Projet de système distribué
Rapport

Matthieu MICHELS Isaac ABOTSI

May 22, 2015

Présentation du problème

Après lecture du sujet, nous avons réfléchi aux différentes manières de modéliser notre problème. Ainsi, nous nous sommes lancé comme défi que la carte ne comporterait non pas une seule porte, mais plusieurs objectifs à atteindre. Ainsi, notre modèle permet de s'attaquer au célèbre "problème du commerçant voyageur". C'est cette ambition qui nous a motivé à nous investir autant dans notre projet.

Répartition du travail

Nous avons commencé par deux ou trois séances d'analyse du sujet, lors desquelles chacun soumettait ses idées à l'autre, afin de confronter nos interprétations du cahier des charges. Après avoir extrait plusieurs tâches de notre réflexion commune, nous les avons groupé en trois thèmes principaux (génétique, distribué et interface).
Ensuite, nous inspirant de la méthode agile "Extreme Programming", nous avons appliqué la technique de programmation en binôme. Grace à cela, nous avons pu corriger la plupart de nos erreurs, afin d'avoir rapidement une base fonctionnel pour la partie génétique. À partir de là, nous avons implicitement développé plus particulièrement un thème chacun (Matthieu : distribué; Isaac : interface) pendant quelques jours. Assez rapidement, nous avons à refait converger nos capacités de développement pour refonder le cœur de tout algorithme génétique : la fonction d'évaluation. En effet, nous avons pris la décision de passer d'une représentation matricielle de la carte, à une représentation vectorielle, afin de simplifier les calculs de collision.
Nous avons ainsi réfléchi et conçu ensemble chaque caractéristique de notre application, développant parfois séparément mais souvent en binôme.

Interface de l'application

La partie interface gère l'affichage du résultat, mais aussi la saisie des paramètres par l'utilisateur. En effet, nous avons mis un autre point d'honneur à créer une interface graphique pour offrir la possibilité de modifier bon nombre des données de l'algorithme génétique. De ce fait, nous avons plutôt créé un outil pour trouver de manière empirique les meilleurs paramètres pour trouver un chemin optimal.

Algorithme génétique

Boucle principale

La partie algorithme génétique est assez simple et est décrit par le pseudo code suivant :

```
Initialiser la population.

Boucle infinie:
    Emigrer des individus vers son serveur.

    Tant qu'on a des enfants à ajouter:
        Selectionner Parent1 et Parent2.
        Opérateur génétique de crossover sur Parent1 et Parent2 => Enfant.
        Opérateur génétique de mutation sur Enfant => Enfant.
        Evaluation du genotype Enfant.
        Ajouter Enfant à la nouvelle génération.
    Fin tant que.

    Immigrer des individus des serveurs voisins.

    Tant que la nouvelle génération n'a pas 100 individu:
        Ajouter les meilleurs de l'ancienne génération.
    Fin tant que.
Fin boucle.
```

ADN

Au niveau de la partie génétique, nous avons conçu les choses comme suit. La longueur d'un ADN est fixée dans le code (par un #define) à 100, mais la taille de pas est variable au sein du même ADN. Cela ajoute une difficulté supplémentaire pour trouver le chemin minimal.

Séléction / Reproduction

Lors de la phase de séléction, nous avons fait le choix limiter l'élitisme. Alors que le sujet propose de choisir les meilleurs parmi l'union de l'ancienne et la nouvelle génération, nous remplaçons les n moins bons individus de la génération actuelle par n nouveaux enfants (n étant fixé au départ). En réduisant l'élitisme de notre population, nous favorisons sa diversité, ce qui a pour effet d'éviter

d'être attiré vers les optimums locaux.
Nous avons du faire un choix pour une fonction selection possible parmi les suivantes :

- Une selection élitiste où on reproduit entre eux les meilleurs éléments de la population.
- Une selection roulette où chaque individu de la génération courante a une probabilité de procréer qui dépend de sa qualité (après évaluation). Les meilleurs individus ont donc une plus grande chance d'être selectionnés pour la reproduction.
- Une selection tournoi. On prend en paramètre une taille de tournoi 'n' et on choisit 'n' individu au hasard dans la population. On retourne ensuite les deux meilleurs individus.

Nous avons opté pour la selection tournoi, car elle est à la fois rapide, tout en permettant à des individus moins bien classés une chance de s'améliorer. Cela a pour conséquence de diversifier la population contrairement à une approche de selection élitiste.

Évaluation

On parcourt le chemin (ou ADN ou génotype), et on teste à chaque mouvement (gène) si on est entré en collision avec un objet de la carte. Si on rencontre un obstacle, le mouvement est annulé, et si c'est un objectif on augmente le score de cet individu.
Cette note est fonction de la distance parcourue pour atteindre cet objectif : moins on a fait de pas pour atteindre un objectif, plus la note est élevée. Par ailleurs, les premiers objectifs ont un plus grand poids dans la note de l'individu que les derniers, afin que l'algorithme améliore en priorité le début du chemin, qui influence la fin.

Algorithme distribué

Notre deuxième grand défi réside dans le type d'architecture distribuée que nous avons mis en place. Nous avons opté pour une vision décentralisée, où chaque programme a le même rôle, indépendant des autres (voir Figure 1).

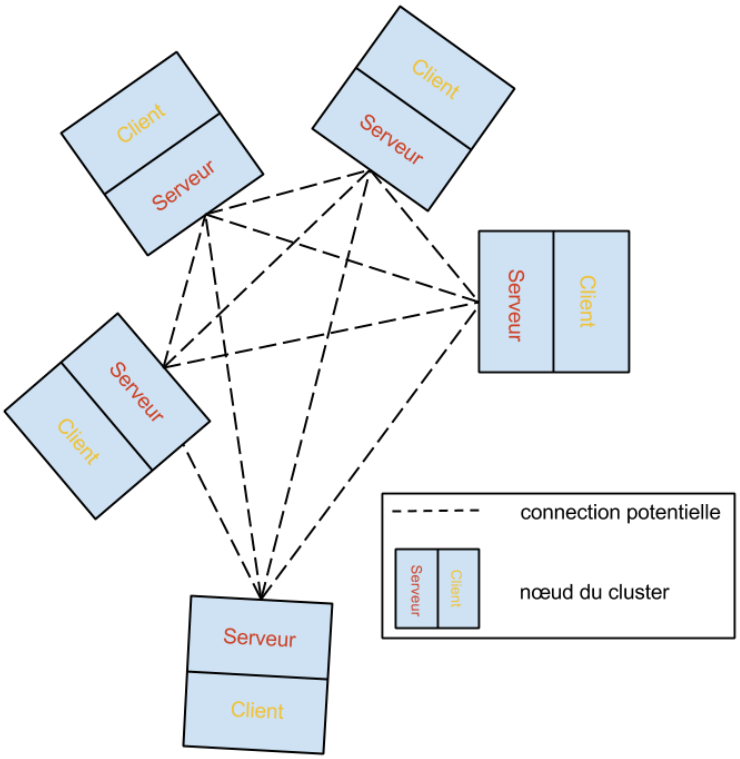

Figure 1: Graphe de connection en maillage complet d'un cluster de 5 nœuds

Pour ce faire, chaque instance de notre programme est divisé en deux processus : un premier, que nous appellerons "client", execute la boucle principale de l'algorithme génétique, tandis que le second, que nous désignerons par le terme "serveur", construit et maintient un graphe de connexion. De cette façon, l'architecture applicative que nous avons mis place est TOTALEMENT distribuée.

Le schéma ci-contre (Figure 2) décrit la création du premier noeud d'un cluster. Tout d'abord on créé la paire client/serveur. Lorsque le serveur est prêt, le client envoie son adresse, la carte qui va être utilisé par l'algorithme et ses premiers migrants. Même si ces premiers migrants ne sont pas encore les meilleurs individus de la population, cette initialisation permet de pouvoir appeler la fonction d'immigration vers n'importe quel nœud du réseau, même les nouveaux.

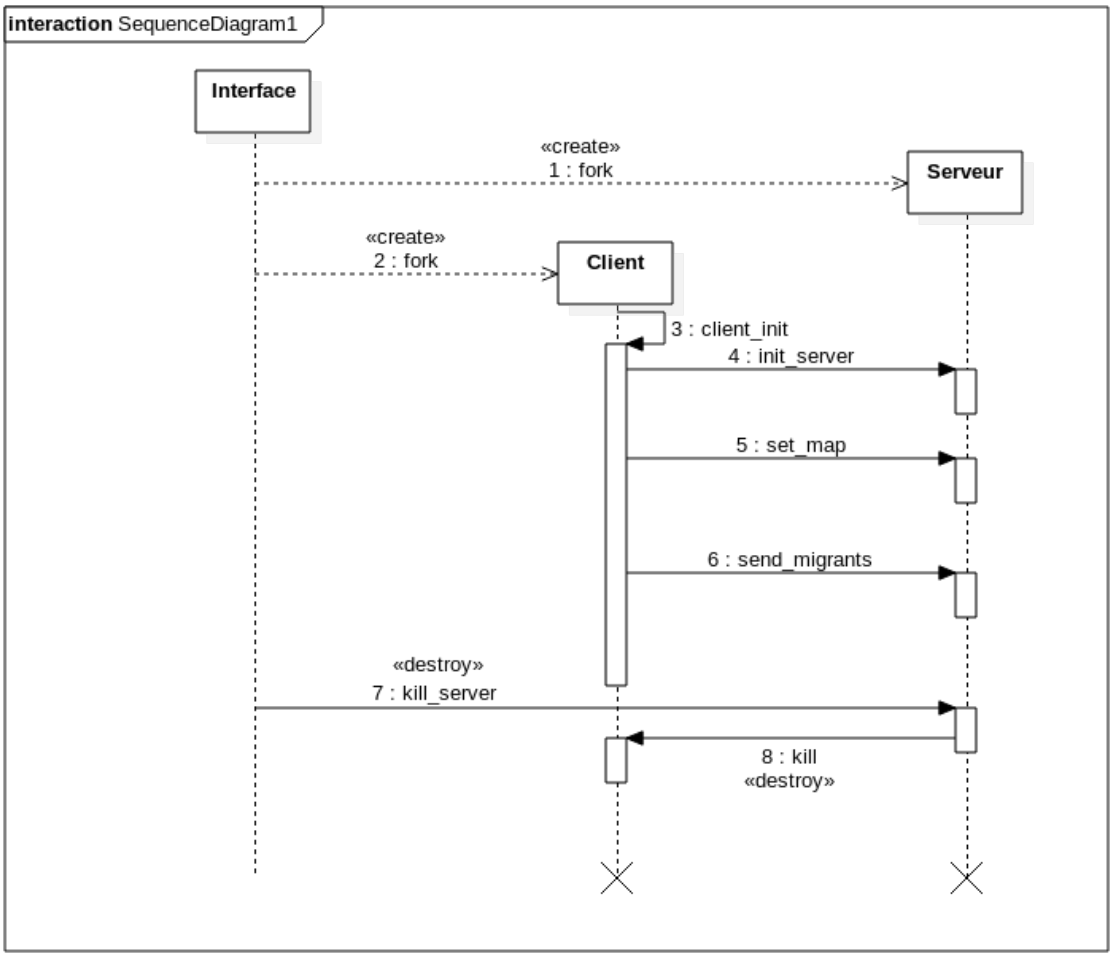

Figure 2: Diagramme de séquence de la création du premier noeud d'un cluster

Lorsqu'un noeud rejoint un cluster, il va à nouveau créer un couple client/serveur (Figure 3). Le client s'annonce au reste du cluster à travers le serveur, qui va demander à son "contact" la liste des nœuds du réseau et la carte utilisée pour l'algorithme génétique. Ensuite il va envoyer un message d'ajout à chaque serveur dans la liste. De cette façon, chaque nouveau nœud est connu de tout le réseau, et vice-versa.

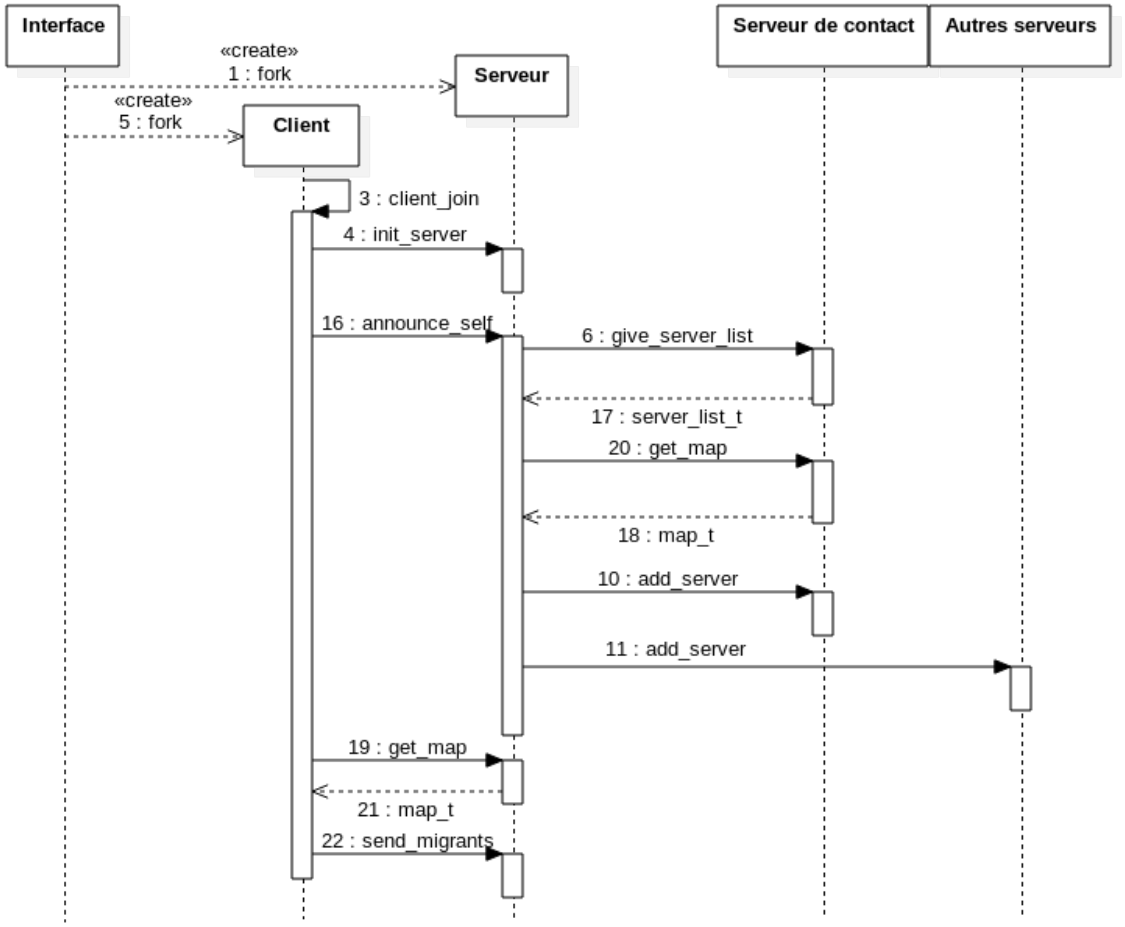

Figure 3: Diagramme de séquence de la création d'un noeud se connectant à un cluster