Méthodes de distribution pour les simulations de mobilité des voyageurs

Matthieu Mastio, Mahdi Zargayouna, Omer Rana, Gérard Scemama

Méthodes de distribution pour les simulations de mobilité des voyageurs

Matthieu Mastio[a] matthieu.mastio@ifsttar.fr
Mahdi Zargayouna[a] hamza-mahdi.zargayouna@ifsttar.fr
Omer Rana[b] ranaof@cardiff.ac.uk
Gerard Scemama[a] gerard.scemama@ifsttar.fr

[a] Université Paris Est, IFSTTAR, GRETTIA, 10-14 Boulevard Newton, 77447 Champs sur Marne, France

[b] School of Computer Science & Informatics, Cardiff University, United Kingdom

Résumé

Avec la généralisation de l'information voyageurs en temps réel, la dynamique des réseaux de transport est de plus en plus difficile à analyser et à prévoir. Il devient nécessaire de développer des outils de simulation pour les décideurs de politiques de mobilité, tenant compte de ce nouvel environnement informationnel. En effet, informer un très grand nombre de voyageurs guidés individuellement peut avoir des conséquences importantes sur l'état du trafic. Il est utile d'évaluer cet impact par la simulation. Or, les simulations multi-agents existantes pour la mobilité de voyageurs ne peuvent prendre en considération qu'une partie du volume réel de voyageurs. En distribuant ces simulations, il serait possible de prendre en compte des volumes réels de voyageurs connectés et d'analyser et de prévoir l'état des réseaux de transport. Dans cet article, nous proposons une comparaison entre deux méthodes pour la distribution des simulations multi-agents de mobilité des voyageurs, permettant la prise en compte de flux réalistes et de zones géographiques étendues.

Mots-clés : *Simulation, Calcul haute performance, Systèmes distribués, Systèmes Multi-agents, Transport*

Abstract

With the generalization of real-time traveler information, the behavior of modern transport networks becomes harder to analyze and to predict. It is now critical to develop simulation tools for mobility policies decision makers, taking into account this new information environment. Indeed, the spread of individualized information may highly influence the traffic network. However, existing mobility multiagent and micro-simulations can only consider a sample of the real volumes of travelers, especially for large areas. With distributed simulations, it would become possible to analyze and predict the status of current and future networks, with informed and connected travelers. In this paper, we propose a comparison between two methods for distributing multiagent travelers mobility simulations, allowing for the consideration of realistic travelers flows and wide geographic areas.

Keywords: *Simulation, High performance computing, Distributed systems, Multiagent, Transport*

1 Introduction

Les systèmes de transport se complexifient de jour en jour et ils doivent évoluer pour intégrer de nouvelles entités connectées (appareils mobiles, véhicules connectés, etc.). Nous pouvons maintenant non seulement fournir des itinéraires optimaux pour les voyageurs, mais nous sommes également en mesure de mettre à jour ces itinéraires en temps réel en fonction de l'état du réseau (congestions, accidents, véhicule de transport en commun ou covoiturage annulé, etc). Donner aux utilisateurs du réseau des informations sur l'état du trafic est généralement bénéfique et permet l'amélioration globale de l'écoulement des flux de voyageurs dans les réseaux de transport.

Toutefois, la diffusion massive de l'information via des panneaux d'affichage, des annonces à la radio ou par le biais d'un guidage individuel peut avoir des effets pervers et créer de nouveaux embouteillages. En effet, avec la généralisation de l'information voyageurs en temps réel, le comportement des réseaux de transport modernes devient plus difficile à analyser et à prévoir. Il devient alors important de modéliser finement et de simuler un nombre réaliste de voyageurs en interaction avec les sources d'information afin d'observer précisément ces effets dans un environnement virtuel contrôlé. La possibilité d'exécuter un simulateur de trafic à l'échelle d'une ville, d'une région ou d'un pays permettrait d'observer les conséquences de différentes stratégies d'information sur l'état du

trafic multimodal avant de les appliquer dans le monde réel. Sur la base des résultats de simulation, il est possible de comparer les prévisions réalisées par le simulateur avec des données mesurées sur le réseau réel afin d'affiner itérativement le modèle.

FIGURE 1 – Le simulateur SM4T [25].

Nous avons récemment développé SM4T (*Multiagent MultiModal Mobility of Travelers*), une plateforme de simulation de mobilité multimodale [26] (cf. Figure 1). Cette plateforme simule le comportement de voyageurs qui interagissent dans un environnement complexe, dynamique et ouvert dans lequel ils n'ont qu'une perception partielle. Elle est fondée sur le paradigme multi-agent, particulièrement adapté pour ce type de simulation [2]. Chaque agent essaye de trouver l'itinéraire le plus efficace pour atteindre sa destination dans un réseau évoluant dynamiquement. Un agent peut potentiellement être informé de l'état du réseau et tirer parti de cette information pour modifier l'itinéraire qu'il pensait initialement prendre. Ainsi, il est intéressant de pouvoir exécuter une simulation sur l'ensemble du réseau, car des décisions prises à l'échelle microscopique par des agents à un endroit et un moment donné peuvent avoir une influence globale sur l'état du réseau à un tout autre endroit plus tard dans la simulation.

Cependant, la simulation du nombre réel de voyageurs d'une grande ville (plusieurs millions de voyageurs) nécessite d'une part une puissance de calcul considérable et d'autre part une architecture permettant la distribution des traitements sur un grand nombre d'hôtes. La version actuelle de SM4T, comme la majorité des simulateurs de mobilité de voyageurs actuels, ne permet pas une telle distribution. Cela induit une limite quant au nombre de voyageurs, de moyens de transports et à la taille des réseaux considérés. Par conséquent, la prévision des effets des réglementations et des stratégies d'information sur de grands réseaux en présence de voyageurs connectés et informés en temps réel devient très difficile. Notre principal objectif est donc de tester le passage à l'échelle de ce type de simulateur. A cet effet, nous cherchons à partager la simulation de manière à répartir équitablement la charge de travail entre plusieurs serveurs. Nous souhaitons pour cela proposer des schémas génériques de distribution qui pourront être appliqués pour passer à l'échelle les simulateurs existants. Nous avons ainsi développé un simulateur de trafic donc le fonctionnement est très simple pour se concentrer exclusivement sur la problématique de la distribution.

La suite de cet article est structurée comme suit. Nous présentons tout d'abord dans la section 2 les précédentes propositions concernant la simulation de mobilité de voyageurs ainsi que les plates-formes multi-agents distribuées existantes. La section 3 présente une définition formelle de notre environnement multi-agent. Dans la section 4, nous décrivons deux méthodes permettant de distribuer le simulateur sur plusieurs hôtes. Enfin, la section 5 fournit une comparaison de ces méthodes ainsi qu'une description de notre configuration expérimentale. Nous concluons et présentons les problématiques sur lesquelles nous travaillons actuellement en section 6.

2 État de l'art

Il existe plusieurs simulateurs multi-agents de mobilité des voyageurs. Par exemple, MATSim [18] est une plate-forme bien connue de micro-simulation de mobilité. Toutefois, les entités mobiles dans MATSim sont passives et leur état est modifié par des modules centraux, ce qui limite sa flexibilité et sa capacité à intégrer de nouveaux types d'agents (proactifs). Transims [19] simule des voyages multimodaux et évalue les impacts des changements de politique dans les caractéristiques du trafic tandis que Miro [8] reproduit les dynamiques urbaines d'une ville française et propose un prototype de simulation multi-agent capable de tester la planification de scénarios en précisant les comportements des individus. AgentPolis [15] est également une plate-forme multi-agent pour

le transport multimodal. SUMO [6] et VISSIM [12] sont des simulateurs microscopiques largement utilisés et principalement axés sur la simulation de trafic. Cependant, aucune de ces propositions ne considère le problème de la distribution. A notre connaissance, aucun patron ni méthode de distribution qui soit spécifique aux simulateurs de trafic multi-agent, prenant en compte ses caractéristiques propres, telles que la présence d'un environnement physique, n'a été proposée.

Certaines plates-formes multi-agents polyvalentes ont été spécifiquement développées ces dernières années pour la simulation à grandes échelles. RepastHPC [10], une version distribuée de Repast Simphony [21], utilise les mêmes concepts de projections et de contextes que Repast et les adapte pour les environnements distribués. Pandora [1] est proche de RepastHPC et génère automatiquement le code nécessaire pour les communications interserveurs. GridABM [14] est basé sur Repast Simphony mais considère une autre approche et propose au programmeur des *templates* généraux qui peuvent être adaptés à la topologie des communications de son application. FLAMME [9] permet au programmeur de générer des simulations HPC (High Performance Computing) à partir d'automates à états finis.

Cependant, ces plates-formes distribuées n'offrent pas de contrôle précis sur la façon dont sont effectuées les communications entre les hôtes. En effet, la couche de communication est transparente pour le programmeur, ce qui rend plus facile pour lui la tâche de développer des simulations distribuées, mais ne lui permet pas d'optimiser les dites communications. La meilleure manière de gérer les communications dépend de l'application et l'utilisation de plates-formes générales pour un simulateur de mobilité ne produit pas de résultats optimaux, puisqu'il existe très souvent des connaissances et des contraintes propres à l'application qui peuvent en optimiser la distribution.

Des travaux plus théoriques étudient des méthodes générales pour résoudre ce problème. Dans [17] et [23], les auteurs proposent d'assouplir certaines contraintes de synchronisation pour obtenir un meilleur passage à l'échelle, en réduisant le temps passé par les hôtes à s'attendre les uns les autres. Cet assouplissement implique nécessairement une perte de précision qui n'est pas viable dans le cas d'un simulateur de trafic. Dans [20] et [22], les auteurs discutent des questions liées à la simulation multi-agent dans un environnement virtuel distribué. Ces deux travaux décrivent des méthodes permettant de partitionner l'environnement virtuel en plusieurs régions afin de paralléliser l'exécution de la simulation. Dans cet article, nous adaptons les approches proposées par [22] pour notre problème qui présente une structure de graphe. Ces méthodes générales, dont nous testerons l'efficacité, pourront être appliquées pour distribuer les simulations de mobilité de voyageurs de la litérature.

3 L'environnement multi-agent

3.1 Le modèle

L'environnement multi-agent d'une simulation de mobilité est constitué du réseau de transport dans lequel les agents voyageurs évoluent. Nous modélisons ce réseau de transport avec un graphe orienté $G(V, E)$ où $E = \{e_1, ..., e_n\}$ est un ensemble d'arcs représentant les routes et $V = \{v_1, ..., v_n\}$ est un ensemble de sommets représentant les intersections. Un ensemble d'agents A se déplace dans ce réseau depuis des origines vers des destinations en essayant de minimiser leur temps de parcours. Le temps de parcours d'un arc au temps t dépend du nombre d'agents présents sur cet arc à ce moment t.

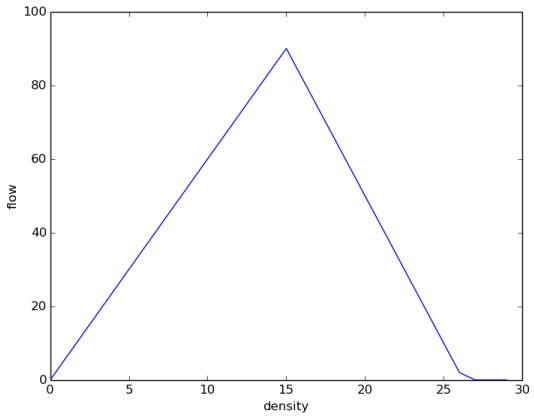


FIGURE 2 – Diagramme fondamental avec $\alpha = 6$, $\beta = 8$ et $k_c = 15$.

Pour le calculer, nous utilisons le diagramme fondamental triangulaire du trafic qui donne une relation entre le flux q (véhicules/heure) et la densité k (véhicules/km). Le diagramme fondamental suggère que si l'on dépasse une densité critique de véhicules k_c, plus il y a de véhicules empruntant une route, plus ces véhicules seront

ralentis. Voici l'équation que nous utilisons pour modéliser ce phénomène :

$$q = \begin{cases} \alpha k & \text{if} \quad k \leq k_c \\ -\beta(k - k_c) + \alpha k_c & \text{if} \quad k > k_c \end{cases} \qquad (1)$$

Cette équation est paramétrée par α la vitesse en régime fluide, β la vitesse de propagation de la congestion et k_c la densité critique. Comme $v = \frac{q}{k}$ on a :

$$v = \begin{cases} \alpha & \text{if} \quad k \leq k_c \\ \frac{-\beta(k-k_c)+\alpha k_c}{k} & \text{if} \quad k > k_c \end{cases} \qquad (2)$$

Ainsi nous pouvons définir une fonction de coût qui retourne un temps de parcours par unité de distance $(1/v)$ en fonction du nombre d'agents présents sur un arc.

$$cost(|A_e|) = \begin{cases} \frac{1}{\frac{\alpha}{|A_e|}} & \text{if} \quad |A_e| \leq k_c \\ \frac{|A_e|}{-\beta(|A_e|-k_c)+\alpha k_c} & \text{if} \quad |A_e| > k_c \end{cases} \qquad (3)$$

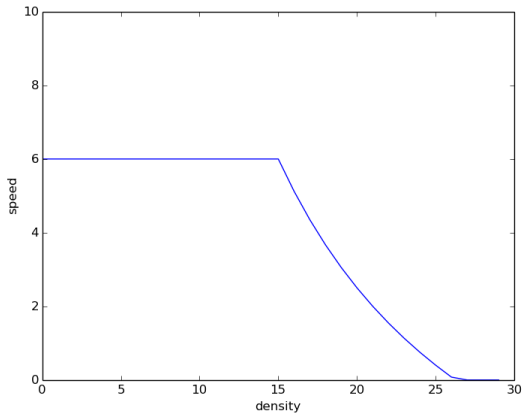


FIGURE 3 – Vitesse en fonction de la densité.

Les arcs et les sommets sont tous deux valués par des poids positifs évoluant dynamiquement avec le nombre d'agents présents. Avec A_v l'ensemble d'agents sur un sommet v et A_e l'ensemble d'agents présent sur un arc e, nous avons $|v| = |A_v|$ qui représente le poids d'un somment et $|e| = |A_e|$ le poids d'un arc. $|V| = \sum_{v \in V} |v|$ est le poids d'un sous-ensemble de sommets. De la même manière, si $|e|$ désigne le poids d'un arc, $|E| = \sum_{e \in E} |e|$ est le poids d'un sous ensemble d'arcs.

3.2 Le simulateur

Nous avons développé un simulateur de référence où chaque agent représente un voyageur évoluant dans un environnement multi-agent comme décrit dans le paragraphe précédent. Les agents sont placés sur le réseau de transport au début de la simulation avec un sommet d'origine et un sommet de destination choisis de manière aléatoire. Ils calculent leur plus court chemin sur la base de l'état actuel du réseau avant de commencer à voyager. Ils recalculent leurs itinéraires à chaque fois qu'ils atteignent un sommet, afin de vérifier s'il existe un nouveau chemin plus court, suivant la dynamique du réseau. A chaque pas de temps de la simulation, si les agents sont actuellement sur un arc, ils avancent sur cet arc autant que possible. La distance parcourue dépend du nombre total d'agent présents sur l'arc et est calculée grâce au diagramme fondamental comme décrit dans la section précédente. Lorsqu'un agent a atteint sa destination, il est supprimé du système. La simulation se termine lorsque tous les voyageurs ont atteint leurs destinations ou quand un certain temps (paramètre) est écoulé.

Ce simulateur garde les propriétés générales de tout simulateur de trafic multi-agent : des agents évoluant dans un graphe et cherchant à minimiser leur temps de parcours. Cependant, il n'est pas aussi complexe que d'autres simulateurs de la littérature dont la vocation est d'obtenir une simulation réaliste en termes de trafic généré. En effet, notre objectif est de ne représenter que les caractéristiques partagées par les simulateurs de trafic qui influent sur la complexité et l'efficacité du processus de distribution.

4 Approches proposées

Pour lancer notre simulation à l'échelle d'une ville, nous avons besoin d'une capacité de mémoire et d'une puissance de calcul importante. C'est pourquoi nous cherchons à la déployer sur plusieurs hôtes. Une telle simulation s'exécutant sur un cluster est typiquement SPMD (*Simple Program Multiple Data*), c'est à dire que tous les processeurs, reliés par un réseau, exécutent le même programme, mais ne possèdent qu'une partie des données du programme dans leurs mémoires privées. Les communications sont explicitement déclarées par le programmeur. L'avantage de cette approche est sa grande évolutivité ; il peut être mis en oeuvre sur la plupart des architectures parallèles et nous

pouvons déployer le même simulation sur des systèmes plus puissants si nécessaire.

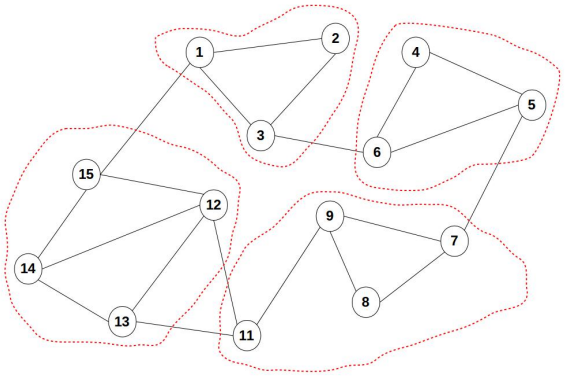
Dans les simulations de mobilité, les agents voyageurs se déplacent sur un réseau de transport. Pour distribuer ces simulations, nous devons diviser efficacement la charge de travail entre les hôtes disponibles. Dans notre modèle, la charge de travail principale est générée par le calcul du plus court chemin. Le chemin d'un agent doit être recalculé chaque fois que cet agent atteint une nouvelle intersection (parce que les temps de déplacement évoluent dynamiquement). Ainsi, un agent peut être considéré comme une unité de travail. Par conséquent, afin de distribuer ce modèle, nous devons partager le plus uniformément possible les agents entre les hôtes. Pour ce faire, nous pouvons distribuer soit l'environnement, soit les agents.

4.1 Distribution des agents

Une première approche à envisager est de diviser l'ensemble des agents en k parts égales (avec k le nombre de serveurs disponibles), de distribuer chaque sous-ensemble sur un serveur et d'exécuter la simulation. Comme les temps de parcours dépendent du nombre d'agents sur chaque arc, tous les agents ont besoin de savoir à chaque étape le nombre d'agents présents sur chacun des arcs pour calculer leur plus court chemin. Dans notre implémentation, à chaque pas de temps, si un agent géré par un serveur quitte un arc ou arrive sur un arc, ce serveur communique l'information à tous les autres serveurs. Ainsi, à n'importe quel pas de temps, tous les serveurs doivent connaître l'état de l'ensemble du réseau. Il s'agit de la seule communication nécessaires à la simulation pour cette approche. En effet, les agents ne sont pas mobiles : ils évoluent pendant tout leur cycle de vie au sein d'un même hôte. En revanche, leur avancement sur le réseau dépend des mouvements des autres agents, gérés par d'autres hôtes. D'où la communication continue entre serveurs pour informer de la dynamique locale de chacun d'eux. Ainsi le coût de communication total est de $k.|E|.I$, avec I représentant la taille d'un entier [1].

4.2 Distribution de l'environnement

Le second patron de distribution s'efforce de garder sur le même serveur les agents qui sont géographiquement proches sur le réseau. Au lieu de distribuer les agents, nous distribuons donc les sommets et les arcs sortants (et donc les agents situés sur ces sommets et ces arcs) de sorte que les agents qui sont situés au même endroit soient sur le même serveur.



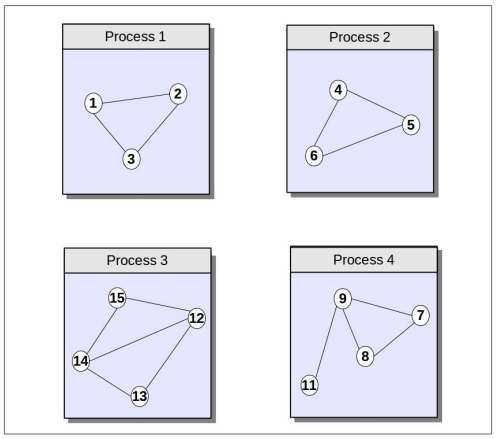


FIGURE 4 – Le graphe est partitionné et chacune des parties est distribuée entre les processus disponibles.

Chaque serveur connaît uniquement ce qui se passe sur la partie du graphe qu'il gère. Ainsi, à chaque pas de temps, avant que les agents n'agissent, tous les serveurs doivent se synchroniser avec les autres serveurs. Il y a maintenant deux types de communications : les serveurs doivent communiquer le poids de leurs arcs (le nombre d'agents présents sur ceux ci) et, quand un agent se déplace sur un sommet qui n'est pas sur son serveur actuel, il doit être transféré vers le serveur gérant ce sommet. Soit C le coût de communication des arcs et M le coût de migration des agents. A chaque étape, le coût global des communications est donnée par $T = C + M$. Le coût des communications du poids des arcs est donné par : $C = |E| \times I$. Un agent peut être

1. L'entier en question renseigne le nombre d'agents quittant ou arrivant sur un arc

codé avec trois entiers (ID, son emplacement actuel et sa destination). Soit n le nombre de migrations pour un pas de temps. Ainsi, le coût de la migration des agents est : $M = 3.I.n$. Il y a en moyenne $|A|/|E|$ agents par sommet. Donc, avec E_c l'ensemble des arcs, nous avons en moyenne $n = |A|/|E| \times |E_c|$. Ainsi T = $I(|E|+3|A|/|E| \times |E_c|)$. Ainsi, moins il y a d'arcs entre deux serveurs et plus le coût de communication est bas.

Pour que la méthode de distribution de l'environnement soit efficace, il faut répartir les sommets dans k ensembles disjoints de sorte que chaque ensemble ait approximativement le même nombre de sommets et que le poids de coupe - le poids total des arcs coupés par la partition - soit minimal. Ce problème est connu sous le nom $(k, 1+\epsilon)$-*balanced partitioning problem*, dont l'objectif est de trouver une collection de sous-ensembles disjoints $V_1, \ldots, V_k$ recouvrant V, c'est à dire $V = V_1 \cup ... \cup V_k$ tel que chaque partie contient au plus $(1 + \epsilon)\frac{|A|}{k}$ et $|E_c|$ soit minimisée.

Algorithme 1 Algorithme *Differential Greedy* modifié

Entrées: Graphe $G = (V, E)$, taille k de la partition
Sorties: Partition P
$P \leftarrow P_0, ..., P_{k-1}$
$V' \leftarrow V$
Pour $p \in [O, k-1]$ **faire**
  $v \leftarrow$ un sommet aléatoire de V'
  $P_p \leftarrow v$
  $V' \leftarrow V' \setminus v$
fin pour
Tant que $|V'| > 0$ **faire**
  $p \leftarrow$ indice de la partition la plus légère
  $m = min_{v \in V'}(1 + \epsilon)$(nombre de v voisins $\in P_p$) − (nombre de v voisins $\notin P_p$)
  mv = un sommet aléatoire de $v \in V'|(1 + \epsilon)$(nombre de v voisins $\in P_p$) − (nombre de v voisins $\notin P_p$) $= m$
  $P_p \leftarrow P_p \cup mv$
  $V' \leftarrow V' \setminus mv$
Fin tant que
Retourner P

Le problème du partitionnement de graphe a été largement étudié dans la littérature. Comme démontré dans [7], il s'agit d'un problème NP-complet. C'est à dire que la tentative de trouver une solution optimale avec, par exemple, un programme en nombre entier n'est pas une option pour des graphes de grande taille. C'est pourquoi certaines heuristiques ont été proposées pour résoudre ce problème dans un temps raisonnable. Outre la méthode spectrale [11] qui a été beaucoup utilisée dans le passé, des approches constructives comme les algorithmes Min-Max [5] et DG [13] se sont révélées efficaces pour diviser de grands graphes. Plus récemment, la méthode de partitionnement multi-niveau, originellement créée pour améliorer les techniques existantes [4] a été reconnue comme étant une méthode très puissante qui offre une vision plus globale des graphes que les techniques traditionnelles. Comme la complexité du problème de partitionnement dépend de la taille du graphe, l'idée simple du partitionnement multi-niveau est de regrouper les sommets et de travailler avec des groupes de sommets plutôt qu'avec des sommets indépendants. La partition multi-niveau a été formalisée dans un cadre générique par Walshaw dans [24]. Nous distribuons notre réseau grâce à un partitionnement multi-niveau utilisant une version légèrement modifiée de l'algorithme DG (Algorithm 1). Nous avons modifié cet algorithme pour pouvoir l'utiliser avec des sommets pondérés et produire des partitions plus connectées.

5 Expérimentations

5.1 Implémentation

Pour tester l'efficacité de ces approches, nous avons implémenté notre modèle et l'avons déployé sur un cluster réel. Nous avons choisi Python comme langage de développement pour son efficacité dans le prototypage rapide. Python est un langage portable, mature, disposant de nombreuses bibliothèques scientifiques éprouvées. Il est avec C et Fortran l'un des langages les plus utilisés pour le calcul haute performance [16]. Comme le but de nos travaux est de fournir une étude de l'efficacité relative de différentes méthodes de distribution, et non une performance absolue, le choix de Python nous semble pertinent.

Pour les communications inter-processus, nous utilisons MPI (Message Passing Interface), qui est le standard *de facto* pour le calcul parallèle avec une importante communauté d'utilisateurs. MPI offre un modèle de communication simple entre les différents processus d'un programme et a de nombreuses implémentations efficaces qui s'exécutent sur une grande variété de machines [2]

2. De plus, nous bénéficions de MPI4PY, qui est une interface efficace qui permet d'utiliser MPI avec Python.

5.2 Résultats

Nous avons exécuté les simulations distribuées sur le cluster de l'Université de Cardiff. Pour nos tests, nous avons utilisé huit hôtes sous CentOS Linux (noyau 2.6.32-220) sur un processeur Intel Xeon E5-2620 CPU (12 cœurs à 2 GHz) avec 32 Go de mémoire. Nous avons exécuté la simulation sur trois configurations : la première est une version séquentielle du programme s'exécutant sur un seul hôte (conf1), la seconde est une version distribuée sur les huit hôtes (conf2), et la dernière est exécutée sur les huit hôtes utilisant les 12 cœurs de chacun d'entre eux (conf3). La simulation est effectuée pour 100 pas de temps sur un réseau invariant d'échelle de 200 noeuds (figure 5) généré avec le modèle Barabasi-Albert [3]. Nous avons comparé les deux méthodes de distribution (distributions des agents et distribution de l'environnement) sur les différentes configurations avec un nombre croissant d'agents (de 1000 à 40000).

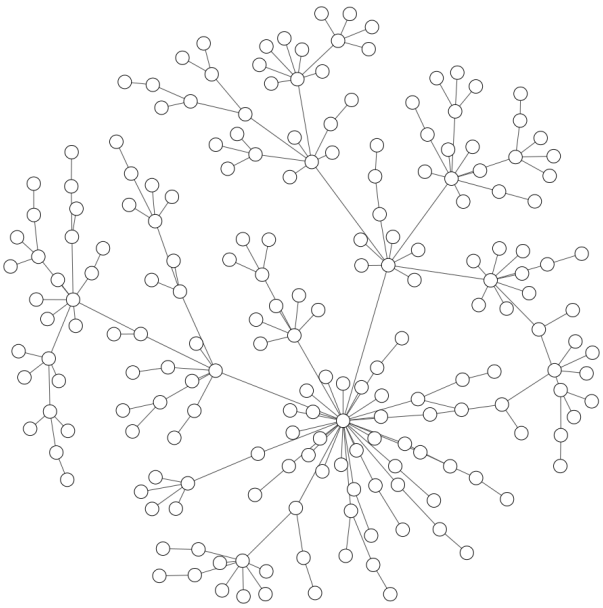


FIGURE 5 – Un graphe de 200 sommets généré avec le modèle de Barabasi-Albert.

Les résultats (table 1) montrent que la distribution des agents est plus efficace que la distribution de l'environnement avec le modèle proposé [3] (figure 6). En effet, par souci

3. Les temps de calcul ne sont pas strictement croissant avec le nombre d'agents pour la distribution de l'environnement. Cela est probablement dû au caractère aléatoire des origines/destinations des agents. La simulation peut en effet parfois être plus complexe alors qu'elle comporte moins d'agents.

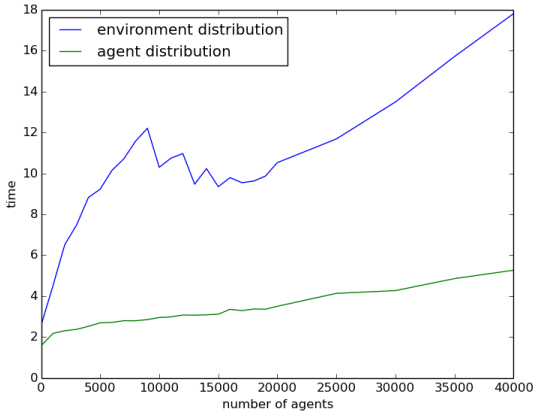


FIGURE 6 – Comparaison des temps d'exécution entre les différentes méthodes de distribution sur conf3.

de généricité, nous n'avons pas défini d'interactions locales entre les agents dans notre simulateur de référence actuel. Par conséquent, nous sommes dans un cas idéal pour une distribution fondée sur une répartition des agents, puisque le nombre de communications inter-serveurs est limité. Avec l'implémentation future d'interactions locales (en intégrant un modèle de poursuite ainsi que des communications inter-véhicules), la distribution de l'environnement sera avantagée grâce à la cohabitation sur le même serveur d'agents physiquement proches. En outre, la distribution de l'environnement ne bénéficie pas à l'heure actuelle de mécanisme d'équilibrage de charge dynamique. Si un nombre important d'agents est concentré sur la même partie du réseau, ils seront sur le même serveur. Ce serveur prendra plus de temps pour calculer tous les plus courts chemins, et tous les autres serveurs devront l'attendre pour passer au pas de temps suivant. Ainsi, un serveur surchargé peut ralentir l'ensemble de la simulation.

Les améliorations de temps d'exécution mesurées entre l'exécution séquentielle et les deux méthodes de distribution sur conf3 sont indiquées sur les figures 7 et 8. L'amélioration mesure combien de fois la simulation est plus rapide sur conf3 (96 cœurs) que sur conf1. Comme nous pouvons le constater sur ces figures, pour 40.000 agents, la simulation est 8 fois plus rapide avec la distribution de l'environnement et 28 fois plus rapide avec la distribution des agents. Ces deux méthodes améliorent largement le temps d'exécution de notre simulateur de mobilité multi-agent. Comme ex-

number of agents	1000	5000	10000	20000	30000	40000
conf1 (1 core)	10	27	43	67	104	140
conf2 agent distribution (12 cores)	3	6	8	12	17	23
conf3 agent distribution (96 cores)	2	3	3	4	4	5
conf2 environment distribution (12 cores)	6	13	17	22	31	41
conf3 environment distribution (96 cores)	5	10	11	11	15	17

TABLE 1 – Temps d'exécution (en secondes) pour une simulation de 100 pas de temps sur un réseau invariant d'échelle de 200 noeuds.

pliqué ci-dessus, la méthode de distribution des agents montre de meilleurs résultats en raison de la généricité et de la relative simplicité du modèle actuel. Nous pensons que l'ajout de communications inter-agents fera que la différence de performance entre les deux méthodes devrait largement diminuer, voire s'inverser.

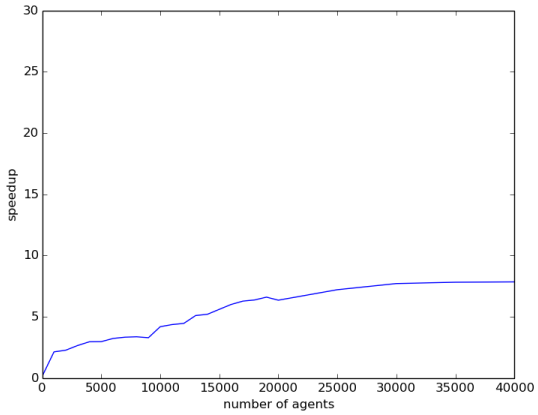


FIGURE 7 – Amélioration de temps d'exécution entre conf1 et conf3 pour la distribution de l'environnement.

6 Conclusions et perspectives

Dans cet article, nous avons présenté deux méthodes de distribution sur plusieurs hôtes d'un simulateur de mobilité multi-agent. Les deux méthodes sont efficaces et améliorent le passage à l'échelle du simulateur. Avec notre modèle de simulation actuel, la méthode de distribution des agents est plus efficace que la méthode de distribution de l'environnement. Notre modèle de simulation est très générique, nos propositions et conclusions sont applicables à tous les simulateurs de mobilité actuels.

Nos travaux futurs porteront sur deux aspects. Nous aborderons tout d'abord des modèles de simulation de mobilité plus spécifiques, où

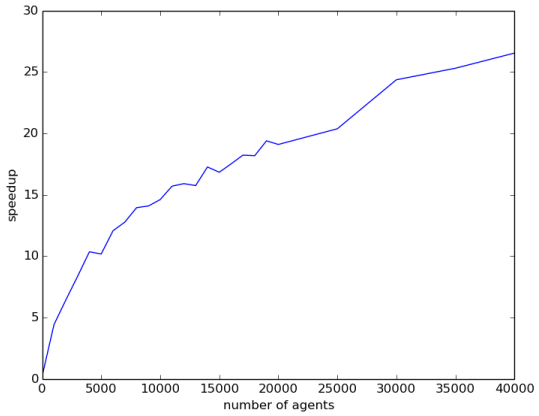


FIGURE 8 – Amélioration de temps d'exécution entre conf1 et conf3 pour la distribution des agents.

tous les types de communications sont présents (locaux, globaux, communautaires, etc.). Nos agents qui se contentaient ici du calcul de plus court chemin auront alors a gérer des situations nécessitant plus d'interactions locales (par exemple pour la négociation de l'ordre de passage dans les intersections). Nous prévoyons que la méthode de distribution de l'environnement affiche de meilleures performances comparatives qu'avec le modèle actuel. De plus, la distribution de l'environnement se fait actuellement de manière statique au début de la simulation et nous pensons que les temps de calcul pourrait être raccourcis en ajoutant des mécanismes d'équilibrage de charge dynamique. Ce point d'amélioration sera notre deuxième axe de travail.

Références

[1] E.S. Angelotti, E.E. Scalabrin, and B.C. Avila. PANDORA : a multi-agent system using paraconsistent logic. In *Fourth International Conference on Computational Intelligence and Multimedia Applications, 2001. ICCIMA 2001. Proceedings*, pages 352–356, 2001.

[2] Fabien Badeig, Flavien Balbo, Gérard Scemama, and Mahdi Zargayouna. Agent-based coordination model for designing transportation applications. In *Intelligent Transportation Systems, 2008. ITSC 2008. 11th International IEEE Conference on*, pages 402–407. IEEE, 2008.

[3] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439) :509–512, October 1999.

[4] Stephen T. Barnard and Horst D. Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency : Pract. Exper.*, 6(2) :101–117, April 1994.

[5] Roberto Battiti and Alan Bertossi. Greedy, prohibition, and reactive heuristics for graph partitioning. *IEEE Transactions on Computers*, 48 :361–385, 1998.

[6] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. SUMO - simulation of urban MObility - an overview. In *SIMUL 2011, The Third International Conference on Advances in System Simulation*, pages 55–60, 2011.

[7] Thang Nguyen Bui and Curt Jones. Finding good approximate vertex and edge partitions is NP-hard. *Information Processing Letters*, 42(3) :153–159, May 1992.

[8] Sébastien Chipeaux, Fabrice Bouquet, Christophe Lang, and Nicolas Marilleau. Modelling of complex systems with AML as realized in MIRO project. In *LAFLang 2011, workshop of the Int. Conf. WI/IAT (Web Intelligence and Intelligent Agent Technology)*, pages 159–162, Lyon, France, 2011. IEEE Computer Society.

[9] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, and C. Greenough. Exploitation of high performance computing in the FLAME agent-based simulation framework. In *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS)*, pages 538–545, 2012.

[10] Nicholson Collier and Michael North. Repast HPC : A platform for large-scale agent-based modeling. In Werner Dubitzky, Krzysztof Kurowski, and Bernhard Schott, editors, *Large-Scale Computing*, pages 81–109. John Wiley & Sons, Inc., 2011.

[11] W. E. Donath and A. J. Hoffman. Lower bounds for the partitioning of graphs. *IBM J. Res. Dev.*, 17(5) :420–425, September 1973.

[12] Martin Fellendorf and Peter Vortisch. Microscopic traffic flow simulator vissim. In *Fundamentals of Traffic Simulation*, pages 63–93. Springer, 2010.

[13] C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. In *19th Conference on Design Automation, 1982*, pages 175–181, June 1982.

[14] Laszlo Gulyas, Gabor Szemes, George Kampis, and Walter de Back. A modeler-friendly API for ABM partitioning. In *ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 219–226, 2009.

[15] Michal Jakob, Zbynek Moler, Antonín Komenda, Zhengyu Yin, Albert Xin Jiang, Matthew Paul Johnson, Michal Pechoucek, and Milind Tambe. Agentpolis : towards a platform for fully agent-based modeling of multi-modal transportation (demonstration). In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, pages 1501–1502, 2012.

[16] Hans Petter Langtangen and Xing Cai. On the efficiency of python for high-performance computing. In *Modeling, Simulation and Optimization of Complex Processes*, pages 337–357. Springer Berlin Heidelberg, January 2008.

[17] D. Mengistu and M. v Lowis. An algorithm for optimistic distributed simulations. In *Modelling, Simulation, and Identification / 658 : Power and Energy Systems / 660, 661, 662*. ACTA Press, 2011.

[18] Maciejewski Michal and Nagel Kai. Towards multi-agent simulation of the dynamic vehicle routing problem in mat-

sim. In *Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics - Volume Part II*, PPAM'11, pages 551–560, Berlin, Heidelberg, 2012. Springer-Verlag.

[19] Kai Nagel and Marcus Rickert. Parallel implementation of the transims micro-simulation. *Parallel Computing*, 27(12) :1611–1639, 2001.

[20] Beatrice Ng, Antonio Si, Rynson W.H. Lau, and Frederick W.B. Li. A multi-server architecture for distributed virtual walkthrough. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST '02, pages 163–170. ACM, 2002.

[21] Michael J. North, Nicholson T. Collier, Jonathan Ozik, Eric R. Tatara, Charles M. Macal, Mark Bragen, and Pam Sydelko. Complex adaptive systems modeling with repast simphony. *Complex Adaptive Systems Modeling*, 1(1) :3, 2013-03-13.

[22] Omar Rihawi, Yann Secq, and Philippe Mathieu. Effective distribution of large scale situated agent-based simulations. In *ICAART 2014 6th International Conference on Agents and Artificial Intelligence*, volume 1, pages 312–319. SCITEPRESS Digital Library, 2014.

[23] Matthias Scheutz and Paul Schermerhorn. Adaptive algorithms for the dynamic distribution and parallel execution of agent-based models. *Journal of Parallel and Distributed Computing*, 66(8) :1037–1051, 2006.

[24] Chris Walshaw. Multilevel refinement for combinatorial optimisation : Boosting metaheuristic performance. In *Hybrid Metaheuristics*, number 114 in Studies in Computational Intelligence, pages 261–289. Springer Berlin Heidelberg, January 2008.

[25] Mahdi Zargayouna, Besma Zeddini, Gérard Scemama, and Amine Othman. Simulating the impact of future internet on multimodal mobility. In *The 11th ACS/IEEE International Conference on Computer Systems and Applications AICCSA'2014*. IEEE Computer Society, 2014.

[26] Mahdi Zargayouna, Besma Zeddini, Gérard Scemama, and Amine Othman. Agent-based simulator for travelers multimodal mobility. *Frontiers in Artificial Intelligence and Applications*, 252 :pp 81–90, January 2013.