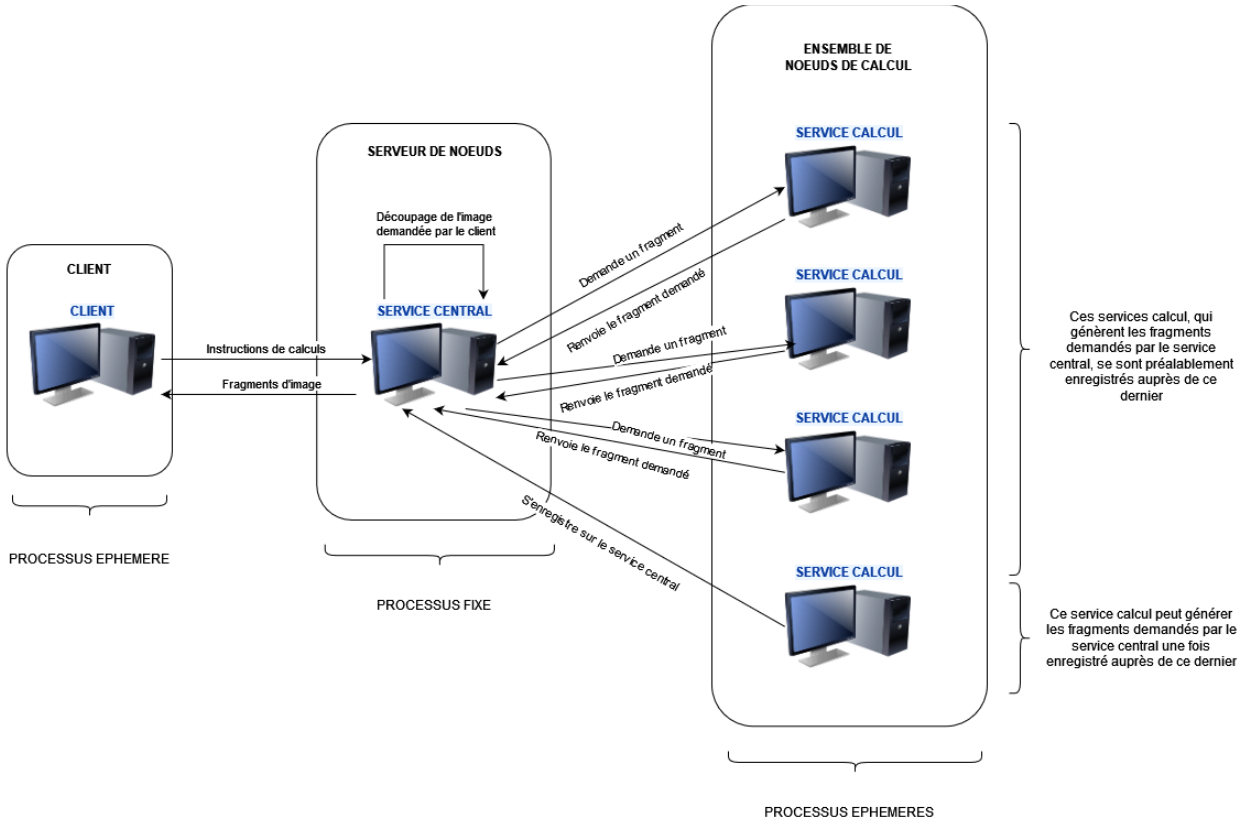
Accélérons les choses

Questions

1) Faire un petit schéma de cette architecture en identifiant les choses suivantes :

1. *Le/les processus fixes (ceux qui écoutent sur un port choisi) et les processus éphémères ? (ceux qui rentrent et sortent à leur guise) ?*
2. *Les types de données échangées entre les processus*

Voici le schéma représentant l'architecture en étoile de l'application, avec le service central au coeur :



2) Si on veut que les calculs se fassent en parallèle, que faut-il faire ?

Initialement, le service central envoie chaque fragment à réaliser à une des machines faisant office de service de calcul, attend le fragment en retour, puis le renvoie au client. Afin de mettre en place un système de calculs parallèles, il faut que le service de calcul puisse envoyer simultanément des instructions de calcul à différentes machines. C'est à dire qu'au moment où il a découpé l'image du client, il puisse ordonner à telle machine de faire tel fragment, puis directement à une autre etc. sans avoir à attendre que la première machine lui renvoie le fragment demandé pour demander à une autre. Ainsi, les appels à la méthode des services calculs peuvent se faire simultanément sur différents Threads.

Il faut donc implémenter les Threads dans la méthode du ServiceCentral qui a pour but de récupérer les fragments composants l'image. L'utilisation des Threads se fait une fois le découpage de l'image demandée par le client faite, lors du parcours des machines service calcul quand il leur demande de calculer des fragments.

3) Lancez-vous et réalisez cette application répartie ! Vérifiez que le calcul est bien accéléré.

Une vidéo de démonstration a été créée pour montrer la parallélisation des calculs à cette URL :

https://youtu.be/57ymhYCMo4A

Code des interfaces créées :

Interface du client :

ServiceClient.java :

```
package Client;

import raytracer.Image;
import raytracer.Scene;

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

/**
 * Interface du Client. Le client c'est celui qui appelle le service
central en lui demandant telle image
 */
public interface ServiceClient extends Remote {

	/**
	* Permet au service central d'appeler le client pour que ce dernier
affiche le fragment
	* @param image
	* @param x0
	* @param y0
	* @throws RemoteException
	*/
	public void afficherFragment(Image image, int x0, int y0) throws
RemoteException;

	/**
	* Permet de récupérer la scene du client
	* @return la scene
	*/
	public Scene getScene() throws RemoteException;
}
```

Interface du service central/serveur :

ServiceDistributeur.java :

```
package Serveur;

import java.rmi.RemoteException;
import java.util.Map;

import Client.ServiceClient;
import ServiceCalcul.ServiceRayTracer;

import java.rmi.Remote;

public interface ServiceDistributeur extends Remote {

	/**
	* Permet aux esclaves de s'enregistrer sur le service central
	*
	* @param r
	* @throws RemoteException
	*/
	public void enregistrerEsclave(ServiceRayTracer r) throws
RemoteException;

	/**
	* Méthode qui permet de construire l'image pour le client
	*/
	public void genererImage(ServiceClient client, int largeur, int
hauteur) throws RemoteException;

	/*
	* Méthode qui renvoie un noeud disponible, actif et pas occupé
	*/
	public ServiceRayTracer distribuerNoeud()throws RemoteException;

	/*
	* Méthode qui renvoie la liste de noeuds
	*/
```

```
      public Map<ServiceRayTracer,String> getServicesRayTracer() throws
RemoteException;
}
```

Interface d’un service de calcul :

ServiceRayTracer.java :

```
package ServiceCalcul;
import java.rmi.Remote;
import java.rmi.RemoteException;
import raytracer.Image;
import raytracer.Scene;

public interface ServiceRayTracer extends Remote{
	/**
	* Méthode qui génère et retourne une image
	* @param x0 coordonnée x de départ
	* @param y0 coordonnée y de départ
	* @param w largeur de l'image généré
	* @param h hauteur de l'image généré
	* @param c la Scene avec laquelle généré l'image
	* @return Image generé
	* @throws RemoteException
	*/
	public Image genererImage(int x0, int y0, int w, int h,Scene c)throws
RemoteException;

	/*
	* Méthode utilisé pour ping le service Calcul et verifié qu'il est
toujours connecté au serveur
	*/
	public boolean estConnecte() throws RemoteException;

	/*
	* Méthode pour savoir si le service Calcul est deja en train de
réaliser un calcul
	*/
	public boolean estOccupe() throws RemoteException;
}
```

Diagramme de séquence d’une exécution complète détaillé (sans Threads) :

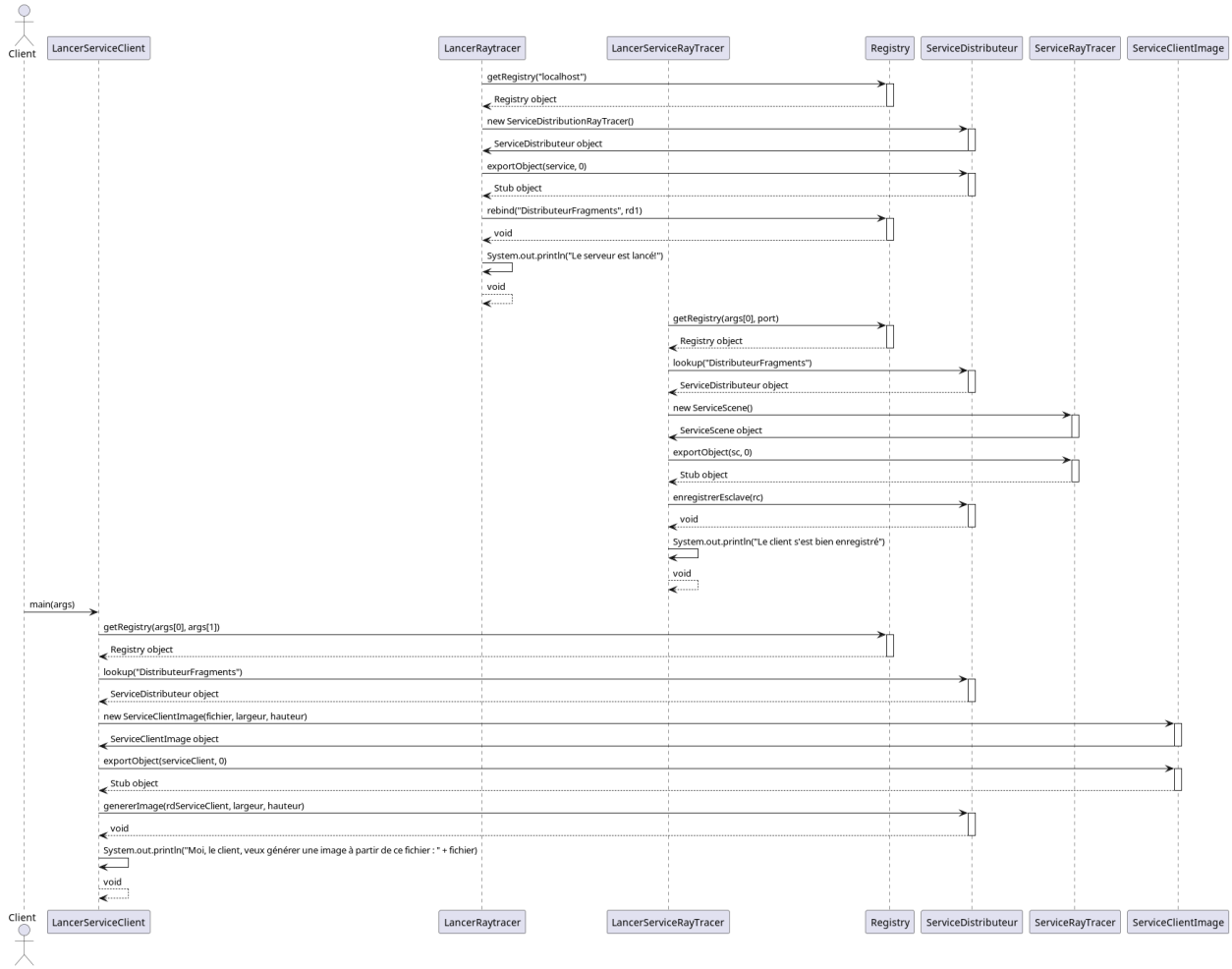
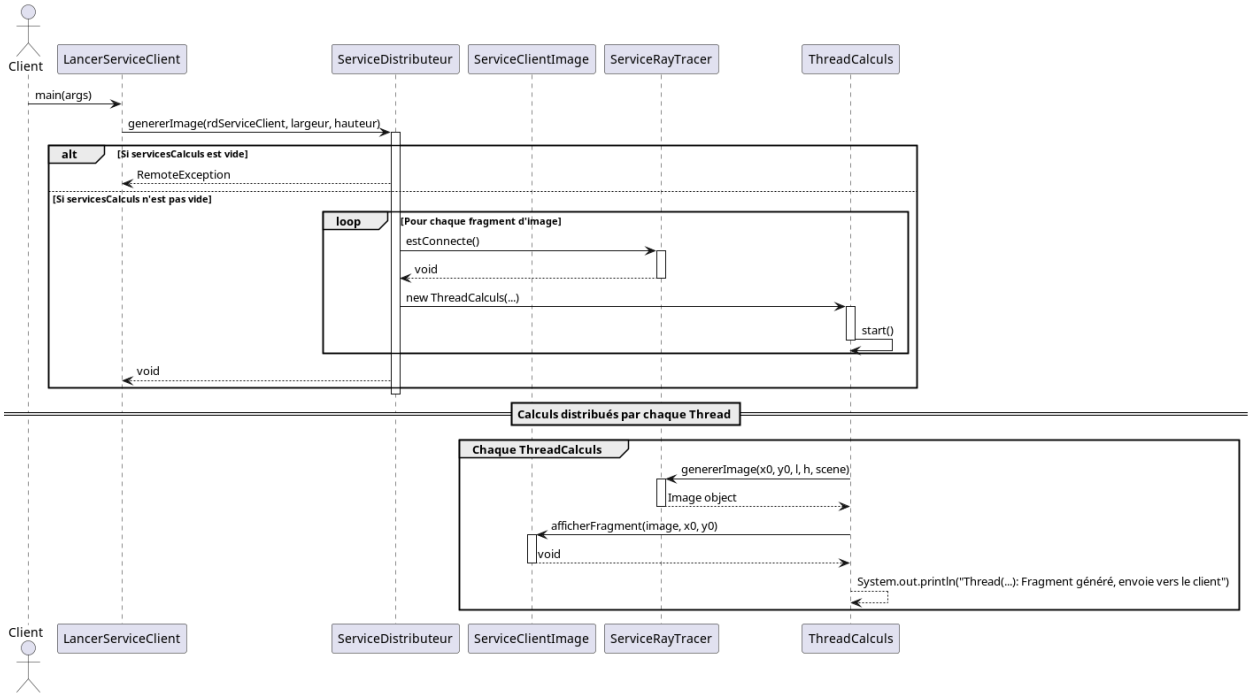


Diagramme de séquence d’une exécution complète (avec Threads) :



Le diagramme de séquence est disponible en plus haute résolution sur Github (Ressources).

Ressources

Repository Github : https://github.com/Komodzin4u/ProjetProgRepartie

Vidéo de démonstration : https://www.youtube.com/watch?v=57ymhYCMo4A

Graphique du rapport entre le temps d'exécution et la résolution de l'image :

https://github.com/Komodzin4u/ProjetProgRepartie/blob/main/Ressources/graphic.jpg

Schéma de l'architecture de l'application en haute résolution :

https://github.com/Komodzin4u/ProjetProgRepartie/blob/main/Ressources/Sch%C3%A9ma_Architecture.png

Diagramme de séquence (Sans Threads) en haute résolution :

https://github.com/Komodzin4u/ProjetProgRepartie/blob/main/Ressources/diagSeq.png

Diagramme de séquence (Threads) en haute résolution :
https://github.com/Komodzin4u/ProjetProgRepartie/blob/main/Diagrammes/diagSeqThreads.png