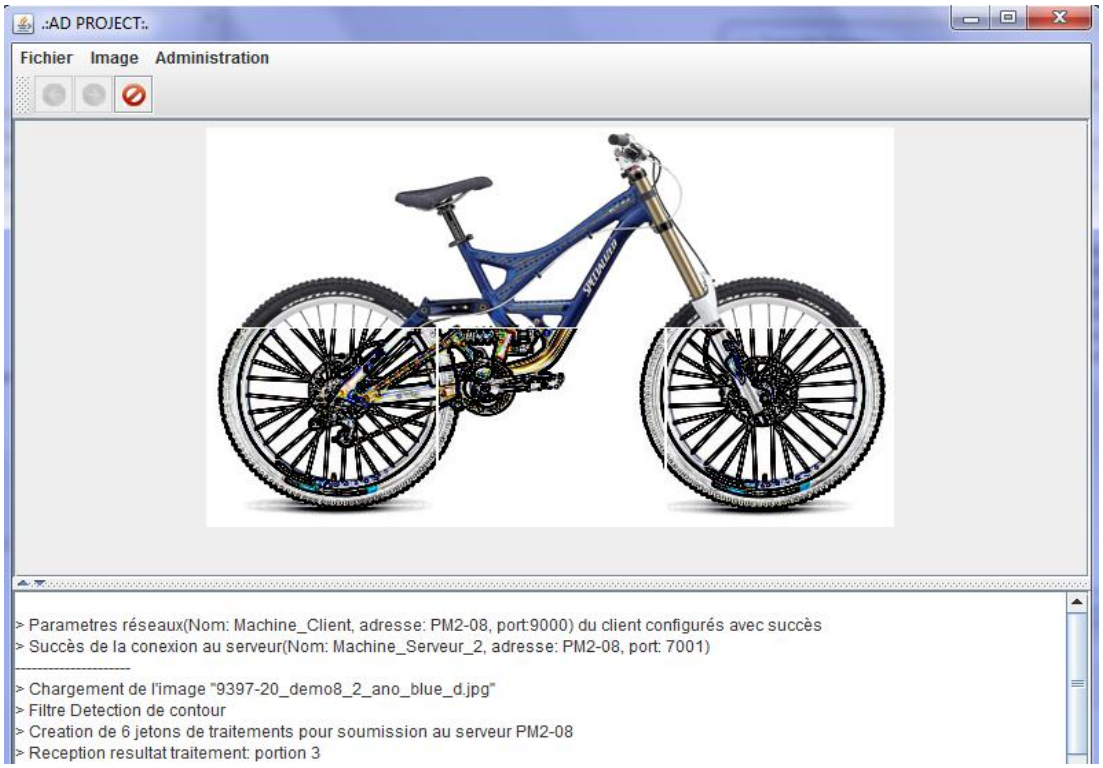
UNIVERSITE DE FRANCHE - COMTE
Master 2 Informatique Option SDR
Novembre 2010

# RAPPORT PROJET ALGORITHMIQUE DISTRIBUÉE
# **TRAITEMENT D'IMAGES DISTRIBUÉ**




**Amine TALBI & Hugues FAGNINOU**

# Table des matières

# I. Introduction

Le projet AD a été réalisé dans le cadre du module Algorithmique Distribuée. Le but du projet et de réaliser un cluster capable d'effectuer des traitements distribués sur des images. L'architecture réseau à adopter a été décrite dans l'énoncé du projet à savoir une architecture sous forme d'anneau avec un objet réseau centralisant les communications.
Un état de terminaison du cluster peut être atteint à la demande de l'administrateur grâce à l'implémentation de l'algorithme Safra. Enfin, l'instauration d'une librairie de traitement d'images a permis d'offrir un ensemble de filtres sur les images.
Dans ce rapport nous insistons principalement sur l'implantation du partage équitable des jobs soumis au cluster ainsi que celle de la terminaison.
Aussi reviendrons-nous en détail sur chaque composant de l'application réalisée ainsi que sur la description du mode d'utilisation présenté en annexe.

# II. Description du mode de distribution du traitement

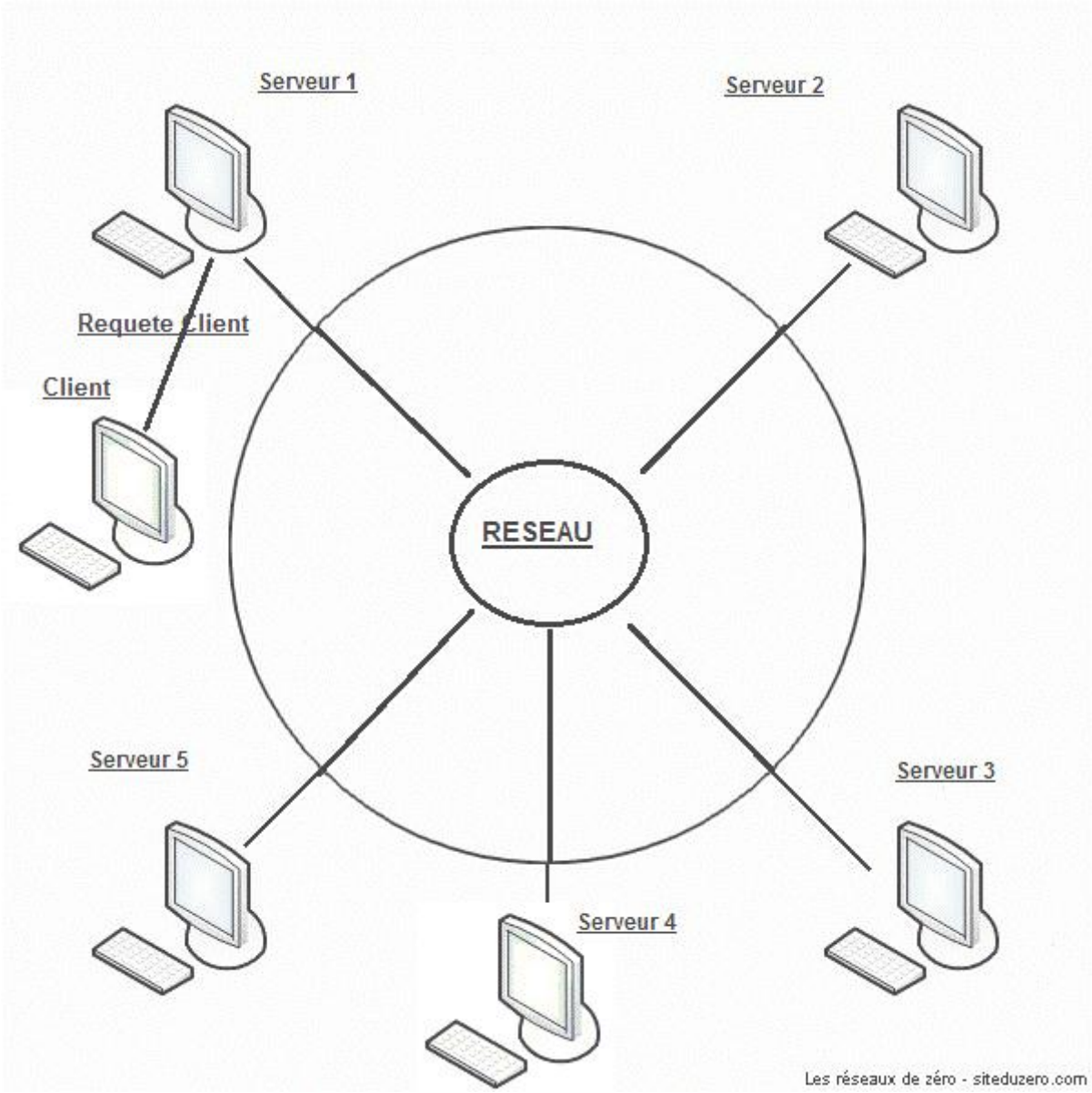



**Figure 1: Aperçu de l'architecture réseau**

Le système implémenté décrit par la figure ci-dessus comporte cinq serveurs interconnectés par une machine réseau au centre. Comme indiqué dans l'énoncé du projet, les cinq serveurs communiquent les uns avec les autres par l'intermédiaire de la machine Réseau tandis que le client ne connait qu'un serveur du système (la machine à qui il peut envoyer directement des demandes de traitement d'images). Le processus global de fonctionnement du système peut se décrire comme suit:

- Lorsqu'un client désire faire traiter une image, il charge en mémoire celle-ci, la découpe en un nombre bien défini, puis envoie un seul jeton portant le numéro de la dernière portion de l'image découpée(les portions d'images portent des numéros allant de 0 à N-1, N étant le nombre portions), et le type de traitement souhaité au

serveur connu (supposons le serveurA) en appelant la méthode « **DemandeTraitraiment** » de la classe Client.

- A la réception du jeton du client, le serveurA parcourt ses threads (classe **ThreadServeur** de type Thread) et en fonction de leur état (libre ou occupé) leur attribue un numéro de portion d'image à traiter. Notons que dès l'attribution d'un numéro de portion d'image à un thread, le numéro porté par le jeton est décrémenté afin d'éviter qu'une même portion ne soit attribuée deux fois; les portions sont donc attribuées aux threads par ordre décroissant de leur numéro.

- Une fois tous les threads d'un serveur servis en tâches à traiter, le serveur envoie par le biais de la machine réseau (invocation distante de la méthode **envoiJetonSuiv()**) le jeton au serveur suivant qui se sert à son tour puis fait passer le jeton au suivant, ainsi de suite.

- Après l'attribution d'un numéro de portion d'image à un thread du serveur, celle-ci se connecte au client grâce aux informations figurant dans le jeton (notamment la référence du client dans le registre RMI), puis récupère à distance, les pixels correspondant dans l'image source. S'en suit le traitement proprement dit (application d'un filtre sur les pixels récupérés). Une fois le traitement terminé, chaque thread serveur retourne au client le résultat du traitement par l'invocation de sa méthode distante **accepterPartieImage().**

- Une fois que le client a reçu la portion d'image qui résulte du traitement, il actualise l'image initiale afin d'intégrer les modifications.

Dans notre implémentation, nous avons fait le choix de l'option du jeton unique envoyé par le client plutôt que celle qui consiste à générer autant de jetons que de portions d'images.
Avec la deuxième option, celle de plusieurs jetons, nous avons été confrontés à un problème de non équité de la charge de travail des différents serveurs. En effet le fait d'envoyer tous les jetons en boucle au premier serveur fait que ce dernier effectue une grande partie du traitement et partage assez mal le traitement avec les serveurs suivants vu qu'il a souvent le temps de finir avec un jeton avant la réception du jeton suivant. L'application de cette option sur plus de deux serveurs montre que les serveurs situés au rang trois ou plus par rapport au serveur qui a reçu la demande de traitement du client, ne reçoivent presque jamais de tâche à traiter.
Par contre, l'option du jeton unique que nous avons adopté permet au premier serveur de se servir, puis de passer le jeton au serveur suivant qui fait pareil puis ainsi de suite tant qu'il reste des portions d'images à traiter. L'équité est ainsi rétablie et chaque serveur ne reprend du travail qu'après que le jeton ait effectué un tour complet de tous les autres serveurs de l'anneau.

Par ailleurs, il est à noter que la communication entre clients, serveurs et réseau a été basée sur du **RMI**. Les trois principales interfaces implémentées pour permettre l'invocation distante de méthodes sont :

- L'interface **IClient** du package **client**:
  Y sont définies les méthodes suivantes :

- ✓ **void accepterPartieImage(Portion_Image partieImg) :** invoquée à distance par les serveurs, cette méthode, leur permet de renvoyer au client, le résultat du traitement demandé ;
- ✓ **Portion_Image getPi(int i)** : elle est invoquée à distance par les serveurs et leur permet de récupérer au niveau du client, la portion d'image correspondant au jeton de traitement reçu.

- L'interface **IReseau** du package **reseau**:
  On y définit les méthodes suivantes:
  - ✓ **void envoiJetonSuiv(Jeton jeton, Machine servSuiv)** : cette méthode permet au serveur de transférer le jeton au serveur suivant dans l'anneau ;

  - ✓ **void setJeton(Jeton jeton)** : cette méthode permet aux serveurs de référencer un nouveau jeton de traitement en l'associant au client correspondant dans une **HMap**. Ce qui permet de gérer les demandes de traitement de plusieurs clients différents simultanément sans risque de mélange ;

  - ✓ **void setJetonPi(String idClient, int pi)** : une fois qu'un serveur a pris une portion d'un numéro **i** donné, il invoque cette méthode pour redéfinir le numéro de la prochaine portion d'image disponible à traiter qui devient **i-1** ;

  - ✓ **Jeton getJeton(String idClient)** : cette méthode permet aux serveurs de récupérer un jeton de traitement en circulation sur l'anneau ;

  - ✓ **void deleteJeton(String idClient)** : cette méthode est invoquée par un serveur chaque fois que l'image d ?un client a totalement été traitée ; ce qui permet de supprimer le jeton correspondant au traitement en question ;

  - ✓ **void ping(String nomServ)** : cette méthode permet de tester la connectivité d'un serveur au réseau.

- L'interface **IServeur** du package **serveur**:
  Elle décrit les méthodes suivantes:
  - ✓ **Machine getMachineServeur()** : permet de récupérer les paramètres réseau du serveur ;

  - ✓ **void recevoirJeton(Jeton jt)** : invoquée par les clients pour formuler une demande de traitement d'image ou pour lancer le processus de terminaison ;

  - ✓ **Machine getServeurSuiv()** : permet de connaître le serveur suivant d'un serveur ;

  - ✓ **void ping(String nomClient)**: permet aux clients de tester la connectivité avec un serveur ;

  - ✓ **boolean isLaunchTerm()**: permet de connaître si un processus de terminaison est en cours sur le serveur.

# III. Description de l'implémentation de l'algorithme de Safra

Comme la transmission des différents jetons est effectuée en mode asynchrone, l'algorithme Safra est le plus adapté pour détecter la terminaison du cluster.
Un état de terminaison est détecté si les deux conditions suivantes sont vérifiées en même temps :

- Tous les serveurs du cluster sont passifs (aucun traitements en cours).
- Le canal de réception ne contient aucun message (aucun jeton de traitement en transit par le réseau).

L'implantation de l'algorithme de Safra nécessite des modifications sur l'algorithme de base et le codage d'un algorithme de contrôle qui vérifiera les conditions de terminaison établis ci-dessus.

1. Modifications sur l'algorithme de base :

- Deux nouveaux types de jeton (**JetonSafra** et **JetonStop**) sont créés. Le premier jeton **JetonSafra** est instancié (couleur=blanc compteur=0) quand l'administrateur demande l'arrêt du cluster. Le **JetonSafra** est ensuite envoyé à l'un des serveurs du réseau connu par l'administrateur.
- A la réception d'un jeton de traitement (de type **JetonTraitement**) provenant d'un serveur, chaque machine du cluster opèrent des changements sur ses variables d'états (compteur décrémenté, couleur passée à noir et le statut à actif).
- Le compteur est incrémenté à l'envoi d'un **JetonTraitement** d'un serveur à un autre.

2. Algorithme de contrôle :

- Une fonction **safra()** est appelée à la réception d'un jeton **JetonSafra**. Cette fonction (correspond à l'algorithme de contrôle) permet de conclure sur la terminaison du cluster en vérifiant les variables d'états de chaque serveur ainsi que les variables estampillées sur le jeton **JetonSafra**.
- Si les conditions de terminaison sont réunies, on lance une annonce de terminaison.

3. Annonce de terminaison :

- Quand la terminaison est détectée, on instancie un jeton de type **JetonStop** et on l'envoie à tous les serveurs (de proche en proche par le biais des serveurs suivant).
- A la réception du **jetonStop** chaque serveur termine, c'est-à-dire, retire sa référence du registre RMI (invocation de la méthode « **unbind** »).
- Une fois que le jeton a fait le tour de l'anneau une notification d'arrêt du cluster est envoyée à l'administrateur par le dernier serveur de l'anneau.

# IV. Autres fonctionnalités implémentées

## 1. Les interfaces Hommes-Machine (IHM)

Nous avons implémenté plusieurs IHM:

- Client:

L'IHM principale permet :

- ✓ de charger une image (menu Fichier) ;
- ✓ de traiter l'image chargée (menu Image) avec possibilité d'annuler ou de rétablir un traitement (fonction UNDO et REDO de la barre d'outils) ;
- ✓ d'enregistrer l'image traitée (menu Fichier) ;
- ✓ à travers le menu Administration, de configurer les paramètres réseau du client (Nom, Adresse machine et port), ainsi que les paramètres réseau du serveur auquel il est connecté ;
- ✓ toujours dans le menu Administration, de configurer le nombre de portions d'images souhaité pour le découpage ;
- ✓ de lancer le processus d'arrêt des serveurs (menu Administration).

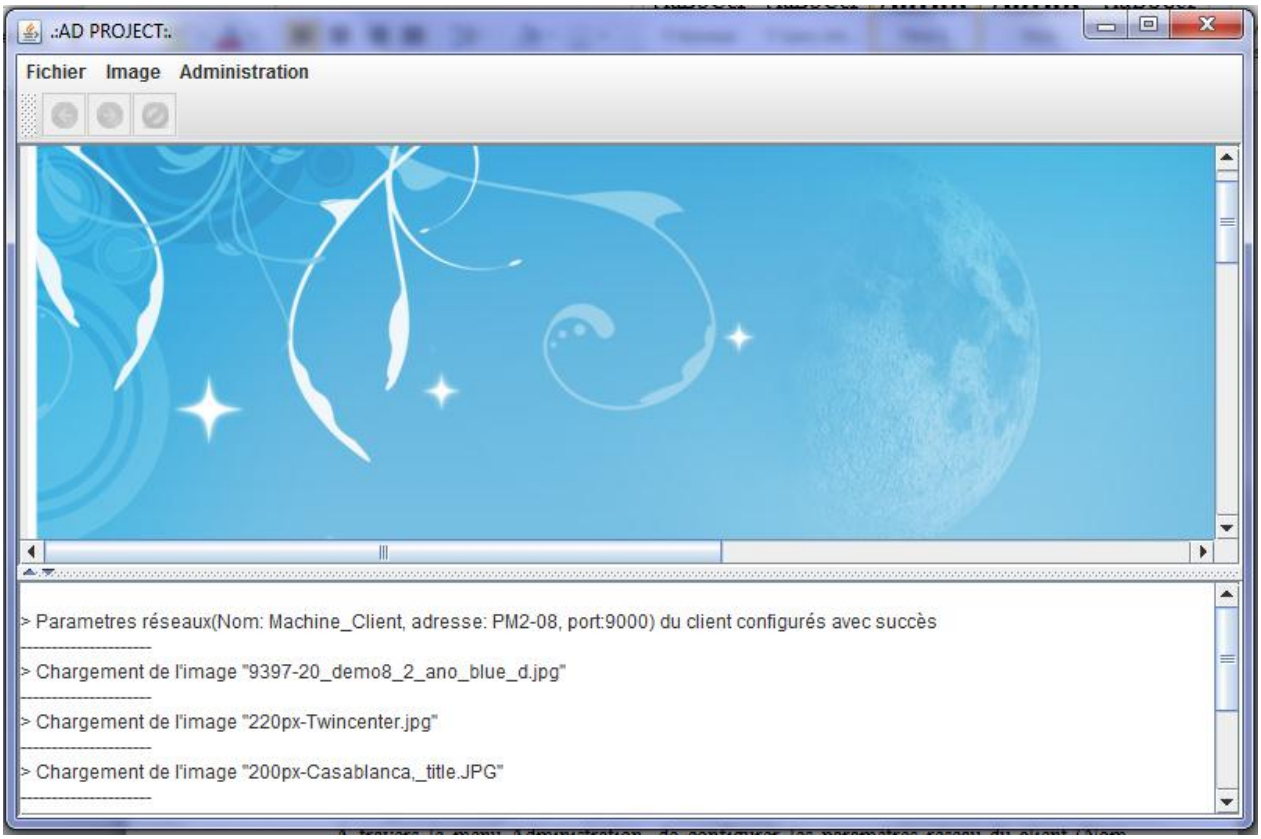



**Figure 2: Aperçu de l'IHM Client**

- Serveur:

L'IHM principale permet :

- ✓ de configurer les paramètres réseau du serveur ainsi que et ceux de son successeur dans la topologie ;
- ✓ de démarrer le serveur.

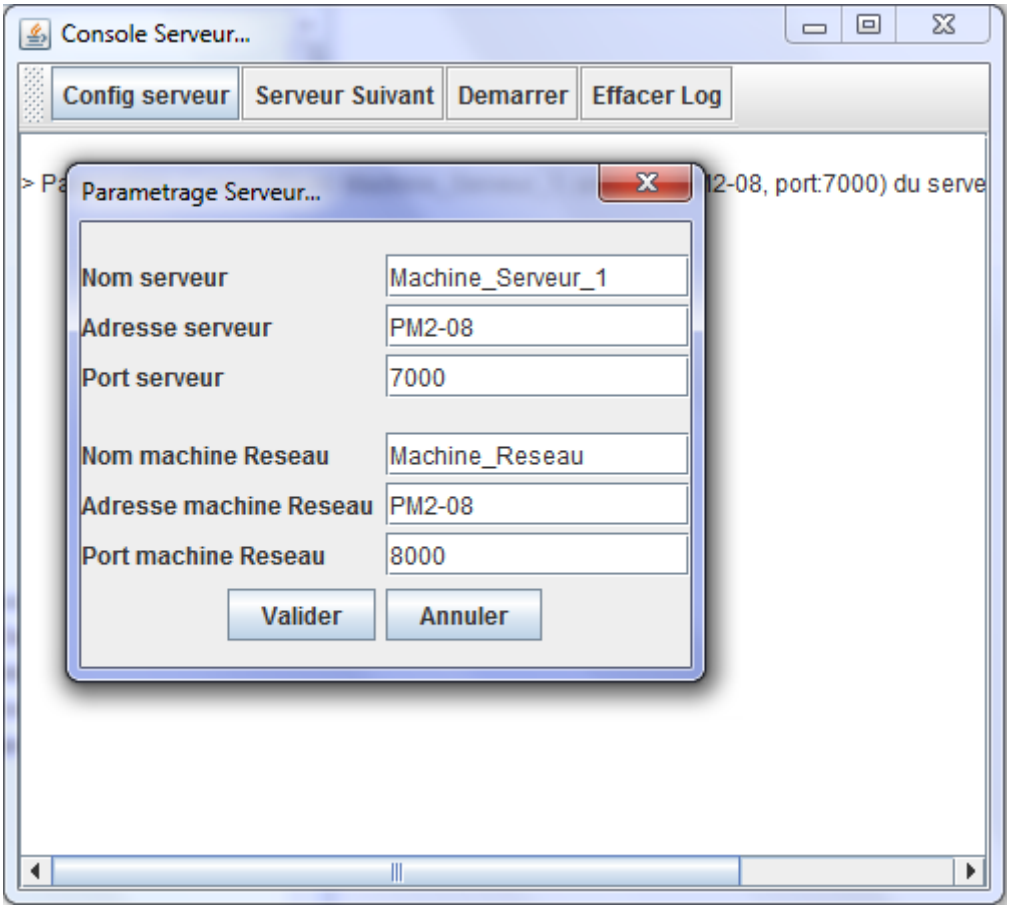



**Figure 3: Aperçu de l'IHM serveur**

- Machine réseau:

L'IHM principale permet :

- ✓ de configurer le délai de transmission des jetons sur le réseau (sachant que le délai réel utilisé est égal au délai indiqué, multiplié par un nombre aléatoire compris entre 0 et 1) ;
- ✓ de configurer les paramètres réseaux ;
- ✓ de démarrer la machine réseau.

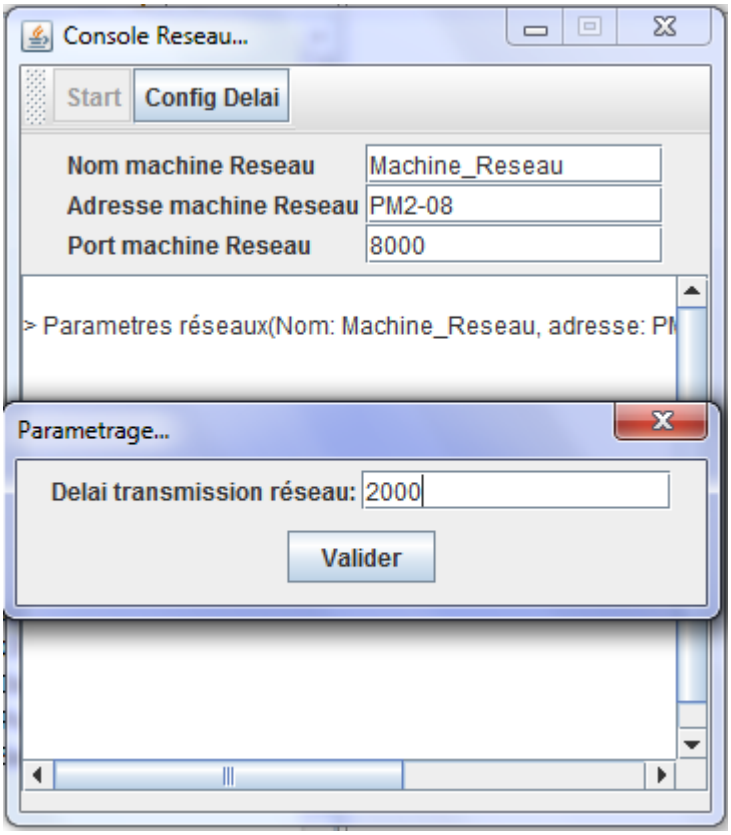



**Figure 4: Aperçu de l'IHM Réseau**

2. Une fonction de traçabilité des traitements effectués

Pour des raisons de visibilité, nous avons implémenté un modèle de log qui permet de connaitre à chaque instant, les traitements en cours ou les traitements effectués. Le système de log mis en place est basé sur une implémentation du pattern Observateur. Les IHM Client, Serveur et Réseau en sont pourvus. Ainsi, par exemple, les logs de l'IHM d'un serveur sont automatiquement mis à jour suivant les différents états de celui-ci.

```
> Parametres réseaux(Nom: Machine_Serveur_1, adresse: PM2-08, port:7000) du serveur configurés avec succès
> Serveur Machine_Serveur_1 créé  (Machine : PM2-08, N° de port : 7000, Nbre de Thread =2)
> Connexion établie avec succès avec la machine Reseau
> Ajout de la machine Machine_Serveur_2(Adresse: PM2-08/ port: 7001) comme serveur suivant...
> Reception jeton numero 2 du client(Machine_Client) pour traitement(Detection de Contour)
> Lancement traitement(client 'Machine_Client', filtre 'Detection de Contour', jeton 2)
> Reception jeton numero 1 du client(Machine_Client) pour traitement(Detection de Contour)
> Lancement traitement(client 'Machine_Client', filtre 'Detection de Contour', jeton 1)
> Fin traitement jeton 2, resultat envoyé au client
> Fin traitement jeton 1, resultat envoyé au client
> Serveur arrêté...
```

**Figure 5: Aperçu des logs d'un serveur**

3. Les algorithmes de traitement d'images

En complément de l'algorithme de détection de contour fourni, nous avons implémenté trois autres algorithmes de traitement d'images à savoir:

- ✓ le niveau de gris
- ✓ la saturation d'intensité -200
- ✓ le filtre RVB (avec possibilité de combinaison des couleurs primitives rouge, vert et bleu)

# V. Conclusion

L'application développée dans le cadre du projet AD a été notre première expérience dans la réalisation d'une plateforme de traitement distribué.
L'implantation de la terminaison nous a permis de mettre en application l'algorithme de Safra étudié en cours. Tandis que la synchronisation des différents threads de traitements et le partage de tâches entre les serveurs, nous ont permis de mettre en œuvre les techniques de synchronisation vus en TP d'Algorithmique Distribuée. Tout cela, couplé à l'utilisation de l'API RMI, nous a permis, de nous familiariser avec la manipulation de différents objets sur des machines distantes grâce à leurs références RMI respectives.

# VI. Annexes (Mode d'utilisation)

Les jars du serveur, client et réseau sont  dans le dossier final dans le zip du rendu.

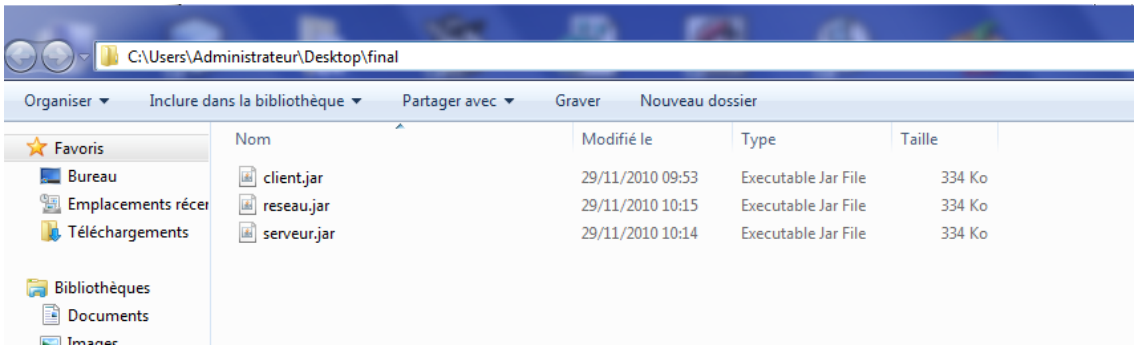



**Figure 6: Aperçu du dossier final contenant les exécutables**

Démarrez (clic sur le bouton « **Start** ») d'abord le réseau en validant les paramètres par défauts.

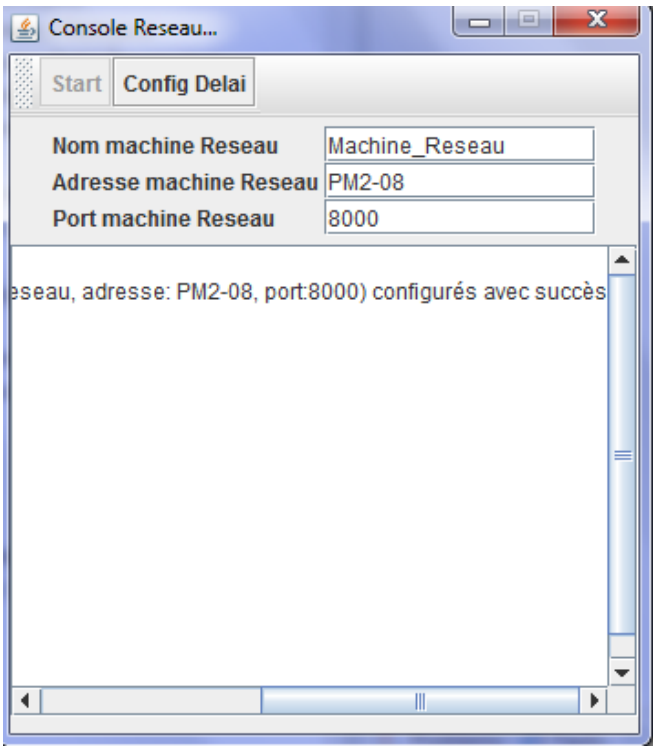



**Figure 7: Aperçu de l'IHM Réseau en mode démarré**

Une fois le réseau démarré, lancez Les serveurs de la plateforme (cinq serveurs comme indiqué dans l'énoncé).

La fenêtre de configuration des serveurs s'affiche au clic sur le bouton « **Config serveur** ».
Toutes les informations devront être saisies pour pouvoir connecter le serveur au réseau. Cliquez ensuite sur le bouton « **Démarrer** » pour lancer le serveur.

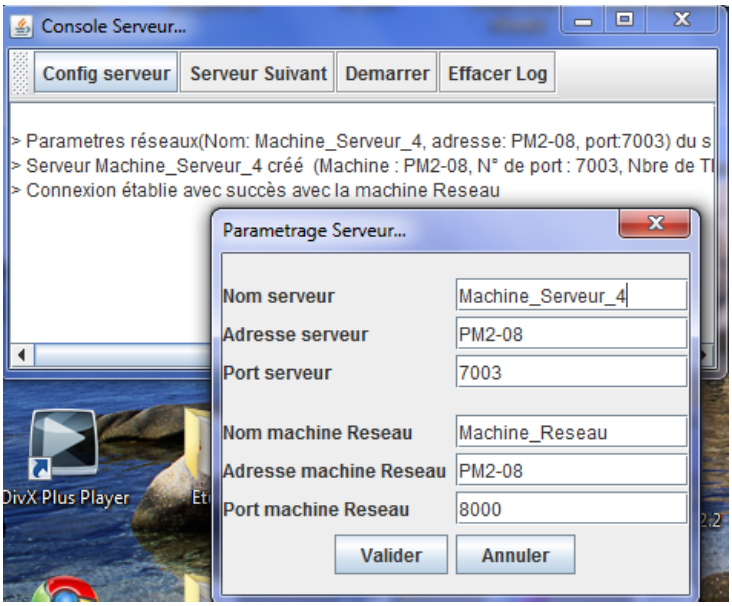



**Figure 8 : Aperçu de la fenêtre de configuration d'un serveur**

Cliquez sur le bouton « **Serveur Suivant** » pour configurer le serveur suivant, puis saisissez les informations demandées.


**Figure 9: Aperçu de la fenêtre de configuration du serveur suivant**

Un test en local est illustré dans la capture d'image suivante.
Chaque fenêtre permet d'avoir les logs de chaque serveur. La console réseau montre l'ensemble de serveurs connectés.

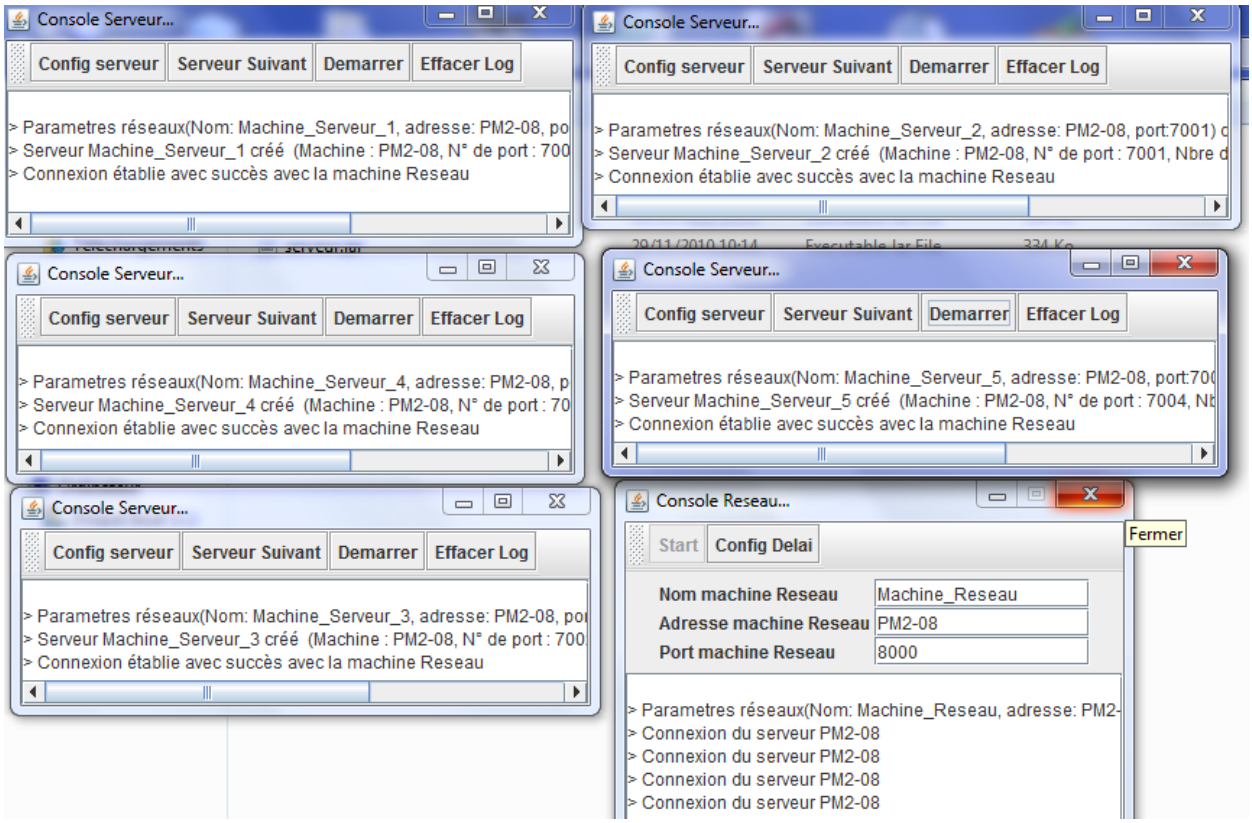



**Figure 10 : Aperçu d'un ensemble de cinq serveurs et d'une machine réseau lancés**

Lancez enfin l'IHM Client puis configurez ses paramètres réseaux par le menu **Administration/ Paramètres de connexion client.**

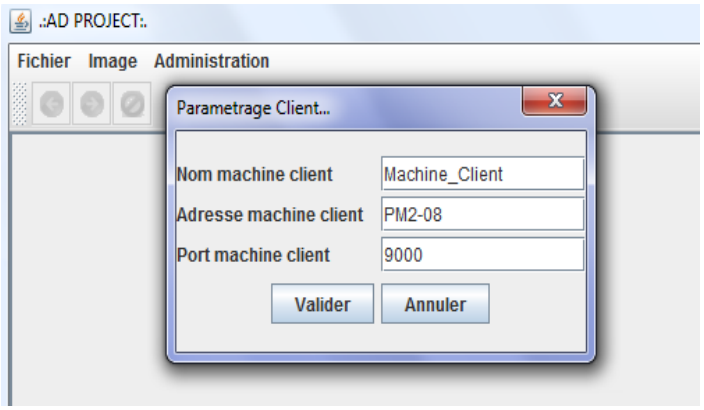



**Figure 11: Aperçu de la fenêtre de configuration des paramètres réseau du client**

Indiquez pour finir les informations réseaux relatives au serveur auquel sera connecté le client par le menu **Administration/Paramètres de connexion Serveur**.

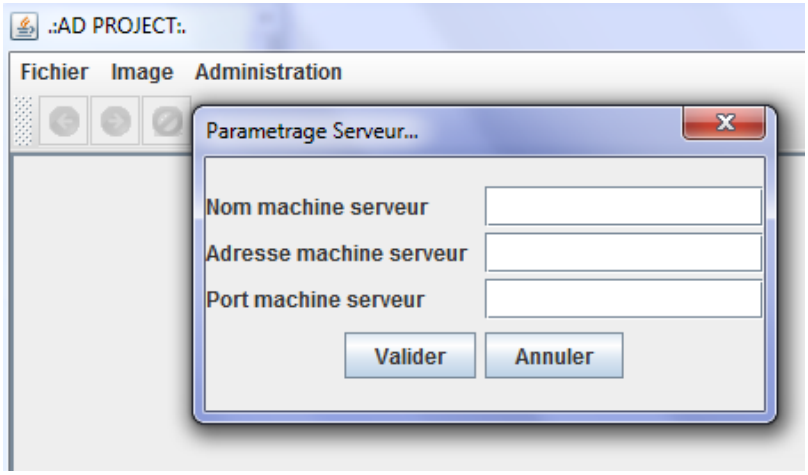



**Figure 12: Aperçu de la fenêtre de configuration des paramètres réseau du client**

Pour modifier le nombre de portions d'images souhaité pour le découpage, allez dans le menu **Administration/Paramétrer le découpage.**

Pour charger une image à traiter, allez dans le menu **Fichier/Charger une image** et pour lancer un traitement, allez dans le menu **Image** et choisissez un filtre à appliquer

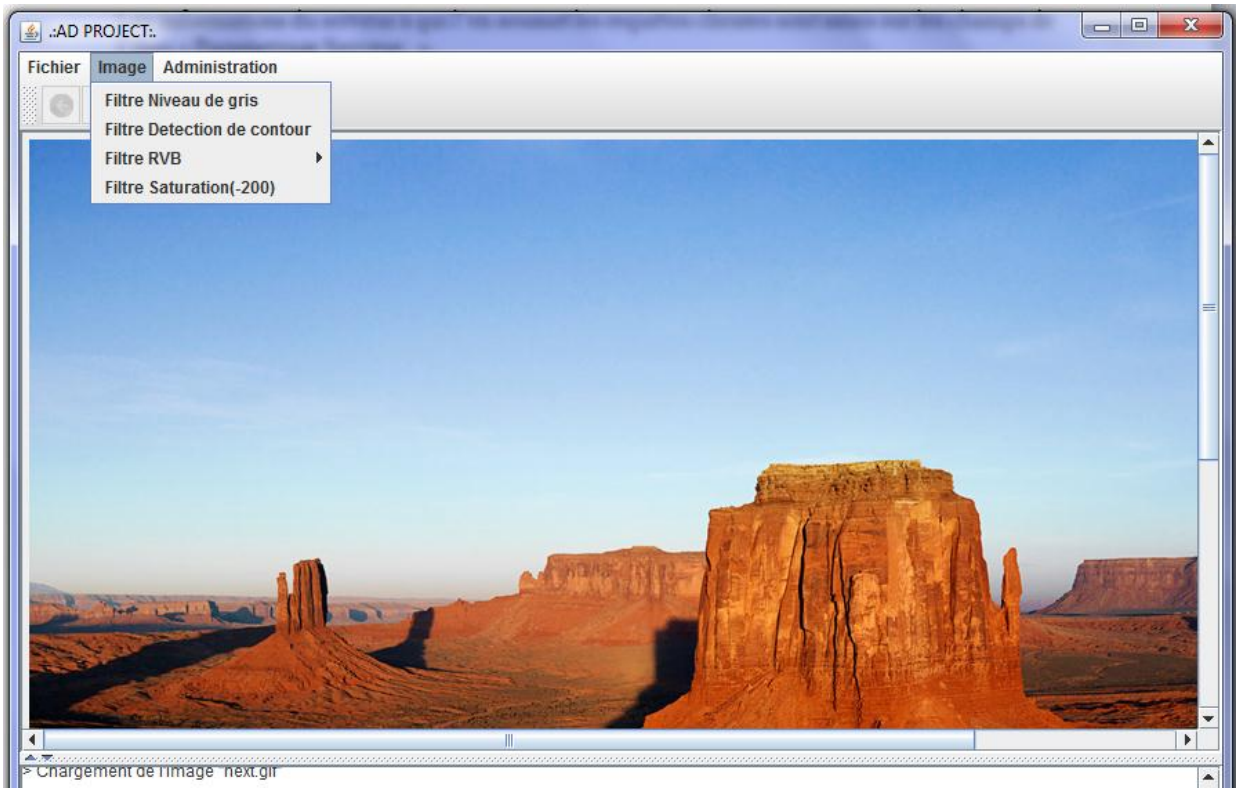



**Figure 13: Aperçu de l'IHM client avec une image chargée**

# VII. Table des illustrations