Politiques de synchronisation dans les systèmes multi-agents distribués parallèles

Paul Breugnot, A Rousset, B Herrmann, C Lang, L Philippe

Politiques de synchronisation dans les systèmes multi-agents distribués parallèles

A. Rousset[1], P. Breugnot[2], B. Herrmann[2], C. Lang[2] et L. Philippe[2]

[1] Faculté des Sciences, de la Technologie et de Médecine (FSTM),
Université du Luxembourg, Luxembourg
[2] Institut Femto-ST, Unversité de Bourgogne-Franche-Comté/CNRS, France

christophe.lang@univ-fcomte.fr

Résumé

Parmi les méthodes de modélisation/simulation, les systèmes multi-agents présentent un intérêt particulier pour simuler les systèmes complexes. Lorsque la taille des modèles croît, le recours aux systèmes multi-agents parallèles est nécessaire mais pose de nombreux problèmes. Dans cet article, nous nous intéressons à l'impact de la synchronisation sur la définition des modèles et leur exécution. Nous mettons en évidence des problématiques de synchronisation à travers des instances de modèles puis nous analysons expérimentalement l'impact des politiques de synchronisation sur des exécutions de grande taille.

Mots-clés

multi-agent simulation, parallélisme, MAS, High Performance Computing, synchronisation

Abstract

Among simulation or modelisation methods, multi-agent systems are interesting candidate to simulate complex systems. As the size of the models increases, the use of parallel multi-agent systems is mandatory but comes with many issues. In this article, we are interested in the impact of synchronization on models design and on their execution. We highlight synchronization problems through model instances then we experimentally analyze the impact of synchronization on large scale testcases.

Keywords

Multi-agent simulation, Parallelism, High Performance Computing, Synchronization

1 Introduction

La simulation numérique est devenue le troisième pilier de la science en tant qu'étape de validation de la théorie, déterminante pour le passage à l'expérimentation. Elle vise à virtualiser le monde réel, à en reproduire les comportements, par exemple pour explorer son évolution dans différentes configurations ou pour comprendre comment le contrôler. Dans les systèmes complexes, plusieurs phénomènes peuvent ainsi être étudiés simultanément mais les comportements sont souvent trop élaborés et interdépendants pour pouvoir être modélisés par une loi unique. Les systèmes multi-agents sont alors souvent utilisés pour modéliser les comportements dynamiques des entités qui composent le système car ils reposent sur une description algorithmique simple d'agents qui interagissent entre eux. De nombreuses plates-formes [13, 7, 12] proposent un environnement de développement pour de tels modèles.

La qualité d'une simulation dépend bien souvent de la taille et de la précision du modèle. Or l'accroissement de la taille du modèle et de sa précision entraîne, de fait, une augmentation du nombre de calculs réalisés et rend nécessaire le recours à des exécutions parallèles, voire à l'utilisation de moyens de calcul haute performance (HPC : High Performance Computing). Si la simulation sur un seul ordinateur est souvent complexe, l'exécution distribuée parallèle d'une simulation est un vrai enjeu car elle pose de nombreux problèmes comme la distribution de l'environnement, la communication entre les instances parallèles de la plateforme, etc. Il existe des plateformes multi-agents (Parallel and Distributed MAS ou PDMAS) qui prennent en charge tout ou partie de l'exécution distribuée. Plusieurs instances, ou processus, de la plateforme coopèrent pour faciliter la mise en œuvre d'un modèle sur un ensemble d'ordinateurs ou au sein d'un cluster mais la synchronisation reste l'un des points clefs pour l'exécution efficace d'une simulation multi-agents parallèle du fait des nombreux échanges et dépendances temporelles qu'elle induit.

Nous nous intéressons dans cet article aux problèmes posés par la synchronisation au sein de simulations multi-agents distribuées parallèles, en visant plus particulièrement les exécutions à large échelle. Comme les plates-formes multi-agents fonctionnent fréquemment par pas de temps, il est nécessaire de s'interroger sur la manière d'échanger les données entre les processus distribués d'une même simulation au regard de ce mode de fonctionnement.

Les contributions proposées dans cet article sont, d'une part, la définition de politiques de synchronisation qui peuvent être utilisées au sein de simulations multi-agents parallèles et leur mise en évidence à travers des instances de modèles, et d'autre part, l'analyse expérimentale de l'impact des politiques de synchronisation sur des exécutions de grande

taille, jusqu'à 512 cœurs, dans les systèmes multi-agents parallèles et distribués.
L'article est organisé comme suit. Dans la section 2 nous présentons un état de l'art sur les systèmes multi-agents parallèles et la synchronisation. Nous proposons une étude des problématiques de synchronisation dans les systèmes multi-agents parallèles distribués dans la section 3. Puis, dans la section 4, avant de conclure, nous présentons une étude expérimentale de l'impact de différentes politiques de synchronisation sur l'exécution de trois modèles agents.

2 PDMAS et synchronisation

Par nature les agents interagissent entre eux, soit directement en consultant les données d'autres agents, soit indirectement, à travers les modifications réalisées sur l'environnement ou par échange de messages. Dans un PDMAS, les agents du modèle sont distribués entre les instances de la plateforme, ou processus, qui prennent en charge leur animation. De la même manière les données de l'environnement sont distribuées entre les processus. Un agent peut avoir à interagir avec des agents ou utiliser des données de l'environnement situés sur un autre processus. Ces accès nécessitent donc la mise en place d'une synchronisation entre les processus pour maintenir un état cohérent et permettre aux agents d'accéder à la valeur à jour des données avec l'objectif que la simulation parallèle donne le même résultat que la simulation séquentielle.
Les processus étant distribués, la synchronisation des données est réalisée sous la forme de communications. L'accès à ces données est alors plus coûteux qu'un simple accès local et il est important d'implémenter la synchronisation au plus juste, de manière à limiter les surcoûts. Cette implémentation suppose de caractériser les propriétés attendues. Dans la littérature, la caractérisation de la synchronisation de modèles parallèles repose sur le mode de dépendance de données et son respect de la causalité.

2.1 Modes de dépendance de données

On distingue deux modes de gestion de la dépendance de données dans les simulations numériques : (i) le mode *ghost* qui utilise les données calculées au pas de temps précédent (le *ghost*) et (ii) le mode *non ghost* qui n'utilise qu'une seule instance des données, dans laquelle les résultats des calculs sont directement reportés. Le *ghost* n'est accessible qu'en lecture alors que, sans *ghost*, les données sont accessibles en lecture et en écriture et l'ordre d'accès aux données impacte les résultats. Dans le cas des PDMAS, le modèle *ghost* permet de ne diffuser les données modifiées qu'à la fin des pas de temps. À noter que, pour limiter le coût de ces mises à jour, plusieurs PDMAS utilisent des zones de recouvrement, copies locales des données distantes limitées à la zone de perception des agents. Le mode *non ghost* nécessite la mise en place de mécanismes de mise à jour continus (au sein d'un même pas de temps) garantissant que les données utilisées lors d'interactions entre processus différents sont les données les plus à jour. Ces mises à jour doivent alors être synchronisées pour éviter les conflits. Nous analysons en 3.1 l'incidence du mode de dépendance de données sur la synchronisation dans les modèles multi-agents et nous montrons que le choix du mode est déterminé par le modèle.

2.2 Respect de la causalité

Pour optimiser l'exécution parallèle de simulations, les systèmes à événements discrets autorisent parfois les processus à s'affranchir de la causalité. Deux approches de synchronisation sont ainsi définies, l'approche conservative et l'approche optimiste. Dans l'approche conservative (ou pessimiste), lorsqu'un processus traite un événement de date T, pour respecter la causalité, il doit être sûr de ne pas utiliser de données qui pourraient être modifiées ultérieurement avec une date $T' < T$. Dans cette approche de synchronisation il est nécessaire de déterminer les événements dont les données sont à jour. Les premiers algorithmes utilisant l'approche conservative ont été proposés par Chandy et al. [2]. Cette approche limite l'exploitation du parallélisme d'un modèle.
Contrairement à l'approche conservative, la violation de la contrainte de causalité est possible avec l'approche optimiste. Dans cette approche, chaque processus traite les événements dont il a connaissance. Cette connaissance des événements à traiter étant locale, et donc partielle, elle peut impliquer l'omission de certains événements provenant d'autres processus et donc ne pas respecter la causalité [11]. Lorsque le processus reçoit une donnée dont la date est antérieure à la date du processus, un retour en arrière est effectué par des mécanismes de *Rollback* [8] qui impliquent de sauvegarder plusieurs points de récupération par processus. Ainsi Xu et al. [17] définissent le *lookahead* comme étant la durée jusqu'au prochain pas de temps auquel il faudra mettre à jour les données. Ceci laisse la possibilité de continuer un processus tant qu'il n'est pas arrivé à son *lookahead*. A noter que cette approche est difficilement généralisable à tous les modèles et qu'elle n'est pas efficace lorsque le temps d'exécution passé à effectuer des *Rollback* devient très important, un *Rollback* pouvant entraîner une réaction en chaîne de *Rollback*.
Dans le cas des PDMAS, les événements sont planifiés tous les pas de temps. L'utilisation des approches conservatives ou optimistes peut s'entendre comme le respect strict ou non de la frontière du pas de temps, comme nous l'étudions dans la section 3.

2.3 La synchronisation dans les PDMAS

Dans [9] nous avons proposé une étude des plateformes multi-agents parallèles et distribuées (PDMAS). Parmi les plateformes que nous avons évaluées, seules quatre d'entre elles permettent d'envisager une exécution sur des ressources de grande taille, de type HPC.

D-Mason. La plateforme D-Mason [5] implémente des mécanismes de synchronisation conservatifs et le mode *ghost* de gestion de dépendance des données. Pour réaliser une synchronisation conservative, chaque pas de temps est divisé en deux étapes : (1) la communication et synchronisation et (2) l'exécution de la simulation. Il y a donc une barrière de synchronisation à chaque pas de temps. Les agents d'une cellule (partition) c ne peuvent pas exécuter

le pas de temps i tant que les cellules voisines n'ont pas terminé d'exécuter le pas de temps $i-1$. A la fin d'un pas de temps, chaque cellule envoie aux cellules voisines les informations concernant les agents qui se situent dans la zone de recouvrement ou les agents qui doivent être migrés d'un processus à un autre. Pour le pas de temps i les comportements de la cellule c sont ainsi calculés à partir du *ghost* des cellules voisines.

RepastHPC. Pour gérer le partage de données, la plateforme RepastHPC propose au programmeur de faire une copie, sur les processus distants, des agents susceptibles d'y être utilisés. La synchronisation entre les processus est effectuée uniquement dans quatre cas [4] : (i) lorsqu'un processus a besoin d'une copie d'agents d'un autre processus, pour maintenir la simulation dans un état cohérent, (ii) quand un processus possède un agent copie d'un autre processus, pour mettre à jour les copies de l'agent original, (iii) lorsque les zones de recouvrement d'une grille doivent être mises à jour, à chaque fin de pas de temps, (iv) quand un agent doit être migré d'un processus à un autre. Avec les outils de la plateforme, les programmeurs ont à développer un ensemble de méthodes qui définissent les informations nécessaires devant être synchronisées dans les agents. La plateforme permet d'implémenter les deux modes de dépendances de données car la mise à jour d'un agent copie peut-être soit périodique soit à la consultation de la copie. En couplant ces modes avec les fonctions d'ordonnancement de la plateforme, il est possible d'implémenter les politiques conservatives et optimistes mais l'ensemble reste à la charge du développeur avec le risque de surcharger le modèle.

Flame. Avec Flame, tous les échanges entre les agents se font par messages, la synchronisation conservative repose donc sur celle de tableaux de messages [3]. Elle est effectuée en deux étapes : la demande de synchronisation puis l'exécution de la synchronisation. Dans un premier temps, lorsqu'un processus a terminé d'exécuter ses agents, il verrouille son tableau de messages et envoie aux autres processus une demande de synchronisation. Après cette étape, il est encore possible de faire des actions qui ne nécessitent pas l'utilisation du tableau de messages. Lorsque tous les processus ont verrouillé leurs tableaux de messages, une seconde étape d'exécution de la synchronisation est effectuée par échange de messages entre les tableaux. Après ces deux étapes, les tableaux de messages sont débloqués et la simulation se poursuit. Puisqu'il n'y a pas de modification des données entre deux synchronisations des tableaux de messages, la plateforme Flame repose sur un mode *ghost*.

Pandora. Dans la plateforme Pandora la synchronisation est conservative [1] et repose sur une grille 2D. Les données et les agents situés dans les zones de recouvrement sont copiés et envoyés aux cellules voisines à chaque pas de temps. Pour résoudre le problème de la dépendance de données, la simulation est découpée en parties, numérotées de 0 à 3. Au cours d'un pas de temps tous les processus exécutent séquentiellement chacune des parties, dans le même ordre : la partie 0, puis 1, 2 et 3. Une fois l'exécution d'une partie terminée, les zones de recouvrement sont envoyées aux cellules voisines. De cette façon, il n'y a pas de conflits de cohérence car les parties exécutées en parallèle ne sont pas adjacentes. Cette approche originale réduit les coûts de synchronisation avec un modèle en partie *ghost* et en partie *non ghost*, suivant l'emplacement des données. Cela limite tout de même l'utilisation de Pandora à des modèles à deux dimensions.

Synthèse. Les PDMAS existants utilisent donc principalement une approche conservative quant à la causalité, certains laissant cette gestion au développeur. Ceci paraît justifié car l'approche optimiste convient peu aux systèmes multi-agents qui animent les agents par pas de temps, donc de manière uniforme au sein de processus parallèles à l'inverse des systèmes à événements discrets où les processus ont une répartition temporelle des événements différente entre eux. Il est donc moins intéressant de faire avancer plus vite certains processus et une politique de répartition de la charge peut s'avérer plus efficace.
Les plateformes proposent principalement un mode *ghost*, moins lourd qu'une mise à jour systématique, qui recopie les données vers les autres processus au changement de pas de temps, lorsque tous les agents sont dans un état fixe.
Une analyse de plusieurs types de modèles permet cependant de montrer que tous n'ont pas les mêmes besoins de synchronisation. Nous proposons donc, dans la section suivante, une étude des besoins de synchronisation de modèles multi-agents et différentes politiques pouvant répondre à ces besoins.

3 Impact des synchronisations

La synchronisation est un point clé pour une exécution efficace d'une simulation multi-agents parallèle, du fait des nombreux échanges et dépendances temporelles qu'elle induit. Son impact sur la parallélisation dépend néanmoins du modèle lui-même : dans un modèle où les agents n'interagissent pas, aucune synchronisation n'est nécessaire. L'intérêt du modèle agent repose cependant justement sur la capacité des agents à interagir [6]. Notre objectif est donc d'étudier l'impact de la synchronisation sur les temps d'exécution et l'impact de politiques de synchronisation plus relâchées. En effet, dans les systèmes multi-agents, l'observation d'un phénomène s'effectue en général au niveau macroscopique et non microscopique. De ce point de vue, il est possible que, dans les simulations composées d'un grand nombre d'agents, des synchronisations erronées ou fausses se compensent et limitent ainsi l'impact d'une synchronisation relâchée. Il est donc intéressant de mettre en relation l'erreur possible avec le surcoût dû à la synchronisation. Nous proposons dans la suite différentes politiques de synchronisation et étudions leur impact.

3.1 Quand synchroniser ?

L'objectif est de garantir l'accès à des données à jour et la cohérence des actions, pour respecter les règles du modèle. Plusieurs synchronisations sont donc nécessaires, que la simulation soit en mode *ghost* ou *non ghost* : (i) à la fin

de chaque pas de temps pour permettre le passage au pas suivant et garantir que tous les processus exécutent le même pas de temps, (ii) à la migration d'un agent d'un processus à un autre pour continuer à exécuter ses comportements il est nécessaire que les processus concernés soient dans le même état, (iii) à la mise à jour des zones de recouvrement pour garder la continuité des champs de perception des agents lors de la distribution de l'environnement sur plusieurs processus.

Une exécution en mode *ghost* ne nécessite pas de gérer ces problèmes d'écriture mais elle ne peut conserver que la cohérence des modèles qui ne nécessitent pas d'écriture concurrente, c'est à dire où deux entités ne modifient pas la même donnée au cours d'un pas de temps. En effet, utiliser le mode *ghost* avec des écritures concurrentes peut conduire à une violation des règles du modèle. Par exemple, dans un modèle proie-prédateur, plusieurs prédateurs pourraient manger une même proie au cours d'un pas de temps puisque, suite à une première attaque, la mort de la proie est enregistrée dans la copie de travail et non dans la copie *ghost* qui est utilisée pour connaître l'état du système. C'est seulement au changement de pas de temps que la mort de la proie sera reportée dans la copie *ghost*. À noter que ceci est vrai, même si la simulation n'est pas parallèle.

À l'opposé, dans le mode *non ghost*, les informations sont accessibles en lecture et en écriture. Ceci nécessite alors des mécanismes de synchronisation pour les données détenues par d'autres processus ou situées dans les zones de recouvrement. Dans ce cas, les points de synchronisation précédemment définis ne sont pas suffisants car ils ne permettent pas de gérer les écritures dans les zones de recouvrement. L'accès aux données doit donc être géré pour garantir qu'aucune incohérence (ou biais) n'est injectée dans la simulation et que les informations sont à jour.

Dans les modèles agents, il existe des modèles qui nécessitent que les données soient accessibles uniquement en lecture, d'autres en lecture et en écriture. La synchronisation est donc à considérer au cas par cas. Pour illustrer ceci, nous analysons différents types de modèles dans la suite.

3.2 Analyse de modèles

Pour évaluer l'impact de la synchronisation sur les résultats d'exécution des simulations multi-agents, nous utilisons trois modèles agents (Proie-prédateur, Virus et Flocking) qui nécessitent des niveaux de synchronisation différents pour s'exécuter de manière cohérente.

Il est important de noter que le système modélisé et son implémentation ont un impact important sur la synchronisation qui doit être mise en place pour garantir la qualité des résultats obtenus par la simulation parallèle. Ceci est, en particulier, vrai si le choix d'implémentation utilise un *ghost* ou non. Ainsi, parmi les modèles suivants, nous avons choisi l'approche que nous avons le plus souvent trouvée par rapport à ce choix d'implémentation. Changer ce choix modifierait les contraintes de synchronisation et conduirait à d'autres conclusions.

Le modèle Flocking [14]. Le modèle Flocking simule le vol d'une nuée d'oiseaux afin d'étudier le comportement collectif. Le modèle est composé d'un seul type d'agent, les oiseaux qui sont localisés dans l'espace et ont une zone de perception réduite. Chaque agent oiseau a trois comportements : (i) la cohésion qui le pousse à se rapprocher des oiseaux proches, (ii) l'alignement qui le pousse à se déplacer dans la même direction que les oiseaux voisins et (iii) la séparation qui le pousse à tourner pour éviter un oiseau qui est trop près. Ces comportements déterminent la nouvelle position de l'oiseau en fonction de la position des oiseaux qui composent son voisinage. Le modèle possède plusieurs paramètres : la taille de l'environnement, la distance maximale qu'un oiseau peut parcourir par pas de temps, la durée d'un pas de temps, les taux de cohésion, d'alignement et de séparation.

Ce modèle fonctionne en mode *ghost*. Les agents calculent leur déplacement en fonction de la position des oiseaux voisins obtenue au pas de temps précédent. La mise à jour des données des zones de recouvrement à chaque pas de temps garantit que chaque agent oiseau dispose des informations correctes pour calculer son déplacement. Il n'y a donc pas de problème de concurrence d'accès sur ces données puisqu'elles sont accédées uniquement en lecture.

Le modèle Virus [15]. Le modèle Virus permet de simuler la transmission et la survie d'un virus dans une population[18]. Il est composé d'un seul type d'agent : les personnes qui sont localisées sur une grille en deux dimensions et n'ont qu'une connaissance partielle de l'environnement dans lequel ils évoluent. Les agents ont cinq comportements : (i) le vieillissement, jusqu'à ce qu'ils meurent, (ii) le déplacement de manière aléatoire sur l'environnement, (iii) l'infection des personnes de leur voisinage, (iv) la récupération qui permet à un agent infecté d'avoir une probabilité de devenir immunisé et (v) la reproduction, pour les personnes non contaminées, qui renouvelle la population. Le modèle possède plusieurs paramètres : la capacité de transport du virus, l'âge maximum, le taux de natalité, le taux de reproduction et le nombre de personnes porteuses du virus à l'initialisation du modèle.

Le modèle Virus fonctionne en mode *non ghost*. Il est donc possible que, dans un même pas de temps, un agent A, qui a été infecté, infecte à son tour un agent B. Si les deux agents ne sont pas exécutés sur le même processus, il est nécessaire de mettre à jour les données distantes. Pour finir, nous pouvons remarquer qu'un agent ne change pas son état même s'il est infecté plusieurs fois. Cette propriété, que nous appelons écriture idempotente, fait que nous n'avons pas à gérer de concurrence en écriture sur le changement de l'état de l'agent, puisque même si deux agents infectent un même agent l'ordre des deux exécutions n'a pas d'incidence sur le résultat final.

Le modèle proie-prédateur [16]. Le modèle proie-prédateur explore la stabilité des écosystèmes. Le modèle étudié possède trois types d'agents : les loups (prédateurs), les moutons (proies/prédateurs) et l'herbe (proie). Les loups et les moutons se déplacent au hasard dans l'environnement. L'herbe disparaît lorsqu'elle est mangée et repousse après un temps fixé. Chaque étape coûte de l'énergie aux loups et aux

moutons qui, lorsqu'ils n'en ont plus, meurent. L'énergie peut-être reconstituée pour un loup en mangeant un mouton et pour un mouton en mangeant de l'herbe. Pour permettre à la population de perdurer, les loups et les moutons ont une probabilité de se reproduire à chaque pas de temps. Tous les agents sont localisés sur une grille à deux dimensions et ne connaissent que leur zone de perception. A chaque pas de temps, les agents loups et moutons exécutent les quatre comportements suivants dans l'ordre donné : (i) se déplacer aléatoirement sur l'environnement, (ii) se nourrir de proies si elles se situent dans leur champ de perception, (iii) mourir s'ils n'ont plus d'énergie ou ont atteint leur durée de vie, et (iv) se reproduire. Un agent herbe apparaît aléatoirement tous les *n* pas de temps. Le modèle possède plusieurs paramètres : la taille de l'environnement, le nombre de loups, le nombre de moutons, le taux de natalité, le taux de reproduction, le gain de vie lorsqu'un prédateur mange une proie et le temps de croissance maximal des agents herbe.
Le modèle proie-prédateur fonctionne en mode *non ghost*. Dans la mesure où les prédateurs mangent les proies, ils en changent l'état, ce qui engendre une écriture dans les données de l'agent. Comme pour le modèle virus cela oblige à une synchronisation pendant le pas de temps pour prendre en compte la modification mais il est, en plus, nécessaire de gérer les écritures concurrentes. A cause du parallélisme, plusieurs prédateurs situés dans des processus différents peuvent être tentés de manger une même proie dans la zone de recouvrement. Si tous les prédateurs mangent cette proie (celle qui est dans la zone de recouvrement) alors chacun bénéficiera d'un apport en énergie et donc d'une augmentation de sa durée de vie, ce qui constitue une erreur par rapport au modèle séquentiel. Il est donc nécessaire de synchroniser tous les agents qui souhaiteraient manger une proie pour garantir qu'un seul prédateur la mangera.

Synthèse. L'analyse de ces modèles met en évidence différents besoins en termes de synchronisation. Ces besoins dépendent des caractéristiques du modèle et nous permettent de définir trois types en fonction des interactions entre les agents : (i) les modèles en lecture (L), comme flocking, (ii) les modèles en écriture idempotente (EI), comme le modèle virus, où l'état de l'agent ne change plus après une écriture et (iii) les modèles en écriture concurrente (EC). Il est alors possible d'assister le modélisateur en lui fournissant le choix de la politique de synchronisation qui permet l'implémentation correcte de son modèle tout en limitant l'impact sur les performances. Ainsi nous définissons dans la section suivante plusieurs politiques de synchronisation.

3.3 Politiques de synchronisation

Nous proposons ici plusieurs politiques de synchronisation, issues de l'analyse des modèles précédents, dont nous souhaitons évaluer l'impact sur les résultats et sur le temps d'exécution des simulations, sachant que d'autres politiques pourraient présenter un intérêt pour d'autres modèles. A ces propositions nous ajoutons le cas "sans synchronisation" qui sert de référence.
La politique **aucune synchronisation (NS)** distribue la simulation en n portions sans zone de recouvrement ni écritures distantes. Les agents peuvent se déplacer d'un processus à un autre en ayant un champ de perception tronqué lorsqu'ils sont proches des limites des processus.
La politique **overlapping zones (OLZ)** ne gère que des zones de recouvrement, qu'elle copie à chaque pas de temps. Les écritures n'y sont pas reportées sur les originaux et sont écrasées au pas de temps suivant par la mise à jours.
La politique **écritures asynchrones (EA)** fait des écritures à distance sans attendre une confirmation ou une valeur de retour. Elle est utilisée lorsqu'un agent modifie une donnée de la zone de recouvrement et que cette écriture doit être prise en compte dans le pas de temps courant mais que l'agent n'attend pas de donnée en retour.
La politique **synchronisation stricte (SS)** gère les zones de recouvrement et les écritures concurrentes pour garantir au maximum la reproduction du cas séquentiel. Chaque demande en écriture est bloquante jusqu'à l'acquittement de sa prise en compte, ainsi la cohérence des données est garantie. Elle est ce qu'il y a de plus strict en termes de synchronisation sans revenir à une exécution séquentielle.
Contrairement à la synchronisation stricte, la **synchronisation stricte décalée (SSD)** s'effectue de manière non-bloquante. Ainsi, lorsqu'un agent effectue une demande de synchronisation, il est mis en attente de réponse jusqu'à la fin du pas de temps afin que l'exécution des autres agents se poursuive, assurant ainsi un meilleur recouvrement calcul-communication.
Pour diriger le choix d'une politique de synchronisation pour un modèle, il est nécessaire de connaître l'incidence qu'aura le choix de cette politique sur le modèle parallélisé. Les spécifications données précédemment permettent de discerner l'impact sémantique et les contraintes garanties. Pour connaître l'incidence de chacune des politiques sur les performances de la simulation, nous présentons dans la suite les mesures de performance réalisées en implémentant ces politiques sur les modèles étudiés. Les modèles choisis sont spatialisés mais ces problématiques s'appliquent de la même manière sur des modèles non-spatialisés puisqu'elles sont liées aux échanges de données entre agents plutôt qu'à la position de ceux-ci.

4 Expérimentations

L'objectif de cette section est de mettre en évidence le lien entre les performances d'exécution d'un modèle et la politique de synchronisation choisie en fonction de la variation d'extensibilité ou de montée en charge. Comme aucune des plateformes vues précédemment n'offre ces politiques de synchronisation, nous les avons implémentées dans des modèles pour les tester.

4.1 Modèles et politiques de synchronisation

Nous utilisons les trois modèles vus précédemment, choisis pour leurs différents besoins de synchronisation. La table 1 donne les politiques de synchronisation utilisées avec chacun des modèles.
Le modèle Flocking est un modèle L. Deux politiques de synchronisation sont donc testées : aucune synchronisation (NS) et utilisation de zones de recouvrement (OLZ). Le

Modèle	Ghost	Type	Politiques
Flocking	Oui	L	NS - OLZ
Virus	Non	EI	EA - OLZ - SSD
PP	Non	EC	OLZ - SSD - SS

TABLE 1 – Politiques de synchronisation utilisés

modèle Virus est un modèle EI. Les politiques de synchronisation utilisées sont donc l'écriture asynchrone (EA), les zones de recouvrement (OLZ) et la synchronisation stricte décalée (SSD). La politique de synchronisation stricte (SS) n'est pas utilisée puisque l'écriture idempotente ne change pas l'état d'un agent contaminé. Il n'est donc pas nécessaire de gérer l'écriture concurrente sur son changement d'état puisque l'ordre d'exécutions de deux écritures n'a pas d'incidence sur le résultat final. Le modèle proie-prédateur est un modèle EC puisqu'une proie ne peut être mangée qu'une seule fois. Les politiques de synchronisation utilisées sont les zones de recouvrement (OLZ), la synchronisation stricte décalée (SSD) et la synchronisation stricte (SS).

4.2 La plateforme de test

L'implémentation des modèles a été effectuée à partir de notre plateforme FPMAS[10], une plateforme multi-agents parallèle qui repose sur la bibliothèque Zoltan pour gérer la distribution de la simulation, le modèle agent étant représenté par un graphe afin de tirer parti des algorithmes de partitionnement parallèles. FPMAS propose également des mécanismes de synchronisation, dont la synchronisation stricte, contrairement aux plateforme précédemment citées et qui n'offrent même pas la possibilité de l'implémenter.
Une fois le modèle et ses politiques de synchronisation implémentés, FPMAS peut l'exécuter dans un environnement parallèle adapté au calcul haute performance. La simulation est divisée en N portions et chacune des portions est distribuée sur P processus. Les performances dépendent alors du modèle exécuté. A noter que les politiques de synchronisation sont implémentées directement dans les modèles, ce qui peut engendrer quelques différences de performances d'un modèle à l'autre. Pour cette raison nous ne faisons par la suite que des comparaisons entre politiques par rapport à un modèle donné.
Pour exécuter les simulations, nous avons utilisé le Mésocentre de calcul de Franche-Comté. Le cluster est constitué de nœuds bi-processeurs, avec des processeurs Xeon E5 (8*2 cœurs) cadencés à 2.6 Ghz et 32 Go de mémoire vive. Le cluster possède un total de 1280 cœurs gérés par le système de batch SGE [1]. Les noeuds sont interconnectés par un réseau non bloquant QDR infiniBand [2] organisé en fat tree. Chaque point des courbes représente une moyenne de 10 exécutions avec 10 graines différentes.

4.3 Impact de l'extensibilité

L'extensibilité des modèles est étudiée en fixant le nombre d'agents de la simulation et en faisant varier le nombre de processus sur lesquels elle s'exécute. Nous mesurons l'impact de l'extensibilité à l'aide de deux métriques : le temps d'exécution et le speed-up.
Pour chacun des modèles nous avons calculé un speed-up avec comme référence le temps d'une exécution parallèle sur 16 cœurs car la taille des données des modèles est trop grande pour un seul processus. Le speed-up sur p processus est donné par $T(p_{ref})/T(p)$ où $T(p_{ref})$ est le temps d'exécution parallèle sur le nombre de processus de référence et $T(p)$ est le temps d'exécution sur p processus. Pour les trois modèles l'extensibilité est bonne avec un speed-up de plus ou moins 20 suivant les modèles, alors que le speed-up idéal est de 32. Les résultats obtenus par les politiques sans synchronisation, ou avec moins de synchronisation, sont meilleurs que ceux ayant des synchronisations plus strictes. Pour des raisons de reproductibilité, la même configuration initiale est utilisée pour toutes les exécutions, seule la graine varie d'une exécution à une autre.

Modèle Flocking. L'environnement du modèle Flocking est basé sur un cube 1000^3. A l'initialisation, 100000 oiseaux sont répartis de manière aléatoire dans l'espace. Les taux de cohésion, de séparation et d'alignement sont fixés à 1 tandis que le taux d'aléa est lui fixé à 1.5. Ces paramètres ont été choisis, afin de ne pas favoriser l'un des trois critères composant les comportements des oiseaux. Seul le taux d'aléa est supérieur aux autres taux dans le but de générer des déplacements plus chaotiques et donc de tester davantage la synchronisation. Chaque simulation est exécutée durant 2000 pas de temps.

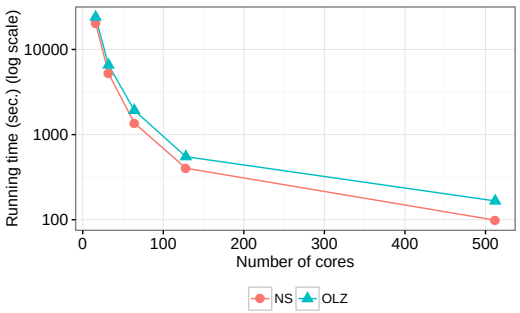


FIGURE 1 – Temps d'exécution du modèle Flocking

La figure 1 donne les temps d'exécution du modèle Flocking pour les politiques de synchronisation *NS* et *OLZ*. La différence des temps d'exécution entre les deux politiques de synchronisation est d'environ 15% pour 16 cœurs, 26% pour 128 cœurs, et 65% pour 512 cœurs. Lorsque le nombre de cœurs augmente, plus de messages sont nécessaires pour mettre à jour les zones de recouvrement, ce qui explique cette différence croissante.

Modèle Virus. Le modèle Virus a été exécuté sur une grille de 1000^2 qui représente l'environnement avec une capacité maximale de 500000 personnes. A l'initialisation 9600 personnes sont saines et 640 sont infectées par le virus. Tous les agents sont positionnés de manière aléatoire sur l'environnement. Le taux d'infection est fixé à 0.65 et le

1. https://en.wikipedia.org/wiki/Oracle_Grid_Engine
2. https://fr.wikipedia.org/wiki/Bus_InfiniBand

taux de reproduction est fixé à 0.2. Le taux de récupération, c'est à dire le fait qu'une personne infectée devienne immunisée, est fixé à 0.5. Ces valeurs sont issues du modèle Virus de NetLogo. Seule la taille de l'environnement et la capacité maximale ont été adaptées pour obtenir un modèle de grande taille. Pour finir, chaque simulation est exécutée durant 800 pas de temps.

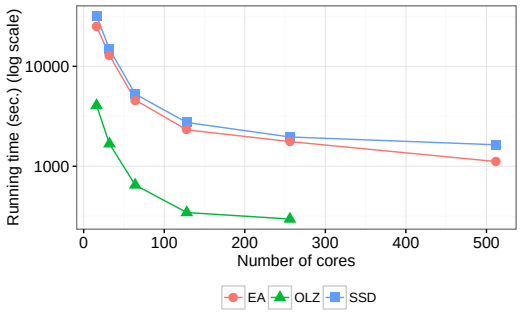


FIGURE 2 – Temps d'exécution du modèle Virus

La figure 2 donne les temps d'exécution du modèle virus pour les synchronisations *OLZ*, *EA* et *SSD*. Pour les courbes *OLZ* il n'y a pas de point pour 512 cœurs car nous n'avons pas eu assez de temps sur le calculateur pour réaliser l'expérimentation. Les résultats obtenus avec le modèle flocking se confirment ici avec une différence de 27% pour 16 cœurs, seulement 15% pour 128 et 47% pour 512 cœurs. Deux raisons expliquent ce surcoût : le traitement additionnel en fin de pas de temps de la synchronisations SSD et l'augmentation du nombre de messages nécessaires à la synchronisation due au plus grand nombre de cœurs. Cette figure nous montre également le coût induit par la synchronisation des agents. Pour 16 cœurs, les temps d'exécution de la courbe *OLZ* sont environ 8 fois meilleurs que la courbe *SSD*. Cette différence tend à décroître avec l'augmentation du nombre de cœurs, par exemple, pour 256 cœurs cette différence n'est plus que de 6.6. Pour les courbes *EA* et *SSD* le ratio n'est que de 1.27 pour 16 cœurs, et de 1.4 pour 512 cœurs. Il croît donc avec le nombre de cœurs du fait du nombre plus important de messages.

Modèle proie-prédateur. Le modèle proie-prédateur (PP) utilisé pour les expérimentations est basé sur un environnement grille de 400^2 où 25000 moutons et 17000 loups sont initialement positionnés de manière aléatoire. Le modèle des comportements et l'initialisation de l'énergie des agents loups et moutons est issue du modèle NetLogo. L'énergie gagnée par un mouton lorsqu'il mange de l'herbe est fixée à 5 et à 20 lorsqu'un loup mange un mouton. En ce qui concerne les taux de reproduction, ils sont fixés à 0.5 pour les moutons et à 0.4 pour les loups. Les durées de vie sont respectivement de 4 et 20 pour les moutons et les loups. La croissance de l'herbe est de 8. Pour finir les simulations sont exécutées durant 2000 pas de temps.
La figure 3 présente les temps d'exécution du modèle PP pour les politiques de synchronisation *OLZ*, *SSD* et *SS*. Pour la même raison que précédemment, aucun calcul n'a été

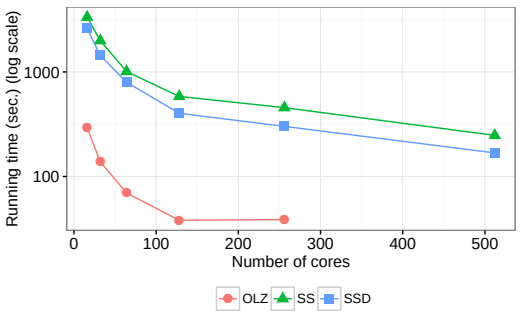


FIGURE 3 – Temps d'exécution du modèle PP

effectué sur 512 cœurs avec *OLZ*. Les points de 16 à 64 cœurs permettent cependant de mettre en évidence, comme pour le modèle virus, le coût important à payer pour avoir des politiques de synchronisations strictes. La courbe *SSD* montre que la relaxation de la synchronisation à la fin du pas de temps de cette politique permet un gain de temps d'exécution par rapport à la synchronisation stricte. Ce gain est dû au fait que le traitement des agents n'est plus bloqué en attente de la réponse à une écriture concurrente : on a donc un meilleur recouvrement calcul-communication.
Sur les modèles étudiés, le calcul des speed-up montre une bonne extensibilité et le niveau de synchronisation choisi pour implémenter le modèle ne semble pas induire d'impact. Les simulations avec un grand nombre de cœurs profitent donc bien du parallélisme, ce qui confirme que les systèmes multi-agents peuvent bénéficier d'une parallélisation. Néanmoins, les politiques de synchronisation ont un coût très important (jusqu'à un facteur 8), qui est dû aux communications engendrées. En effet, dans les modèles étudiés, les agents ont un comportement relativement simple qui s'exécute beaucoup plus rapidement qu'une communication.

4.4 Impact de la montée en charge

La montée en charge est réalisée en fixant le nombre de processus et en faisant varier le nombre d'agents. Le modèle proie-prédateur n'est pas présenté dans cette section car il est très difficile d'y faire varier le nombre d'agents puisque la population s'auto-équilibre.

Modèle Flocking. Le jeu de valeurs utilisé pour évaluer la montée en charge du modèle Flocking est le même que précédemment, à la différence que le nombre d'agents oiseaux qui composent la simulation varie de 100000 à 1000000.
La figure 4 présente l'impact de la montée en charge sur 512 cœurs. Les politiques *NS* et *OLZ* supportent bien la charge jusqu'à 500000 agents. Au delà de 500000 agents, les courbes croissent plus rapidement. Sans surprise, la version sans synchronisation supporte mieux la charge que la version avec zone de recouvrement, qui consomme environ un tiers de performance en plus. L'accroissement de la charge à partir de 500000 agents s'explique par un manque d'optimisation lors de la recherche du voisinage des agents. Au lieu de mettre à jour uniquement les agents qui arrivent ou partent du voisinage, un parcours de l'environnement est effectué à chaque pas de temps pour établir le champ de

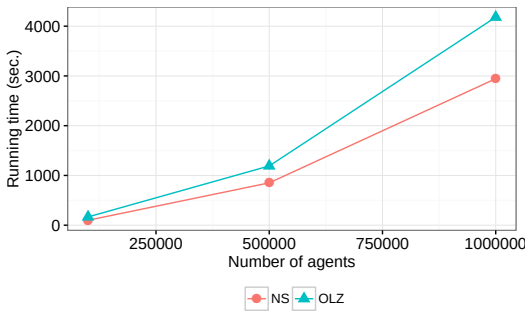


FIGURE 4 – Montée en charge du modèle Flocking

perception de chaque agent.

Modèle Virus. Les paramètres utilisés pour la montée en charge sont les mêmes que précédemment.

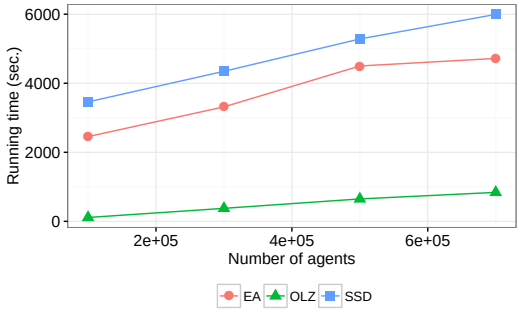


FIGURE 5 – Montée en charge du modèle Virus

La figure 5 montre l'impact de la montée en charge du modèle Virus de 100000 à 700000 agents et 800 pas de temps. Seulement 64 cœurs ont été utilisés pour cette courbe car il n'a pas été possible d'obtenir à nouveau autant de ressources de calcul (512 cœurs) sur le cluster partagé.
La courbe *OLZ* supporte évidemment mieux puisqu'elle ne gère pas les écritures. Pour les autres, on constate que *EA* supporte mieux la charge que *SSD*. La courbe *SSD*, reste linéaire, alors que la courbe *EA* croît très peu de 500000 à 700000 agents. Ceci est dû au surcoût lié à l'accumulation des synchronisations en fin de pas de temps. Le ratio de performance obtenu entre 100k et 700k pour la courbe *EA* est de 1.9, 1.73 pour *SSD* et 7.5 pour *OLZ*.
Les résultats sur la montée en charge confirment que, quelle que soit la politique de synchronisation utilisée, les modèles multi-agents tirent bénéfice d'une parallélisation. Les résultats obtenus avec la politique de synchronisation *OLZ* montrent que cette politique est plus efficace que les politiques plus contraintes. Nous avons donc étudié l'impact des politiques de synchronisation sur les résultats des exécutions, ce que nous présentons dans la suite.

4.5 Impact sur les résultats

Pour étudier l'impact des différentes politiques, nous observons deux résultats : le résultat de la simulation lui-même et le nombre d'interactions incohérentes. Le résultat d'une simulation est ce qui est attendu par le modélisateur. Nous étudions l'impact d'une exécution plus ou moins synchronisée sur les résultats de l'exécution. Par ailleurs, dans les cas où les écritures dans les zones de recouvrement sont gérées, c'est à dire avec les politiques *EA*, *SS* et *SSD*, des interactions peuvent mettre en évidence une incohérence entre l'information locale et l'information distante. Par exemple pour le modèle proie prédateur, un loup essaie de manger un mouton dans la zone de recouvrement seulement si il est vivant. Par conséquent, un retour négatif pour cette action montre que le mouton était en fait déjà mort dans les données distantes. En conséquence, prendre seulement en compte la zone de recouvrement aurait mené à une action incohérente où au moins deux loups auraient mangé le même mouton. De même, pour le modèle virus, un agent essaie d'en infecter un autre seulement si ce dernier est sain. Ainsi, pour analyser l'impact des interactions sur les résultats des simulations nous comptabilisons le nombre total de demandes de synchronisation, les synchronisations pour lesquelles l'information était cohérente (notées *CO*) et celles pour lesquelles l'information était incohérente (notées *NCO*). Cette analyse n'inclut pas le modèle Flocking car elle ne s'applique pas à un modèle en lecture.

Modèle Virus. La figure 6 présente le détail des demandes de synchronisation pour les synchronisations *EA* et *SSD*.

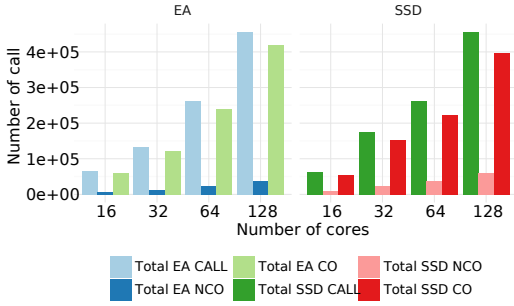
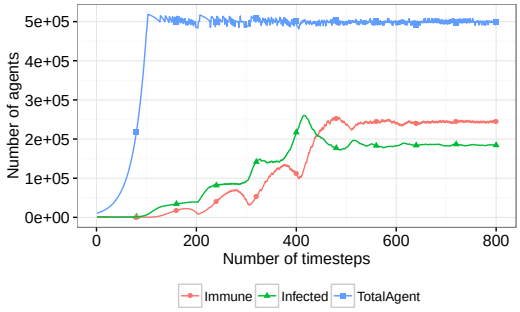


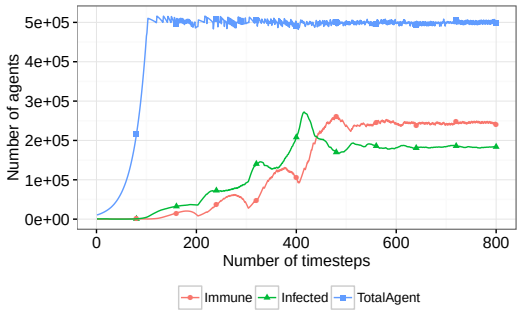
FIGURE 6 – Détail des synchronisations du modèle Virus

Pour les deux politiques de synchronisation le nombre total de demandes de synchronisation croît de manière linéaire avec le nombre de cœurs car, lorsque le nombre de cœurs augmente la simulation est divisée en plus de processus. Il y a donc plus de zones de recouvrement et plus d'interactions potentielles. Le nombre d'interactions *NCO* est très faible pour *EA*. Il croît d'environ 10% pour *SSD* à cause des synchronisations gérées en fin de pas de temps. Les agents qui effectuent une demande de synchronisation sont suspendus et exécutés à la réception de la réponse, en fin de pas de temps. De ce fait, certaines interactions qui étaient potentiellement *CO* au moment de la demande de synchronisation peuvent devenir *NCO* si les agents concernés n'avaient pas encore été exécutés. Il faut donc poser la question de savoir quel est l'impact de ces interactions *NCO* sur les résultats.
La figure 7 présente les résultats de l'exécution sur 128 cœurs. Les résultats pour *SSD* (figure 7(a)) et *EA* (7(b)) sont quasi identiques. Les quelques variations s'expliquent par

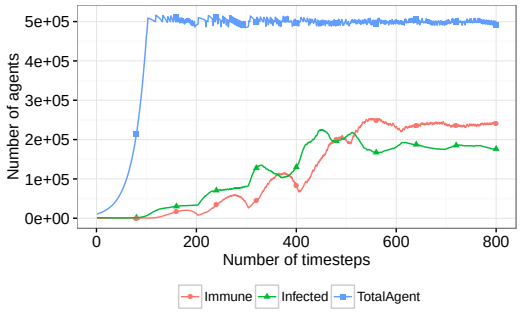
les interactions *NCO* traitées en fin de pas temps par la politique *SSD*. En revanche, les résultats de *OLZ* (figure 7(c)) présentent de nombreuses différences, malgré une tendance identique.



(a) SSD



(b) EA



(c) OLZ

FIGURE 7 – Résultats d'une exécution du modèle Virus

Modèle proie-prédateur. La figure 8 présente le détail des synchronisations lors de l'exécution du modèle PP. Comme pour le modèle Virus, le nombre de demandes de synchronisations croît de manière linéaire avec le nombre de cœurs. Le nombre d'appels *NCO* reste plus faible pour la politique *SS* que pour la politique *SSD*.
La figure 9 présente les résultats sur 128 cœurs des politiques *SSD*, *SS* et *OLZ*. Le modèle PP est un modèle très sensible : la modification d'un paramètre peut conduire à une instabilité qui se traduit par la mort d'une des espèces. Le changement d'une politique induit ainsi des différences de résultats entre les courbes *SSD* (figure 9(a)) et les courbes *SS* (figure 9(c)). La figure 9(b) qui représente l'exécution sans synchronisation montre des résultats proches de la politique *SSD* (figure 9(a)). Nous retrouvons tout de même des courbes cycliques qui tendent à s'équilibrer.

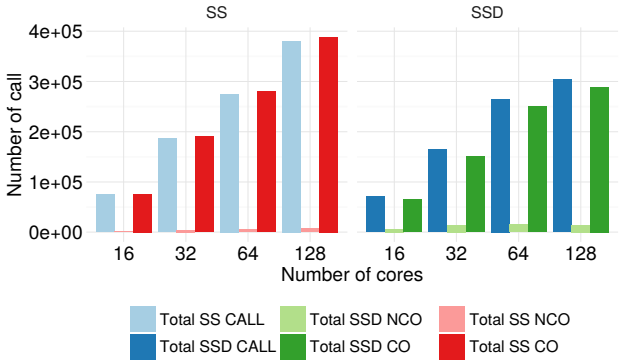
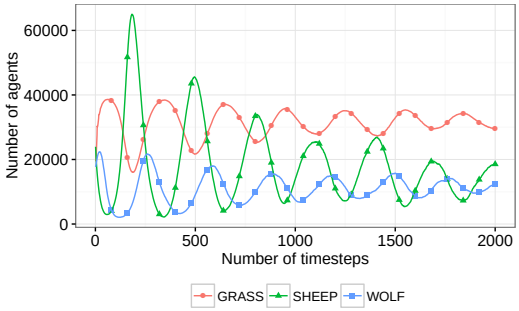
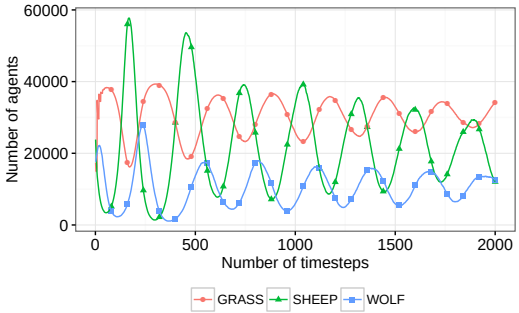


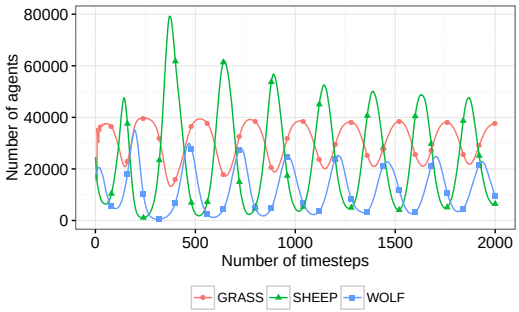
FIGURE 8 – Détail des synchronisations du modèle PP



(a) SSD



(b) OLZ



(c) SS

FIGURE 9 – Résultats d'une exécution du modèle PP

Au vu des courbes, on peut s'interroger sur ce qu'il est important d'observer dans les résultats du modèle. Si nous regardons plus en détails ces figures, nous remarquons que le nombre maximal de moutons diffère, environ 60000 pour *OLZ* et *SSD* et environ 80000 pour *SS*. On remarque également que sur la figure 9(c) (synchronisation *SS*) la phase est régulière avec un décalage de l'ordre de $\pi/2$. En revanche, sur les figures 9(b) et 9(a), un décalage de phase s'effectue au cours du temps. Les courbes *OLZ* et *SSD* sont semblables et donnent des résultats erronées.

5 Conclusion

Notre étude sur l'impact des politiques de synchronisation vise à sensibiliser la communauté agent, et plus particulièrement les modélisateurs, aux problèmes liés à la parallélisation d'un modèle. Nous évaluons pour cela différentes politiques de synchronisation en lien avec différents modèles et nous mettons en évidence que, suivant le modèle à implémenter, le niveau de synchronisation a un impact sur performances d'exécution et les résultats. A travers nos expérimentations, nous quantifions le coût de la synchronisation sur les performances de la simulation suivant les modèles et nous montrons son impact sur les résultats. La synchronisation est de fait dépendante des comportements des agents : c'est au modélisateur de prévoir l'exécution du modèle et donc d'adapter la modélisation pour prendre en compte le coût et l'impact de la synchronisation lors de la conception d'un modèle parallèle.
Pour la suite nos travaux s'orientent vers une analyse de nombreux modèles en vue d'identifier les problèmes génériques de synchronisation et de proposer des politiques efficaces, qui pourraient être implémentées dans une plateforme pour simplifier le travail du modélisateur.

Remerciements

Les calculs présentés dans cet article ont été effectués sur le calculateur du Mésocentre de calcul de Franche-Comté

Références

[1] E. S. Angelotti, E. E. Scalabrin, and B. C. Ávila. Pandora : a multi-agent system using paraconsistent logic. In *Computational Intelligence and Multimedia Applications, ICCIMA*, pages 352–356. IEEE, 2001.

[2] K. Mani Chandy and Jayadev Misra. Distributed simulation : A case study in design and verification of distributed programs. *IEEE Transactions on software engineering*, (5) :440–452, 1979.

[3] AL Chin, AD Worth, AC Greenough, S Coakley, M Holcombe, and M Kiran. Flame : An approach to the parallelisation of agent-based applications. *Work*, 501 :63259, 2012.

[4] Nicholson Collier and Michael North. Parallel agent-based simulation with repast for high performance computing. *SIMULATION*, Nov 2012.

[5] G. Cordasco, R. Chiara, A. Mancuso, D. Mazzeo, V. Scarano, and C. Spagnuolo. A Framework for Distributing Agent-Based Simulations. In *Euro-Par 2011 : Parallel Processing Workshops*, volume 7155 of *LNCS*, pages 460–470, 2011.

[6] Jacques Ferber and Jean-François Perrot. *Les systèmes multi-agents : vers une intelligence collective*. Inter-Editions Paris, 1995.

[7] Olivier Gutknecht and Jacques Ferber. Madkit : A generic multi-agent platform. In *4th intl Conf. on Autonomous agents*, pages 78–79. ACM, 2000.

[8] David R Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(3) :404–425, 1985.

[9] A. Rousset, B. Herrmann, C. Lang, and L. Philippe. A survey on parallel and distributed multi-agent systems for high performance computing simulations. *Computer Science Review*, 22 :27 – 46, nov 2016.

[10] A. Rousset, B. Herrmann, C. Lang, L. Philippe, and H. Bride. Nested graphs : A model to efficiently distribute multi-agent systems on hpc clusters. *Concurrency and Computation : Practice and Experience*, 30(7) :e4407, 2018.

[11] B. Samadi. Distributed simulation, algorithms and perfomance analysis. *PhD Thesis, Computer Science Department, Univ. of California, Los Angeles*, 1985.

[12] P. Taillandier, DA. Vo, E. Amouroux, and A. Drogoul. GAMA : A Simulation Platform That Integrates Geographical Information Data, Agent-Based Modeling and Multi-scale Control. In *Principles and Practice of Multi-Agent Systems*, volume 7057, pages 242–258. Springer, 2012.

[13] S. Tisue and U. Wilensky. Netlogo : Design and implementation of a multi-agent modeling environment. In *Proceedings of Agent*, volume 2004, pages 7–9, 2004.

[14] U. Wilensky. Netlogo flocking model. *Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL*, 1998.

[15] U. Wilensky. Netlogo virus model. *Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL*, 1998.

[16] U. Wilensky. Netlogo wolf sheep predation (docked) model. *Center for connected learning and computer-based modeling, Northwestern Univ., Evanston, IL*, 2005.

[17] Yadong Xu, Wentong Cai, Heiko Aydt, Michael Lees, and Daniel Zehe. Relaxing synchronization in parallel agent-based road traffic simulation. *ACM Trans. Model. Comput. Simul.*, 27(2) :14 :1–14 :24, May 2017.

[18] J. A. Yorke, N. Nathanson, G. Pianigiani, and J. Martin. Seasonality and the requirements for perpetuation and eradication of viruses in populations. *American Journal of Epidemiology*, 109(2) :103–123, 1979.