Prise en compte de la mobilité logique des données en environnement pair-à-pair

Rapport de stage

Sergey Legtchenko
sergey.legtchenko@upmc.fr

Encadrement:

Sébastien Monnet
sebastien.monnet@lip6.fr

Gaël Thomas
gael.thomas@lip6.fr

Résumé

La mobilité de données est une propriété importante de certaines applications distribuées en pair-à-pair fortement dynamiques. Un avatar d'un jeu massivement multijoueur est en déplacement constant dans son monde virtuel. Les métadonnées d'un participant à un réseau social évoluent dans le temps. Si l'emplacement de ces entités dans la topologie pair-à-pair est déterminé en fonction de leurs caractéristiques sémantiques, les entités sont fréquemment amenées à migrer de nœud en nœud. Cela impose de fortes contraintes d'adaptabilité au réseau logique sous-jacent. Ce document présente un mécanisme permettant au système pair-à-pair de mieux s'adapter à ce contexte fortement dynamique en anticipant la mobilité de la donnée grâce à l'analyse de ses caractéristiques sémantiques. Une évaluation des gains via le simulateur Peersim est menée.

Table des matières

1 Introduction

Les technologies de l'information jouent un rôle fondamental dans la société actuelle. Participant à tous les secteurs d'activités, elles font maintenant partie intégrante de notre mode de vie [22]. Chaque unité de calcul est désormais amenée à jouer un rôle *actif* dans cette Société de l'Information. On voit ainsi apparaître le besoin pour une multitude d'applications massivement réparties, possédant des contraintes aussi nombreuses que variées. Le modèle pair-à-pair semble parfaitement adapté pour servir de support à ces applications. Pourtant, le nombre d'applications reposant sur le modèle pair-à-pair est encore relativement faible. De plus, la grande majorité de ces applications répandues s'occupe uniquement de partage de fichiers (tels Gnutella, Bittorrent ou Kademlia).

Parallèlement à cela, de nombreuses applications émergentes (telles que les jeux multijoueurs, les réseaux sociaux collaboratifs, etc.) massivement réparties se basent toujours sur le modèle client-serveur, pourtant plus limité en terme de passage à l'échelle. Ceci illustre le fait que le fonctionnement des réseaux logiques (ou *overlays*) pair-à-pair existants est peu adapté aux besoins de ces applications émergentes. Ces applications ont pour caractéristique commune leur forte dynamicité : les valeurs sémantiques de leurs données sont amenées à évoluer (parfois très rapidement) dans le temps, nécessitant une fréquente migration de données de nœud en nœud dans le système. De plus, cette modification sémantique doit généralement être propagée à l'ensemble des copies de la donnée, elles aussi en déplacement. Les systèmes pair-à-pair "classiques" ne sont plus en mesure de supporter de telles contraintes, car incapables de rapidement s'adapter aux besoins de l'application. En effet, une grande partie des algorithmes des systèmes pair-à-pair actuels est conçue dans le seul but de maintenir les propriétés de l'overlay en lui-même (connexité, routage efficace, etc.) sans prendre en compte les besoins de l'application. Un tel overlay, bien que disposant de bonnes propriétés topologiques, est alors trop rigide pour être réellement efficace en environnement dynamique.

Il est donc nécessaire de mettre au point des mécanismes efficaces pour rapidement adapter la topologie d'un overlay pair-à-pair à la dynamicité de l'application. Axé sur cette problématique, ce travail propose un mécanisme permettant d'améliorer l'adaptabilité d'un système pair-à-pair existant, Solipsis, à la mobilité logique de données. Ce mécanisme (ainsi que Solipsis lui-même) ont intégralement été implémentés dans le simulateur Peersim, permettant de mettre en evidence un gain non négligeable apporté par notre mécanisme adaptatif.

Le reste du document s'organise comme suit : La première partie introduit des définitions importantes et met en lumière quelques contraintes posées par les applications émergentes. La seconde partie introduit le concept de malléabilité applicative et présente le fonctionnement de Solipsis. La troisième partie expose en détail notre mécanisme de prise en compte de la mobilité des données. Son évaluation est menée dans la dernière partie. Enfin, une conclusion permet de présenter les perspectives du travail réalisé.

2 Contexte : Mobilité logique et Environnements Virtuels

Afin d'assurer la bonne compréhension de l'ensemble de la contribution apportée par ce document, il est nécessaire de définir quelques notions clés, et de présenter un tour d'horizon des études déjà effectuées sur le sujet.

2.1 Définitions et exemples

Pour définir la mobilité logique, il convient en premier lieu de définir la notion d'*Environnement Virtuel* dans lequel cette mobilité survient. Il sera donc entendu dans la suite de l'étude qu'un Environnement Virtuel (E.V.) est un espace de nommage applicatif[1] à $n \geq 1$ dimensions muni d'une notion de distance. Il peut ainsi bien évidemment s'agir d'une simulation du monde réel, mais peut tout aussi bien être un espace dans lequel la notion de distance porte une signification sémantique quelconque. Cette définition introduit alors tout naturellement la notion de déplacement logique d'un objet d'un tel environnement, que l'on peut caractériser comme la variation dans le temps de la distance de cet objet par rapport à un référentiel donné. Par la suite, uniquement les E.V. massivement répartis seront mentionnés. On considère qu'un système est massivement réparti lorsqu'il s'agit d'un système à grande échelle, c'est-à-dire qu'il comporte au moins plusieurs centaines d'entités physiquement distantes. Enfin, par souci de clarté il est nécessaire de préciser que les termes *pair* et *nœud* sont considérés comme synonymes et utilisés de manière interchangeable au sein de ce document.

Les exemples les plus répandus d'E.V. tel qu'il a été défini existent actuellement dans le domaine des Jeux Massivement Multijoueur (*Massively Multiplayer Online Games* ou *MMOG*) dont le plus populaire est certainement World Of Warcraft [24] et de mondes virtuels à caractère social (e.g. Second Life, [25]). Dans les deux cas, l'E.V. simule de manière plus ou moins éloignée le monde réel. Les utilisateurs sont représentés dans le monde virtuel par des avatars, via lesquels ils peuvent communiquer entre eux et interagir avec le monde virtuel. Il s'agit, dans le premier cas, d'un jeu de rôle, dont le but est d'accomplir diverses quêtes afin de faire progresser son avatar. Second Life, quant à lui, propose d'accomplir toute sorte d'actions à caractère social (discussions, activités de groupe, etc...). Il est intéressant de remarquer que ces deux applications possèdent une architecture centralisée, et ne sont donc pas indéfiniment scalables. Leur monde virtuel est divisé en larges zones, chacune sous la responsabilité d'un serveur distinct. Le transfert entre les zones ne se fait pas dans la continuité du déplacement du personnage. Il s'agit le plus souvent d'une téléportation suivie d'un temps de chargement correspondant au transfert entre les serveurs [12]. Par ailleurs, le nombre d'avatars présents à un instant donné dans une zone est limité. Dans le cas de Second Life, il est généralement inférieur à 200 [13]. De plus, une telle infrastructure est souvent payante pour les utilisateurs et sujette aux pannes des serveurs. Pour pallier à ces problèmes, un certain nombre de mondes virtuels basés sur le paradigme pair-à-pair ont été proposés [8, 7, 4, 2].

[1] i.e. les coordonnées d'entités de l'espace ne sont pas déterminées statiquement par l'infrastructure du système sous jacent, mais définies dynamiquement en rapport avec les propriétés logiques de ladite entité.

Les deux premiers, Donnybrook et Colyseus, ciblent les jeux de type *First Person Shooter (FPS)*, c'est à dire des jeux d'action rapide pour lesquels une très faible latence est primordiale. Ces deux plates-formes ne sont pas indéfiniment scalables, car chaque joueur doit périodiquement envoyer des informations à *tous* les joueurs présents dans son champ d'action. On peut alors constater que le nombre de participants à ces jeux ne peut dépasser au plus quelques milliers [8].

Le projet Solipsis [4, 2] propose quant à lui un monde virtuel (i.e. une simulation du monde réel ou *Métavers*[23] en deux dimensions) entièrement decentralisé et scalable, sans toutefois viser une application particulière (pas de contraintes fortes sur la latence, ni de borne sur le nombre d'utilisateurs).

2.2 Caractéristiques des Environnements Virtuels

Le processus adaptatif d'un système pair-à-pair est, dans les cas exposés précédemment, dépendant de la dynamique de l'E.V. qu'il supporte. Il apparaît dès lors essentiel de dégager les caractéristiques de cette dynamique afin de pouvoir optimiser grâce à cela la capacité d'adaptation de l'infrastructure pair-à-pair.

Plusieurs études ont été menées afin d'estimer la répartition de données ainsi que leur dynamique dans les E.V. existants. Ainsi, le monde virtuel Second Life [25] a fait l'objet d'analyses statistiques sur la répartition et le déplacement de ses participants [12, 13]. Une étude similaire [17] a été menée sur le jeu de rôle World of Warcraft [24]. Dans un cadre plus dynamique, une collecte de données empiriques a également été réalisée pour le jeu *FPS* Quake III[2] [7]. Enfin, une estimation de la répartition des données en fonction de leurs caractéristiques sémantiques a été faite dans [9].

Plusieurs tendances se dégagent de ces études. Tout d'abord, on remarque que l'ensemble des travaux dénotent une répartition très disparate des données (objets, avatars, etc...). L'environnement virtuel est peuplé par amas, avec des zones très densément fournies, et d'autres quasiment désertes. C'est le cas pour des données regroupées par valeurs sémantiques [9], ou bien pour les avatars d'un monde virtuel [7, 17, 13]. Il s'agit de distributions à rapprocher d'un modèle suivant une loi de Zipf [16].

En ce qui concerne la mobilité dans les mondes virtuels, on constate que les avatars restent longtemps dans certaines zones populaires, alors que de nombreuses autres sont quasi-désertes (c.f. Fig 5). Le déplacement dans les zones populaires est lent et chaotique (voire inexistant), alors que les zones impopulaires sont traversées très rapidement, de façon quasi réctiligne [13]. Étonnamment, ces observations sont en grande partie confirmées pour le *FPS* Quake III [7], alors même que la nature du jeu est très différente des mondes virtuels étudiés en [13]. De plus, La et Michiardi font remarquer que les *patterns* de déplacement observés sont somme toute très proches de ceux constatés dans dans le monde réel [12].

Enfin, on voit que l'intéressement mutuel des avatars dépend beaucoup du type de monde virtuel. En effet, les avatars de Second Life ont tendance à rester en contact de

[2]jeu d'action populaire nécessitant beaucoup de déplacements rapides et d'ésquives de la part des joueurs.

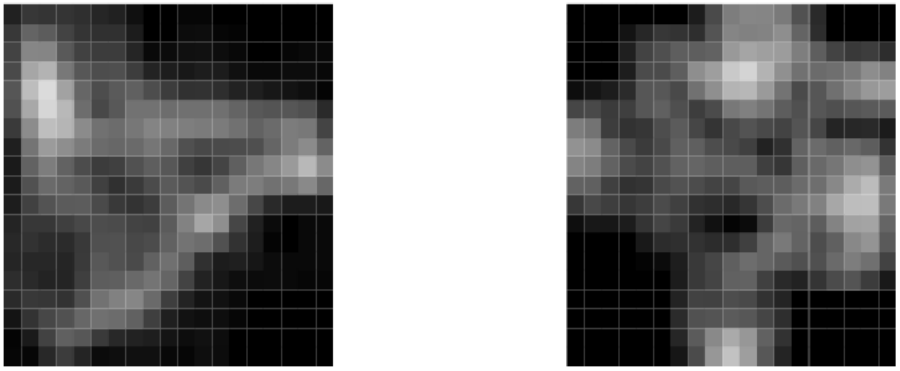

FIG. 1 – *Deux cartes d'environnements du jeu Quake III (avec Colyseus). Les zones claires correspondent aux zones populaires*

manière prolongée, alors que les zones d'intérêt des participants à un jeu de type *FPS* varient très rapidement [8].

La dynamiques des E.V. observés est donc très disparate, que ce soit dans la répartition des données et des participants, leurs mouvements, ou bien leurs zones d'intérêt, suggérant que les modèles de distribution aléatoire de données parfois utilisés pour l'évaluation de systèmes pair-à-pair de ce type sont très loin de la réalité.

2.3 Optimisations liées à la mobilité logique

Afin d'anticiper au mieux la dynamique de l'Environnement Virtuel, il est souhaitable d'inclure quelques mécanismes d'optimisation dans l'infrastructure pair-à-pair correspondante. La mise en place de caches, sous réserve d'employer une bonne politique de gestion de cache, peut être une solution adaptée.

Cette piste a notamment été explorée dans [10, 18, 20]. Ces articles décrivent entre autres quelques politiques de gestion de caches d'objets du monde virtuel. Ces études sont toutefois basées sur le modèle client/serveur (c'est pourquoi il s'agit d'un cache d'objets, et non d'un cache des pairs du voisinage logique). Ainsi, un serveur central possède une base de données d'objets virtuels éparpillés dans l'E.V. qu'il supporte. Au fur et à mesure du déplacement de ses clients dans cet environnement, le serveur leur transmet les caractéristiques des objets rencontrés (position, textures, formes, propriétés...). Afin d'optimiser la bande passante entre le serveur et les clients, l'implémentation d'un cache chez le client est décrite. Ainsi, chaque participant au monde virtuel garde en cache les objets récemment rencontrés.

Plusieurs politiques de gestion sont proposées. Une politique appelée MRM (*Most Required Movement*) est par exemple fréquemment citée [18, 20, 10]. Il s'agit d'ordonner les objets présents dans le cache selon leur distance algébrique et angulaire par rapport

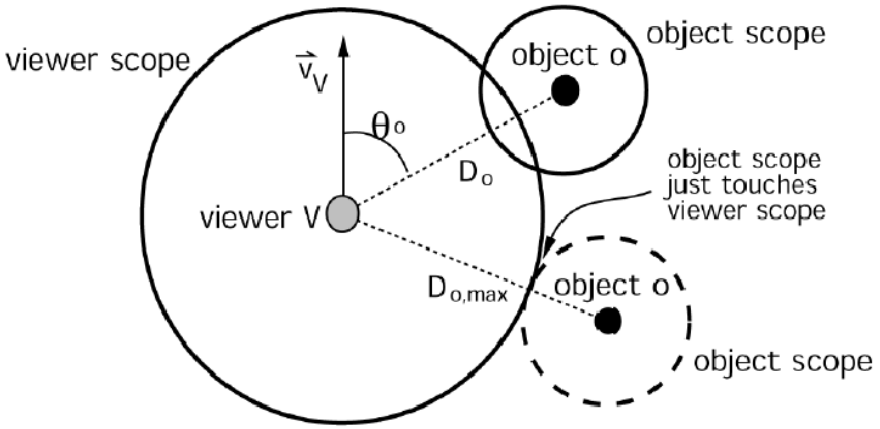

FIG. 2 – *Calcul du Most Required Movement pour un objet o vu par un utilisateur v*

au propriétaire du cache[3] (c.f. Fig. 2). Les objets les plus éloignés sont ceux qui seront en priorité éjectés du cache lors d'une mise à jour. Cette politique adaptée aux déplacements en Environnement Virtuel apparaît significativement plus efficace (de l'ordre de 40% d'amélioration du *hit ratio* [20]) que le LRU classique.

Une autre politique est proposée par Park et al. [10]. Elle prend en compte la popularité d'un objet ainsi que l'intérêt que lui porte le propriétaire du cache, en plus de sa distance. Dans ce cas, l'intérêt d'un objet pour un utilisateur est déterminé *à priori* (e.g. en fonction du rôle de l'avatar dans le monde virtuel), ou bien dynamiquement au fur et à mesure des interactions de l'utilisateur. La popularité est la mesure du nombre d'accès à un objet par *l'ensemble* des utilisateurs. Une somme pondérée des trois facteurs (distance, intérêt, et popularité) permettent de définir une politique de gestion plutôt efficace (10% d'amélioration par rapport à *Most Required Movement* en moyenne).

Pour minimiser l'occupation de la bande passante et proposer à l'utilisateur un rendu satisfaisant, la mise en cache peut être complétée par une méthode appelée *Multi Resolution Modeling* [18, 20]. En effet, les objets du monde virtuel éloignés de l'observateur, bien qu'étant dans sont champ de vision, sont très peu visibles. Ils peuvent donc être représentés avec très peu de détails, et leur chargement sera par conséquent plus rapide. Le *Multi Resolution Modeling* permet de stocker un unique objet sous forme de liste ordonnée de vues, allant de la moins détaillée à la plus détaillée, chaque maillon stockant la différence dans les détails entre les vues. Ainsi, lorsqu'un objet apparaît aux limites du champ de vision de l'utilisateur, seule sa vue peu détaillée est chargée. Cependant, au fur et à mesure du déplacement, de plus en plus de détails sont chargés, pour enfin aboutir à la vue rapprochée la plus détaillée. Ceci permet donc d'optimiser la bande passante,

[3] $d_{v,o} = \omega(\frac{D_o - r_{O_v}}{r_{O_v}}) + (1-\omega)(1 - \frac{|\theta_{o,v}|}{\pi})$ avec D_o la distance entre l'observateur et l'objet o, r_{O_v} le rayon du disque d'interêt de l'observateur, et $\theta_{o,v}$ la distance angulaire entre l'observateur et l'objet, ω étant le coefficient de pondération.

mais peut également influer sur la gestion du cache [20]. Ainsi, lors de la mise à jour du cache, un objet éloigné ne sera pas systématiquement entièrement éjecté du cache : une représentation "faible fidélité" pourra tout de même y être conservée pendant un certain temps.

Une technique de pré-chargement est souvent associée à ces méthodes de mise en cache. Il s'agit dans ce cas de prédire le mouvement de l'utilisateur dans le monde virtuel et de charger les objets *en avance* sur le chemin de celui-ci. La politique de prédiction peut être identique à la politique de gestion de cache, comme c'est le cas dans [10]. Dans ce cas, les objets à télécharger en avance sont par exemple déterminés grâce à la formule prenant en compte l'intérêt de l'utilisateur pour l'objet, sa popularité et son éloignement, ou bien avec MRM.

La notion de pré-chargement a par ailleurs été utilisée par Colyseus [7] et Donnybrook [8], deux plates-formes pair-à-pair pour les jeux FPS à faible latence, afin de déterminer à l'avance les objets à charger au fur et à mesure du déplacement du joueur. Dans ces deux systèmes, les zones d'intérêt des joueurs évoluent en fonction de leur déplacement. En complément à cette technique, Colyseus utilise un cache des zones d'intérêt. Ces deux systèmes pair-à-pair ont été testés en conditions réelles (avec les jeux Quake II et Quake III, deux FPS populaires), et les propriétés de scalabilité et de faible latence mises en évidence suggèrent que les techniques de mise en cache et de prédiction s'avèrent efficaces pour gérer la mobilité.

Les techniques décrites dans les deux dernières parties, à savoir la mise en cache, et la prédiction des mouvements, permettent de gérer efficacement la complexité d'un environnement pair-à-pair en anticipant les actions et créant des mécanismes de mise à jour rapide. Cependant, dans certains cas, ces techniques s'avèrent insuffisantes. Cela peut par exemple se produire si la structure pair-à-pair est est complexe et se prête mal à la mobilité, ou bien que la densité d'une portion de l'environnement virtuel est très élevée.

Dans le cas de VoroNet [1], le découpage de l'environnement virtuel avec un diagramme de Voronoï [15] est calculatoirement coûteux pour des espaces de dimension supérieure à 2. Par conséquent, la mobilité spatiale (déplacement) et temporelle (churn) des pairs ne peut y être gérée de manière efficace. Pour remédier à cela, dans Raynet [5], une technique probabiliste est employée. Un algorithme de Monte-Carlo est utilisé afin d'estimer de manière approximative le volume de chaque cellule de Voronoï. Ces approximations, nettement moins coûteuses, permettent malgré tout de bâtir un maillage pair-à-pair basé sur les diagrammes de Voronoï à n ($n \geq 2$) dimensions.

L'emploi de techniques d'approximation peut également être une solution pour augmenter de manière significative la scalabilité d'un système pair-à-pair. Ainsi, Donnybrook, une plate-forme de jeu répartie pour les jeux *FPS* [4] [8] exploite le fait que le cerveau humain n'est capable de maintenir l'attention que sur un nombre d'objets très limité. C'est pourquoi, à chaque période, un pair sélectionne dans sa zone d'intérêt cinq objets qu'il juge les plus dignes d'intérêt pour le joueur. L'état de ces cinq objets est alors mis à jour à une fréquence élevée, pour garantir une cohérence maximale jusqu'à

[4] Il s'agit de la "suite" de Colyseus.

l'élection d'un nouvel ensemble de cinq objets. Les autres entités de la zone d'intérêt sont jugées peu intéressantes, et leur état est mis à jour à une fréquence très faible. L'élection se fait selon trois critères :

- La proximité : un objet proche est plus enclin à attirer l'attention qu'un objet éloigné.
- La présence dans le champ de vision du joueur : un objet attire d'autant plus l'attention qu'il se trouve au centre de l'écran.
- La proximité temporelle : un objet qui a déjà été au centre d'une attention de la part du joueur le sera probablement encore pendant la période suivante.

La somme pondérée de ces critères détermine les cinq objets dont le comportement est à reproduire avec une haute fidélité.

En revanche, les autres objets de la zone d'intérêt peuvent n'être mis à jour que de manière très relâchée. Ainsi, la période de synchronisation des copies de l'objet avec l'original proposée dans Donnybrook est d'environ une seconde. Un tel intervalle est bien entendu perceptible par le joueur et peut entraîner des incohérences de vues du jeu. Pour éviter cela, le contrôle des entités en mouvement (avatars, objets en déplacement...) est pris par un algorithme local au joueur. Son but est de simuler de manière crédible le comportement de l'entité durant l'intervalle séparant deux mises à jour authoritatives. Une approximation du comportement de l'entité est ainsi obtenue, et la cohérence de l'environnement est conservée avec un faible coût de maintenance (une mise à jour toutes les secondes seulement). Les imprécisions dues aux imperfections de l'algorithme de guidage ne sont pas critiques, étant donné que les objets ainsi guidés ne sont pas au centre de l'intérêt du joueur. Cette technique, connue sous le nom de *Dead Reckoning* [21], permet à Dannybrook d'être scalable jusqu'à plusieurs centaines (voire des milliers) de joueurs sous Quake III, alors que les versions centralisées du jeu ne peuvent supporter plus d'une soixantaine de joueurs.

Les approches par approximation permettent ainsi de réduire la charge de l'E.V. sur les nœuds de son infrastructure, optimisant les performances et assurant le passage à l'échelle du système.

3 Overlays supportant la mobilité logique

Après avoir défini les notions associées d'Environnement Virtuel et de mobilité logique, puis présenté quelques exemples de techniques d'optimisation liées à celle-ci, il convient de s'intéresser à l'ensemble des overlays capables d'en satisfaire les contraintes.

3.1 Malléabilité applicative : une propriété indispensable

Plus particulièrement, la propriété de *malléabilité applicative* apparait comme caractéristique commune des overlays pair-à-pair supportant les E.V. existants. En effet, les entités de l'E.V. tel qu'il a été défini précédemment possèdent des coordonnées virtuelles

pouvant varier de manière dynamique au cours du temps de vie de ces entités. Or l'environnement est logiquement partagé par un ensemble de pairs, chacun ayant la responsabilité d'un sous-ensemble de l'espace virtuel. Une entité en déplacement est donc amenée tôt ou tard à entrer en contact avec des entités régies par un pair distant. Cela implique un échange de données entre les pairs responsables de ces entités. Afin de permettre un transfert efficace, il est judicieux de s'assurer que les deux pairs sont alors voisins dans la structure logique, ou du moins qu'un mécanisme efficace permette le transfert rapide et optimisé de l'information. Dans le cas contraire, le temps de routage à travers l'infrastructure engendrerait une latence accrue et un trafic réseau superflu. Cela signifie que l'organisation de l'E.V. influe dans une large mesure sur l'organisation de la structure sous-jacente. L'application gérant l'Environnement Virtuel doit donc pouvoir modifier le graphe logique des pairs qui la supportent : on parle de *malléabilité applicative*.

Plusieurs exemples de systèmes pair-à-pair "malléables" existent actuellement. Ainsi, MOve [3] est une infrastructure pair-à-pair dans laquelle certains liens entre voisins évoluent dynamiquement en fonction des besoins de la couche applicative. En permettant à l'application de modifier la topologie du maillage pair-à-pair, il faut cependant prendre soin de conserver les bonnes propriétés structurelles de celui-ci, à savoir sa connexité et un routage efficace. En effet, il est possible que l'application nécessite une forte agglomération (ou *clustering*) des pairs, c'est-à-dire un regroupement d'ensembles de nœuds du maillage en amas faiblement reliés au reste du système [3]. Dans ces cas, il est nécessaire de restreindre quelque peu l'influence de l'application, et maintenir un bon compromis entre clustering et connexité.

MOve résout le problème en divisant le voisinage de chaque pair en deux catégories. La première est un ensemble de voisins structurels, qui ne peuvent être modifiés par l'application. Ces pairs sont choisis aléatoirement de façon à garantir une bonne connexité du graphe. Il est en effet connu [14] que lorsque chaque nœud d'un graphe possède au moins $log(n)$ (avec n le nombre de nœuds du graphe) liens aléatoirement répartis, le graphe est connexe avec une très forte probabilité. La deuxième catégorie de voisinage englobe les voisins applicatifs d'un pair, c'est-à-dire les pairs dont la proximité est décidée par l'application. Cette couche du maillage possède habituellement un fort coefficient de regroupement permettant d'exprimer la proximité sémantique des pairs.

Les liens applicatifs peuvent être à leur tour divisés en sous catégories, chacune dépendant d'une sémantique applicative particulière. Une telle approche est globalement assez générique et peut permettre le support d'applications aux sémantiques très diverses.

Un autre système pair-à-pair capable de malléabilité applicative est Mercury [9]. Ce système est malléable par l'intermédiaire de son mécanisme d'équilibrage de charge. Il s'agit d'une infrastructure pair-à-pair permettant la recherche de données par champs sémantiques (*semantic range queries*). Ainsi, les données sémantiquement proches dans l'E.V. sont stockés sur des pairs logiquement proches dans la structure du maillage pair-à-pair. Ceci pose un problème en terme d'équilibrage de charge. En effet, le système doit pouvoir fournir la possibilité de recherche par champs, ce qui signifie que les données ne peuvent y être aléatoirement distribuées comme cela est fait via une fonction de hachage

dans les DHT.
Chaque pair de Mercury est responsable d'une parcelle de l'espace sémantique. Il peut s'avérer qu'un pair ait la charge d'une parcelle particulièrement dense en données, provoquant une surcharge. Pour remédier à cela, Mercury propose un mécanisme d'équilibrage de charge particulier. Chaque pair réalise à intervalles réguliers un sondage de l'espace de nommage global pour isoler les zones denses en données. Les pairs en charge de parcelles peu denses sont alors invités à rejoindre les zones denses pour en répartir la charge. Une fois le processus stabilisé, la structure du système pair-à-pair reflète la répartition des données dans son Environnement Virtuel : l'infrastructure s'est transformée pour mieux supporter la charge de l'application.

Enfin, l'overlay de Solipsis[5] présente des caractéristiques prononcées de malléabilité applicative. Cet overlay pair-à-pair sert de support à un *Métavers*. Chaque entité du *Métavers* représente un nœud du maillage logique. Les entités peuvent se déplacer à leur gré dans l'E.V. de Solipsis. Naturellement, en fonction de son déplacement, le voisinage virtuel d'une entité est amené à changer. Le maillage logique doit alors suivre cette évolution, et dynamiquement changer au fur et à mesure du déplacement. Afin d'assurer la connexité d'une structure aussi mobile, la mise à jour du voisinage d'un pair est régie par des règles strictes. Ces règles sont relativement simples et faciles à implémenter. Cependant, malgré sa simplicité de fonctionnement, il est performant. C'est donc un parfait exemple d'overlay fortement malléable. Pour ces raisons, Solipsis a été choisi dans la suite de l'étude comme cas particulier pour l'implémentation de nos algorithmes.

3.2 Cas d'étude : Solipsis

Étant donné que Solipsis sert de base à nos expérimentations, il convient d'en décrire le fonctionnement avec plus de précision. Tout d'abord, il est important de bien distinguer l'overlay de Solipsis de l'application qui utilise cet overlay (i.e. le Métavers). En effet, l'overlay de Solipsis est hautement malléable par le Métavers, et un nœud de l'overlay est en charge d'un unique avatar dans le Métavers, ce qui entraine un amalgame entre les deux entités pourtant bien distinctes. Par ailleurs, il est important de remarquer que chacune des entités possède sa propre notion de voisinage. En effet, alors que le voisinage du Métavers est déterminé par la distance algébrique entre deux avatars, celui de l'overlay est déterminé par le nombre de *hops* entre deux nœuds dans la topologie pair-à-pair. C'est pourquoi deux nœuds voisins dans l'overlay ne sont pas forcement en charge des avatars les plus proches dans le Métavers. En revanche, par souci de cohérence du Métavers, les algorithmes de Solipsis s'efforcent d'assurer que deux nœuds proches sémantiquement (ie. la distance entre leurs avatars dans le Métavers est faible) soient voisins topologiques. Pour ce faire, la construction de sa topologie se base sur deux règles :

1. *Règle de la zone de connaissance :* Tout avatar A possède un rayon de connaissance R_c. La zone de connaissance de A est définie comme le cercle de centre A et de

[5]Il existe en fait deux versions de Solipsis de conception très différente, il s'agit de [4] et [2]. Par la suite, nous considérerons uniquement la première version.

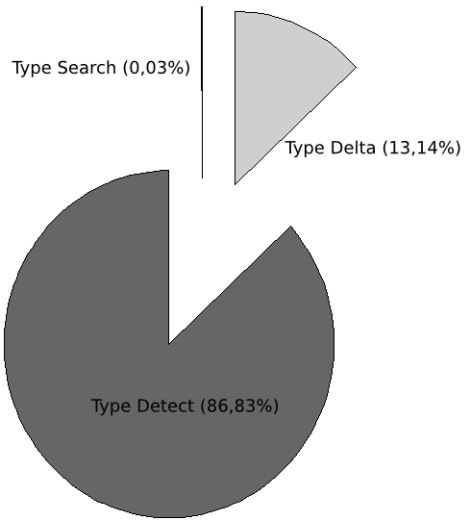

FIG. 3 – *Répartition des messages de Solipsis pour un degré de mobilité d'environ 50‰(cf. section 5.1). Le type Detect est émis par le protocole de collaboration spontanée. Le type Delta désigne la propagation des mises à jour des coordonnées. Le type Search concerne la recherche de voisinage en cas d'échec du protocole de collaboration spontanée*

rayon R_c. Tout avatar situé à l'intérieur du cercle de connaissance de A doit être voisin de A dans la topologie pair-à-pair.

2. *Règle de l'enveloppe convexe :* Tout avatar du Métavers doit se trouver à l'intérieur de l'enveloppe convexe contenant l'ensemble de ses voisins logiques.

L'intérêt de la première règle est évident : un avatar doit avoir connaissance de toute entité située dans sa zone d'intérêt. Deux avatars proches (i.e. chacun est dans la zone d'intérêt de l'autre) sont très probablement amenés à échanger une quantité importante de données, c'est pourquoi il est primordial qu'il soient voisins. L'intérêt de la seconde règle est explicité dans la suite de la section.

En complément de ces deux règles, un mécanisme d'entraide mutuelle appelé *collaboration spontanée* est mis en place dans l'overlay. Lorsqu'un avatar A détecte le déplacement de l'un de ses voisins D (ou bien que D arrive dans le voisinage de A), A analyse l'ensemble de son voisinage, et prévient ceux de ses voisins qui sont susceptibles d'être intéressés par la modification. Soit V_A un voisin de A. A considère que V_A est intéressé par D si D se trouve désormais dans la zone d'intérêt de V_A ou bien que D est désormais plus proche de V_A que de A ne l'est (i.e. D se déplace *vers* V_A). La nouvelle d'un déplacement dans un secteur du Métavers est alors rapidement et efficacement propagée à tous les avatars intéressés.

Le bon fonctionnement de ce mécanisme est assuré par la deuxième règle. En effet, un avatar situé à l'intérieur de l'enveloppe convexe formée par son voisinage est de ce fait

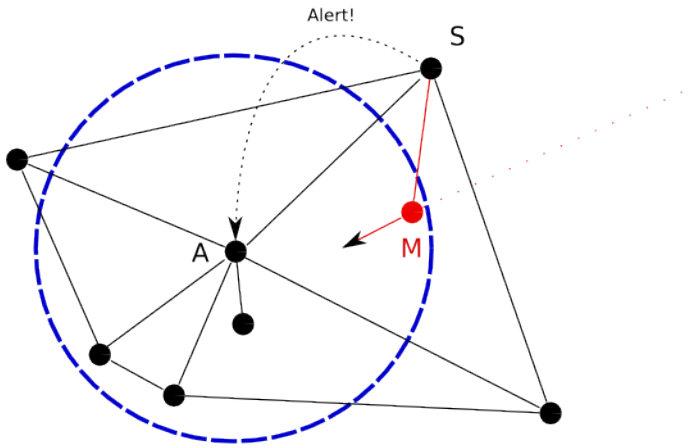

FIG. 4 – *Algorithme de collaboration spontanée en action : l'avatar M entre dans la zone d'intérêt de A. L'avatar S, voisin de M et A, avertit A.*

entouré par ses voisins dans le Métavers. Chaque voisin de cet entourage sert en quelque sorte de sentinelle et prévient l'avatar de tout changement dans le Métavers. La propriété de l'enveloppe convexe assure qu'il existe un voisin dans "chaque direction" du Métavers, ne laissant aucune direction sans surveillance. En cas d'échec du maintien de la topologie par le mécanisme d'avertissement, un nœud doit lancer une succession de requêtes à ses voisins afin de restaurer au plus vite les deux règles fondamentales. Cette procédure est assez fastidieuse, car il faut tour à tour interroger son voisinage en quête d'un nouveau nœud à ajouter, ce qui est assez long, surtout dans un contexte dynamique. Toutefois, nos simulations montrent que ce cas de figure n'arrive que rarement, car le mécanisme de propagation spontanée est extrêmement efficace : une fois la topologie correctement constituée, un avatar sera averti de tout changement dans la majeure partie des cas, et ne devra pratiquement jamais initier de recherches pour rétablir la cohérence par ses propres moyens (cf. figure 3). Il s'agit en fait d'un mécanisme puissant de malléabilité applicative : l'overlay s'adapte très rapidement au déplacement d'un avatar.

Chaque nœud/avatar de Solipsis ne peut posséder qu'un nombre limité de voisins. La taille de la zone d'intérêt de l'avatar est dictée par cette limite. En effet, lorsque le nombre limite de voisins est dépassé, cela signifie qu'il se trouve dans une zone de forte densité. Il réduit alors le rayon de sa zone de connaissance pour en exclure les voisins les plus éloignés, qui sont alors déconnectés. Inversement, si l'avatar se trouve dans une zone désertique et n'a pas assez de voisins, il lui suffit d'agrandir sa zone de connaissance pour se voir attribuer de nouveaux voisins par l'intermédiaire du mécanisme d'entraide

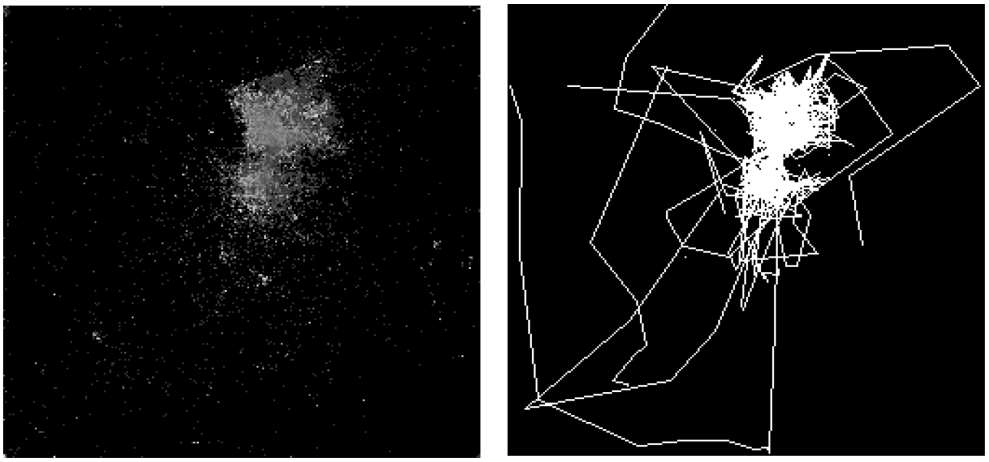

FIG. 5 – ***A droite :*** *Étude de la répartition des avatars d'une ile de Second Life, les parties claires représentent des zones de forte densité.* ***A gauche :*** *déplacement des avatars sur la même ile. Le déplacement en zone déserte est rectiligne, alors qu'il est chaotique dans les zones denses*

spontanée décrit précédemment. Les simulations montrent qu'un nombre relativement faible de voisins (de l'ordre de la dizaine) est amplement suffisant dans la plupart des cas[6] pour garantir la stabilité d'un overlay de plusieurs milliers de nœuds.

3.3 Solipsis et la mobilité logique

Malgré sa simplicité, Solipsis est donc un système bien adapté aux exigences posées par son application. La topologie de son overlay évolue avec le Métavers au fil des mouvements de ses avatars. Cependant, lors de la conception de Solipsis, aucune hypothèse particulière n'a été prise quant au modèle de déplacement des avatars ni à la distribution de densité du Métavers. Celui-ci est donc supposé uniforme, avec un mouvement des avatars portant un caractère plutôt aléatoire. Or, d'après les diverses études menées sur les Mondes Virtuels existants (cf. section 2), ce n'est clairement pas le cas. Notamment, deux tendances se dégagent :

1. La densité est *fortement* disparate : Il existe dans le Métavers des zones extrêmement denses, séparées par des quasi-déserts.
2. Le mouvement des avatars est le plus souvent rectiligne, uniforme et très rapide dans les zones désertes, et très chaotique dans les zones denses.

Ces propriétés du Métavers sont plutôt néfastes pour son overlay, car elles requièrent une capacité d'adaptation accrue. Pour s'en convaincre, il suffit d'observer quelques cas

[6] il est théoriquement possible qu'un nœud situé dans une zone particulièrement dense soit entouré uniquement de nœuds déjà surchargés et se voit refuser quelques connexions, ce qui peut nuire au respect de la règle de l'enveloppe convexe de Solipsis, mais la probabilité que *tous* les nœuds des environs soient également surchargés est infime.

de figure susceptibles se fréquemment se produire dans un Métavers possédant les deux tendances énumérées précédemment. Considérons un avatar en déplacement à grande vitesse dans une zone déserte entre deux zones denses. Le mécanisme de collaboration spontanée est dans ce cas moins efficace que dans le cas uniforme. En effet, la densité des avatars y est moindre, ce qui diminue la probabilité pour un avatar d'être mis au courant du mouvement par collaboration spontanée. Plus grave, lors de l'arrivée rapide de l'avatar dans la zone dense, un grand nombre de nœuds devront subitement être avertis, leurs petites zones d'intérêt (à cause de la forte densité) ne leur permettant pas d'être avertis suffisamment à l'avance par la collaboration spontanée. Une telle arrivée provoquerait alors un pic localisé de charge réseau extrêmement néfaste pour l'overlay.

Des problèmes peuvent également survenir lorsque un avatar placé dans une zone dense décide de changer de zone. Il augmente donc sa vitesse au milieu de la zone dense, ce qui active la collaboration spontanée entrainant des messages d'alerte en cascade. Or les distances entre les avatars des zones de forte densité sont faibles, ce qui fait que la propagation se fera très lentement dans le Métavers[7], et sera susceptible d'être dépassée par le mouvement de l'avatar.

Enfin, dans un autre cas de figure, le mouvement chaotique d'un avatar au sein d'une zone dense peut provoquer de nombreux messages inutiles. Cet avatar va fréquemment revenir sur ses pas, obligeant la topologie à constamment annuler son début d'adaptation pour revenir à la position initiale. Cependant, ce dernier effet peut dans une large mesure être atténué grâce à l'utilisation d'un cache de voisinage.

Il devient tout de même clair que le modèle de mobilité choisi (et également le plus répandu d'après les études effectuées) nécessite que la collaboration spontanée soit secondée par un nouveau mécanisme plus réactif.

4 Conception et réalisation de mécanismes d'anticipation de la mobilité

4.1 Déscription générale

Ainsi, comme le montre le cas particulier de Solipsis, les caractéristiques des E.V. peuvent poser des contraintes difficiles à respecter même avec des algorithmes adaptatifs très efficaces. Ceci est en partie dû au manque d'anticipation de ces algorithmes, qui ne prennent en compte que la modification effective de l'E.V., sans capter les *tendances* de ces modifications. Dans le cadre général, considérons que chaque donnée mobile d'un E.V. possède un ensemble de métadonnées relatives à son état (position dans l'E.V., vitesse, nature du déplacement, etc.). L'évolution de ces métadonnées au cours du temps obéit aux régles en vigueur dans l'E.V. Si le système est capable d'intégrer ces règles (*"saisir"* la sémantique de l'évolution des métadonnées), il est capable de prévoir la modification des métadonnées, et donc le déplacement de la donnée dans l'E.V..

[7] bien qu'elle soit toujours aussi rapide dans la topologie, ce qui illustre bien la différence entre les deux notions de distance.

Bien entendu, étant donné le caractère aléatoire des déplacements, il est quasiment impossible de prédire l'évolution d'une donnée en particulier sur le long terme. Néanmoins, les tendances globales sont décelables, et le déplacement de certaines données est même prévisible sur le court ou moyen terme. Un ensemble de mécanismes "capturant" les tendances de la dynamique d'un E.V. pourrait garantir à l'overlay une réactivité et une adaptabilité remarquable.

Solipsis n'échappe pas à la régle. En effet, le Métavers, doit posséder un modèle de mobilité semblable à celui décrit précédemment, qui constitue en soi un ensemble de règles (observées statistiquement) régissant le déplacement des avatars. Dans ce cas, les métadonnées de chaque avatar sont ses coordonnées, sa vitesse, le vecteur de son déplacement, et la nature de celui-ci (rectiligne ou chaotique). En outre, en tant que simulation du monde réel, chaque avatar possède une masse, et donc une cinétique propre sur sa trajectoire.

Ces règles sont relativement simples, ce qui signifie qu'il est en partie possible de les intégrer au mécanisme de base de Solipsis et ainsi anticiper la dynamique du Métavers. En premier lieu, le caractère rectiligne et rapide des déplacements des avatars dans les zones peu denses est prédictible avec une grande probabilité. Il est alors possible d'anticiper le déplacement et obtenir les informations nécessaires en avance sur le mouvement. On peut remarquer qu'il s'agit de l'idée du pré-chargement des objets, déjà exposée dans plusieurs travaux (cf. section 2.3) dans un cadre client-serveur. En revanche, aucune publication à notre connaissance ne traite du pré-chargement d'objets en environnement pair-à-pair. L'objectif est donc d'avertir par avance les nœuds de la topologie de Solipsis susceptibles de se trouver sur le chemin d'un avatar dans le Métavers. Le problème posé peut être décomposé en deux sous-problèmes distincts :

- Intégrer le plus finement possible les règles du Métavers au sein de l'organisation de Solipsis.
- Identifier et mettre en relation les nœuds concernés avec le nœuds hébergeant l'avatar en déplacement.

4.2 Techniques et Algorithmes employés

Il est tout à fait possible de décrire un ensemble de règles à l'aide d'automates. C'est la solution choisie pour le premier sous-problème. Ainsi, chaque nœud N de Solipsis intègre un automate à états finis α_N responsable de la description du déplacement de l'avatar E_N. Il existe deux types de mouvements dans le modèle de déplacement décrit dans les Mondes Virtuels existants, auxquels s'ajoute le déplacement nul (ie. l'état immobile). Chaque automate α_N possède donc trois états (cf. figure 6) :

1. **H**(alted) : L'état dans lequel l'avatar est immobile.
2. **T**(ravelling) : L'avatar est dans un déplacement rectiligne rapide.
3. **W**(andering) : Le mouvement de l'avatar est lent et chaotique.

Ainsi, α_N est initialisé à l'état H lors d'une connexion de N dans le Métavers. À intervalles réguliers, le mécanisme de mise à jour de l'automate analyse l'historique les modifications des coordonnées de l'avatar et en déduit la dynamique du déplacement.

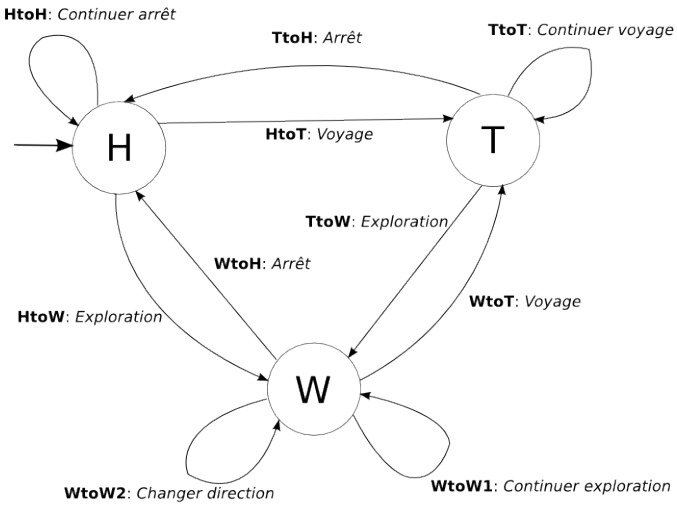

FIG. 6 – *Automate décrivant les mouvements d'un avatar.* ***En gras*** *: le nom de la transition, en italique sa sémantique*

Tant qu'aucun mouvement n'est perçu, l'automate reste dans l'état H. Cependant, aussitôt qu'un déplacement est détecté, il est évalué et classé comme rapide/rectiligne ou bien lent/chaotique. Selon les résultats de la classification, l'automate change d'état soit vers T (rapide/rectiligne), soit vers W (lent/chaotique). En complément de cette analyse, l'algorithme déduit également :

– le vecteur vitesse
– l'accélération

En possession de toutes ces informations, le nœud N est en mesure d'émettre une hypothèse concernant les futures positions de son avatar E_N dans le Métavers. Si l'automate est dans l'état T (le mouvement de l'avatar porte donc un caractère uniforme et rapide), les positions futures peuvent être prédites avec une précision acceptable, et le mécanisme peut donc commencer à anticiper le déplacement en recherchant les avatars susceptibles de se trouver sur le chemin de E_N : il s'agit là du second sous-problème.

En effet, il est nécessaire en premier lieu d'identifier les avatars concernés, puis de router un message d'avertissement (requête de pré-chargement) à leurs nœuds respectifs. Solipsis ne possède pas vraiment de mécanisme de routage spécialement efficace, sa topologie n'étant pas à priori formée pour respecter la propriété de Kleinberg[8] [6]. Le routage doit donc être effectué de proche en proche. Des échanges rapides ne peuvent être initiés qu'avec le voisinage proche, à savoir les nœuds situés à un *hop* de N. Il serait donc peu optimal de faire d'abord la recherche des nœuds concernés, puis de router les requêtes jusqu'à eux. La collecte des nœuds intéressés doit se faire au moment de la

[8]ie. aucun mécanisme n'est spécialement dédié à la formation et au maintien de liens longs.

propagation de la requête. Pour ce faire, la requête de pré-chargement initiée par l'avatar en déplacement inclut les valeurs du vecteur vitesse, de l'accélération, et la position de l'avatar au moment de l'envoi de la requête. Ainsi, les nœuds recevant la requête pourront évaluer par rapport à leurs propres coordonnées si ils se trouvent ou non sur le chemin de l'avatar en mouvement. La recherche des nœuds candidats au pré-chargement ne doit s'effectuer ni trop près de la position du nœud initiateur, ni trop loin de celui-ci.

Plus précisément :

- Les avatars qui se trouvent bien sur le chemin du déplacement de l'émetteur de la requête, mais qui se trouvent trop près de celui-ci sont de mauvais candidats pour le pré-chargement, car une fois prévenus, ils seront déjà dépassés par l'avatar en mouvement et seront avertis par les mécanismes de base de Solipsis. La distance minimale d_{min} entre un avatar E et un candidat au pré-chargement C est déterminée de la manière suivante :

$$\boxed{d_{min} = KR_E + V_E * RTT_{C,E} + \epsilon}$$

  avec :
  - KR_E le rayon de connaissance de E,
  - V_E la vitesse de E,
  - RTT_C une estimation du RTT (*Round Trip Time*) moyen à partir de C,
  - ϵ une variable représentant la marge d'erreur des valeurs estimées.

  Ainsi, une fois la requête de pré-chargement reçue par un nœud, celui-ci estime la distance entre la position de son avatar et les coordonnées auxquelles *devrait* se trouver l'avatar initiateur calculée à partir des informations contenues dans la requête. Si cette distance est inférieure à la distance minimale, le nœud n'est pas un bon candidat, et le message est simplement transmis au destinataire suivant. Le processus de choix des destinataires sur chaque nœud est décrit en détail dans le paragraphe suivant.
- De plus, le nœud source doit fixer un TTL en nombre de *hops* pour le message au moment de l'envoi de la requête. Ce TTL est décrémenté à chaque fois qu'un avatar placé sur le chemin supposé du mouvement est découvert, et va déterminer le nombre maximal de réponses que l'émetteur souhaite recevoir. Il peut s'agir d'une constante du système, ou bien être déterminé dynamiquement en fonction de la charge réseau ou des caractéristiques du mouvement de l'avatar. Il est ainsi aisément possible de contrôler le nombre de messages générés indifféremment de la densité de la zone. En effet, une autre solution aurait été de fixer un TTL en limitant la distance *algébrique* de propagation dans le Métavers, mais le nombre de messages généré serait alors dépendant de la densité de la zone traversée, pouvant provoquer une surcharge réseau.

Algorithme de propagation

L'algorithme de propagation (cf. Algorithme 1) consiste à choisir parmi ses voisins celui dont la distance angulaire par rapport à l'axe du mouvement est minimale et qui

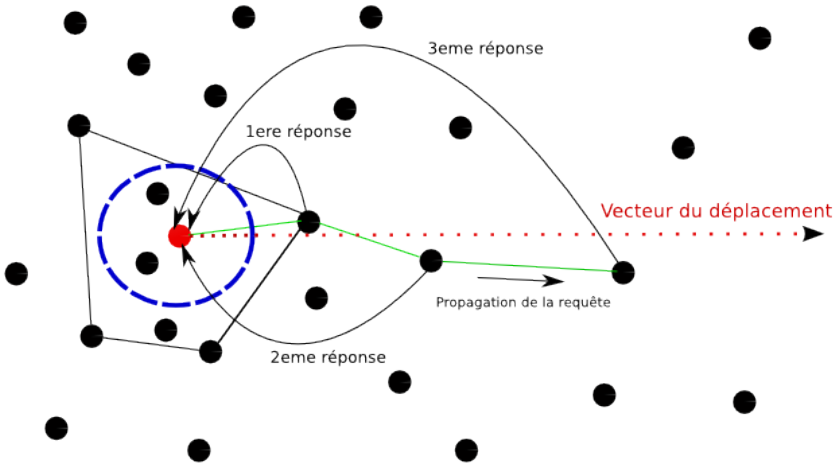

FIG. 7 – *Algorithme de propagation : la requête est transmise aux nœuds situés dans la direction du mouvement. Chaque nœud répond pour lui* ***et*** *tous ses voisins concernés*

se situe dans le sens du mouvement de l'avatar initiateur de la requête (cf. figure 7).

Lorsqu'un nœud C reçoit une telle requête, il vérifie en premier lieu qu'il se trouve bien à bonne distance du nœud source (*lignes -1 à 4-*).

Si le TTL du message est encore suffisamment élevé pour supporter la recherche de candidats supplémentaires, il analyse son voisinage connu, et choisit un ensemble R_C de nœuds susceptibles d'intéresser le nœud source de la requête (*lignes -6 à 8-*). Dans l'algorithme 1, cet ensemble est ordonné : nœudsProchesDuChemin[0] désigne ainsi le meilleur candidat.

Si la taille de R_C dépasse le TTL du message, R_C est réduit à une taille TTL (*lignes -10 à 12-*)[9].

Le message avec la liste des candidats munis de leurs identifiants réseau (adresses IP) est constitué (*lignes -13 à 16-*).

Le message de réponse ainsi constitué est envoyé au nœud source (*lignes 18 et 19*).

Enfin, si le TTL le permet encore, le message est envoyé au nœud suivant dans le sens du déplacement (*lignes -22 à 25-*). De cette manière, le nombre de messages émis est strictement contrôlé, et la propagation du message se fait en profondeur suivant le vecteur de déplacement de l'émetteur.

Une variante de l'algorithme de propagation consiste à choisir plusieurs successeurs à

[9]Ceci est fait pour que l'initiateur puisse récupérer au maximum le nombre de candidats egal au ttl qu'il a spécifié.

Algorithm 1: Sur réception de msg, un message de pré-chargement en profondeur :

Result: recherche de nœuds propices au préchargement et propagation de la requête.

```
1  info = msg.informationPrechargement ();
2  dmin = calculerDmin ();
3  estimation = estimerPosition (info);
4  if distance (estimation, maPosition) ≥ dmin then
5      ttl = msg.recupererTTL ()−1;
6      if ttl > 0 then
7          noeudsProchesDuChemin = choisirProches (ensembleVoisins, info);
8          taille = noeudsProchesDuChemin.recupererTaille ()−1;
9          hopSuivant = noeudsProchesDuChemin [0];
10         if taille > ttl then
11             taille = ttl ;
12             noeudsProchesDuChemin = noeudsProchesDuChemin [1 .. taille ];
13         end
14         for noeud ∈ noeudsProchesDuChemin do
15             reponse.ajouter (noeud);
16             ttl = ttl −1;
17         end
18         source = msg.emetteur ();
19         envoyer (reponse, source);
20     end
21 end
22 if ttl != 0 then
23     msg.setTTL (ttl);
24     envoyer (msg, hopSuivant);
25 end
```

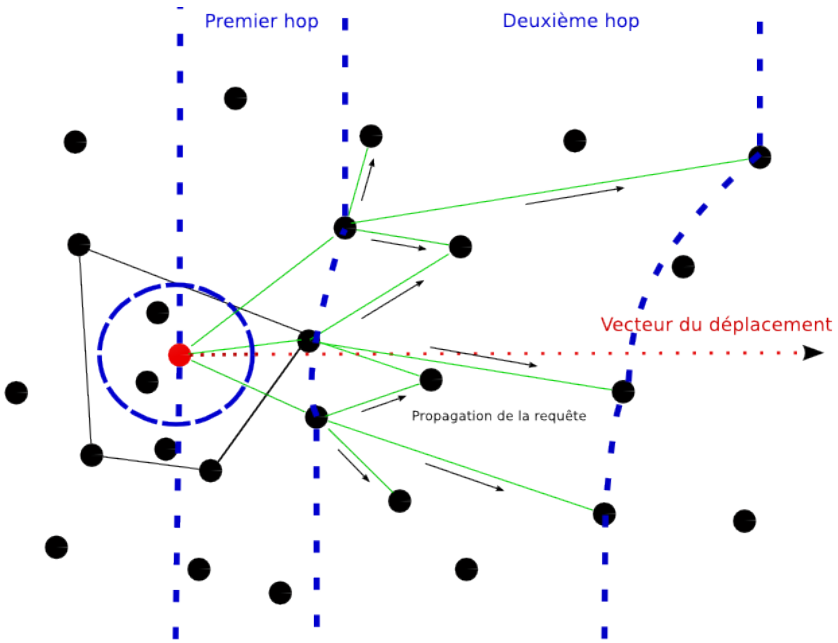

FIG. 8 – *Variante de l'algorithme de propagation : le coefficient d'expansion est fixé à 3. Le TTL est fixé à 10. Chaque nœud répond pour lui* ***et*** *tous ses voisins concernés*

chaque étape de la propagation. Le nombre de successeurs choisis sera par la suite appelé *coefficient d'expansion* de l'algorithme[10]. Dans ce cas, l'expansion se produit plutôt en largeur (cf. figure 8), avec un temps de réponse plus court, car le nombre de nœuds atteints à chaque *hop* croit de manière exponentielle. Il s'agit en quelque sorte d'une inondation dirigée dans le sens du mouvement. Le TTL restant d'un message reçu est alors équitablement divisé parmi tous les successeurs. Par exemple, si le TTL réstant est de 10 avec un coefficient d'expansion égal à 3, chaque successeur recevra un message avec un TTL de 3. Cependant, cette variante est nettement moins efficace en nombre de messages, car elle introduit avec une grande probabilité de redondance dans les réponses. En effet, deux nœuds proches recevant une telle requête auront très probablement de nombreux voisins en commun, qu'ils choisiront tous les deux comme candidats. Une telle variante aura donc comme défaut la génération de messages superflus, mais possède un meilleur temps de réponse, et sera sans doute à privilégier en cas d'hypothèses pouvant admettre des pertes de messages et des pannes.

[10] On pourra alors remarquer que le cas de base décrit précédemment possède un coefficient d'expansion de 1.

Choix des candidats au pré-chargement

À chaque étape de la diffusion de la requête, le nœud courant doit choisir parmi ses voisins un ensemble de candidats dont la position est jugée suffisamment proche de la trajectoire de l'avatar instigateur du pré-chargement. Étant donné le caractère *spéculatif* de cette trajectoire, aucun avatar ne peut avoir la certitude qu'il se trouvera bien sur le chemin de l'instigateur dans le futur. Par conséquent, un algorithme probabiliste nous a semblé adapté pour le choix des candidats : la probabilité pour un nœud d'être pré-chargé est d'autant plus élevée que sa distance angulaire par rapport à l'axe de déplacement est faible.

Plus précisément, soit un nœud N chargé d'évaluer la probabilité $p_{V,A}$ pour nœud V de se trouver sur le chemin d'un avatar A. Considérons le triangle rectangle formé par les coordonnées de V, A[11] et h (h est le projeté orthogonal de V sur la droite dont le vecteur directeur est le vecteur vitesse de A). Ainsi, $p_{V,A}$ dépend de l'angle $\widehat{VAh}$: plus l'angle est aïgu, plus $p_{V,A}$ est grande (cf. figure 9).

De plus, il est nécessaire d'assurer que :

- Si V a déjà été dépassé par A, il ne soit pas choisi ($p_{V,A} \approx 0$).
- Un nœud qui va très bientôt entrer dans la zone d'intérêt de A soit pré-chargé à coup sûr ($p_{V,A} = 1$).

La prise en compte de la distance angulaire dans le calcul de la probabilité est effectuée par l'intermédiaire de la fonction *tangente*. En effet, cette fonction trigonométrique possède les propriétés adéquates :

- Elle mesure le rapport entre le côté opposé et le côté adjacent d'un angle d'un triangle rectangle (et donc indirectement l'angle lui-même).
- Elle vaut 1 en $\frac{\pi}{4}$
- $lim_{x \to \frac{\pi}{2}} \tan x = \infty$

On définit donc :

$$p_{V,A} = \frac{1}{t(\delta \times \widehat{VAh})}, \ avec \ t(x) = \begin{cases} \lceil \tan x \rceil & \text{si } x \in]0, \frac{\pi}{2}[\\ 1 & \text{si } x = 0 \\ 0 & \text{sinon} \end{cases}$$

où V est le nœud candidat, A le nœud instigateur du pré-chargement, et h le projeté orthogonal de la position de V sur la droite symbolisant la trajectoire de V. Quant à δ, il s'agit d'une variable d'ajustement dont l'utilité sera discutée par la suite.

On peut alors remarquer que :

- La probabilité pour un nœud d'être choisi augmente bien avec sa proximité angulaire vis-à-vis de la trajectoire.

[11] Dans la suite du paragraphe, par commodité, on confond les noms des nœuds avec le nom des points ayant pour coordonnées dans le Métavers les coordonnées de ces nœuds.

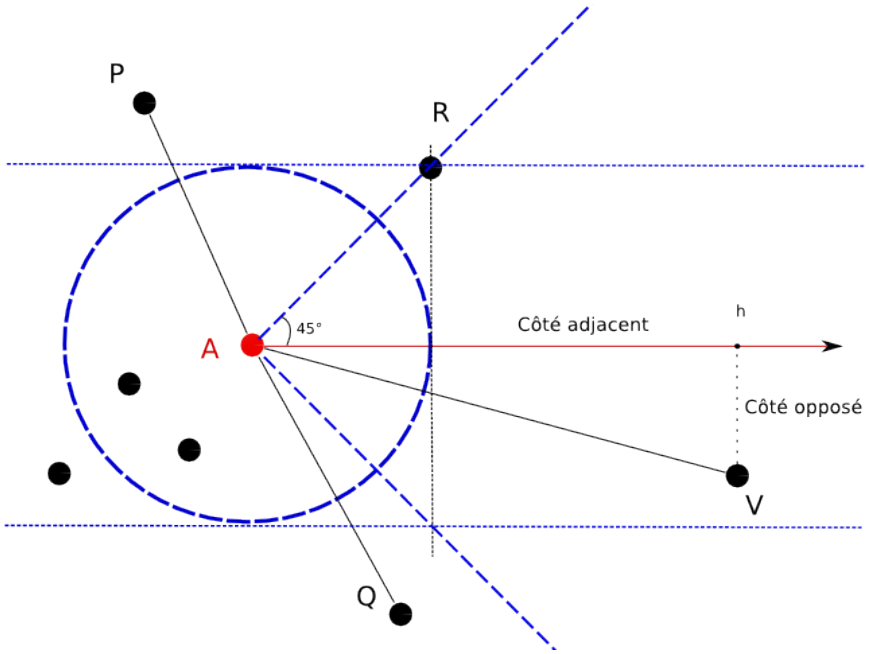

FIG. 9 – *Choix des candidats au pré-chargement : A est en déplacement. V sera choisi, R est à la limite d'être choisi avec certitude, Q n'est pas assuré d'être choisi (mais peut tout de même l'être), P est certain de ne pas être choisi. Le couloir formé par les pointillés bleus représente le déplacement de la zone d'intérêt de A. Tout nœud dont la distance angulaire dépasse 45° ne sera pas dans le couloir en avant de la zone d'intérêt.*

- Si la distance angulaire de V est inférieure à $\frac{\pi}{4}$ (soit 45°), alors $p_V = 1$ et V sera retenu. Cela signifie que tout nœud entrain de pénétrer dans la zone d'intérêt de A sera choisi (cf. figure 9).
- Un nœud dont la distance angulaire se rapproche de $\frac{\pi}{2}$ voit p_V tendre vers l'infini et sera rejeté.

Pour une prédiction plus précise des futurs voisins de l'avatar en déplacement, il est nécessaire de prendre en compte sa vitesse. En effet, il est logique de supposer que la trajectoire sera d'autant plus rectiligne que la vitesse est élevée. C'est pourquoi le pré-chargement doit s'adapter en fonction de la vitesse de déplacement de l'avatar : plus la vitesse est élevée, plus les nœuds pré-chargés doivent être proches de l'axe représentant la trajectoire du déplacement. C'est le rôle de la variable d'ajustement δ. La valeur de cette variable est choisie à chaque pré-chargement par le nœud déclencheur proportionnellement à la vitesse de son avatar. Ainsi, plus la vitesse est élevée, plus l'angle maximal pour lequel $p_{V,A}$ vaut 1 est petit.

Renouvellement des requêtes de pré-chargement

Une fois qu'un avatar en déplacement a effectué une requête de pré-chargement et a reçu une réponse de tous ses futurs voisins, il est possible qu'il aît besoin de propager une nouvelle requête. Cela peut arriver dans le cas où l'avatar est toujours en déplacement, et que tous les nœuds pré-chargés ont été intégrés comme voisins où bien ont été dépassés et sont donc devenus inutiles. À intervalles réguliers le mécanisme de pré-chargement de chaque nœud analyse donc l'état de son avatar selon les quatre conditions suivantes :

1. L'automate régissant le déplacement est dans l'état T (correspondant au mouvement rectiligne et rapide).
2. Le nombre de nœuds déjà pré-chargés ne dépasse pas un certain seuil.
3. Le nœud n'est pas en attente de réponse d'une requête qu'il aurait précédemment lancé.
4. Les voisins se trouvant en avant du mouvement sont trop proches de l'avatar en mouvement.

Le nombre maximum de nœuds pré-chargés est dans nos simulations du même ordre de grandeur que la taille de l'ensemble des voisins (c'est-à-dire entre 10 et 20). Par ailleurs, pour la troisième règle, le mécanisme sait qu'il n'est pas en attente de réponse d'une précédente requête si il n'a pas reçu de réponse durant un temps au moins égal à deux fois le temps de réponse maximal sur le réseau[12]. Quant à la quatrième règle, elle est vraie si tous les voisins dont la distance angulaire par rapport à l'axe du mouvement est inférieure à 90° sont plus proches que d_{min} (cf. définition de d_{min} précédemment énoncée dans cette section) de l'avatar en mouvement. Lorsque les quatre conditions sont remplies, une nouvelle requête de pré-chargement est lancée.

Optimisation de la propagation des mises à jour

Pour tenir son voisinage au courant de ses changements d'état, un avatar doit propager à intervalles réguliers des messages de mise à jour. L'intervalle de ces mises à jour doit être judicieusement choisi, car un intervalle trop petit provoquerait la surcharge du réseau (chaque nœud doit envoyer un message à *tous* ses voisins), et un intervalle trop grand provoquerait des incohérences de vues du Métavers. Les études menées sur le sujet [21] semblant indiquer qu'un délai de 100ms entraine des déviations somme toute acceptables, c'est le délai fixé dans nos simulations pour la propagation des mises à jour. Cependant, il nous a semblé possible de profiter du caractère hautement prédictible des déplacements entre les zones denses pour diminuer le nombre de mises à jour sans significativement impacter la cohérence des vues des avatars. Ainsi, l'intervalle de mise à jour est doublé pour les nœuds dont l'automate est dans l'état T.

Pour contrebalancer le manque d'information dans l'intervalle entre deux mises à jour, un mécanisme basique de prédiction a été ajouté. Ainsi, un nœud qui reçoit des

[12]La manière dont le mécanisme estime ce délai n'est pas précisé. Il peut être fixé *à priori* ou bien calculé dynamiquement.

mises à jour éparses de l'un de ses voisins analyse les métadonnées de ce voisin (position, vitesse, etc.) et en déduit l'évolution de sa position. L'algorithme employé consiste uniquement à calculer le vecteur vitesse de l'avatar en fonction de ses deux positions précédentes, et en déduire la position suivante. Bien que très basique, l'algorithme apporte une amélioration non négligeable, et peut par ailleurs être très simplement amélioré. Il est par exemple possible de calculer l'accélération de l'avatar, ou bien implémenter des modèles comportementaux plus évolués (à l'instar de ce qui est fait dans Colyseus [7]).

5 Évaluation

5.1 Mode opératoire

L'overlay de Solipsis a été implémenté au dessus du simulateur Peersim [11]. Il s'agit d'un simulateur événementiel discret : la simulation se déroule par cycles. Les messages produits durant un cycle sont mis en file d'attente jusqu'à leur date de délivrance. La réception de chaque message provoque une réaction avec possible émission de nouveaux messages, et ainsi de suite jusqu'à la fin de la simulation.

De plus, l'évaluation des performances de Solipsis nécessite l'implémentation d'une simulation de son application : le Métavers. Il peut être vu comme un espace vectoriel en deux dimensions, peuplé d'avatars ayant chacun un ensemble de métadonnées, incluant un identifiant, des coordonnées, ainsi que quelques données sur sa dynamique en tant qu'objet du Métavers (son accélération et sa vitesse maximale). Ces avatars sont libres de déplacement dans le Métavers, ce qui nécessite la conception d'un modèle de mobilité évolué (cf. section 2.2 pour les caractéristiques de ce modèle).

Simulation du Métavers

La surface du Métavers telle qu'elle a été implémentée pour notre évaluation est un plan carré de 3000m de côté peuplé de 1000 avatars. Le nombre d'avatars a été choisi à cause des caractéristiques techniques du simulateur Peersim. En effet, Peersim n'est pas un simulateur réparti, et la charge de la simulation de l'ensemble des nœuds repose donc sur un unique processeur. C'est pourquoi le temps de simulation avec un nombre d'avatars égal à plusieurs milliers est excessivement long et coûteux en mémoire. La taille du Métavers choisie permet d'assurer la densité de peuplement voulue avec 1000 avatars. Bien que cette taille puisse sembler très faible par rapport à l'idée qu'on peut se faire d'un Métavers, comparée à Second Life, la surface simulée représente tout de même plus de 100 parcelles[13]. De plus, il s'agit d'une surface "sans bord", c'est à dire que les coordonnées des avatars évoluent *modulo* la taille du Métavers.

Notre modèle de distribution simule la présence de plusieurs zones denses séparées par des zones de faible densité. La distribution comprend deux types de zones denses :

[13] Chaque parcelle de Second Life est gérée par un serveur et représente une surface de 256×256 mètres [13].

FIG. 10 – *Capture d'écran de la distribution du Métavers avec 3 zones denses peuplées par une loi gaussienne. Chaque point représente un avatar.*

les petites zones denses et les grandes zones denses. L'ensemble des caractéristiques de notre distribution est paramétrable, ce qui permet de contrôler :

- Le nombre de zones denses de chaque type.
- La taille des zones de chaque *type* (et donc y atteindre la densité voulue).
- Le nombre d'avatars qui n'appartiennent à aucune zone.

En revanche, le placement des zones dans le Métavers ainsi que celui des avatars dans chaque zone est totalement aléatoire. La distribution des avatars au sein de chaque zone dense suit ainsi une loi gaussienne (cf. figure 10). Le modèle de distribution ainsi obtenu permet de simuler un véritable Métavers, avec ses zones de forte attraction (comme des villes) et ses déserts (océans, forêts, etc.). Le nombre de larges pôles d'attraction est fixé à 3 dans nos simulations (au delà, la densité intra-zone devient plus faible avec 1000 nœuds). Le nombre de petites zones denses est lui fixé à 10 (les petites zones ont une taille 3 fois inférieure aux grandes, et sont sensiblement moins denses).

Modèle de mobilité

Une fois élaboré le modèle de distribution du Métavers, il est nécessaire de mettre en place le modèle de mobilité décrit précédemment. De plus, il est primordial que la distribution initiale soit conservée tout au long de la simulation malgré la mobilité des avatars. En outre, on définit le ***degré de mobilité*** du Métavers à un instant donné (en

‰) comme le nombre d'avatars dans l'état T de l'automate à cet instant sur le nombre total d'avatars.

Le modèle de mobilité implémenté dans nos simulations se base sur l'automate décrit précédemment (cf. figure 6). Un ensemble de probabilités est associé aux transitions de l'automate (cf. table 5.1). À intervalles réguliers (100ms dans notre évaluation), Peersim décide pour chaque avatar en fonction de ces probabilités si il doit changer de transition (et donc la nature de son mouvement), ou bien continuer son déplacement.

À chaque fois qu'un déplacement est décidé (indépendamment de sa nature), une destination est choisie dans le Métavers, et le mouvement est initié. Les avatars possèdent une accélération, ce qui fait que chaque mouvement est rectiligne, mais pas forcement uniforme. Lorsque la vitesse maximale est atteinte, l'accélération est mise à zéro et l'avatar entame un déplacement uniforme. La vitesse maximale est fixée dans nos simulations à 4 $m.s^{-1}$ pour le déplacement chaotique, et 100 $m.s^{-1}$ pour le déplacement rectiligne entre les zones denses. La première correspond approximativement à la vitesse d'un coureur, alors que la deuxième, nécessaire aux déplacements rapides au sein du Métavers, est raisonnable pour un objet en vol ou un véhicule terrestre à grande vitesse.

L'algorithme de choix des destinations est conçu pour ne pas perturber la distribution initiale : la probabilité pour une destination de se situer à l'intérieur d'une zone dense est plus élevée que pour une zone déserte. De plus, à l'intérieur d'une zone dense, les destinations sont choisies selon la même loi gaussienne que lors de leur création. Il en résulte que malgré la forte dynamique du Métavers, sa distribution initiale constitue à peu de choses près un invariant durant toute la simulation.

Le changement des probabilités sur les transitions entraîne le changement du comportement des avatars. L'automate étant mis à jour toutes les 100ms, même un faible changement dans les probabilités peut entraîner une évolution radicale de la dynamique du Métavers. Pour l'évaluation, nous avons fait varier la probabilité de la transition $WToT$ de l'automate pour faire évoluer le degré de mobilité du Métavers. La transition $WtoT$ déclenche le passage d'un avatar d'un état d'exploration désordonnée à un état de translation rapide. Plus cette probabilité est élevée, plus le nombre d'avatars en déplacement rapide est grand. La figure 11 montre la correspondance (expérimentalement obtenue) entre la probabilité sur la transition $WtoT$ et le degré de mobilité du Métavers. Ainsi, lorsque la probabilité pour un automate dans l'état W de passer dans l'état T est de 0.002, très peu d'avatars (à peine 10‰) sont en déplacement rapide, ce qui correspond à un Métavers plutôt statique. À l'opposé, lorsque la probabilité associée à $WtoT$ atteint 0.05, près d'un avatar sur 5 est en mouvement accéléré. Il s'agit du degré de mobilité maximal, car des valeurs plus élevées génèrent trop de messages nécessaires à la maintenance de l'overlay, ce qui provoque un dépassement mémoire au niveau du simulateur. De toute manière, 1 avatar en déplacement rapide pour 5 en déplacement lent constitue un Métavers déjà très dynamique, certainement plus que ce qui est observé dans la plupart des Mondes Virtuels multijoueurs existants.

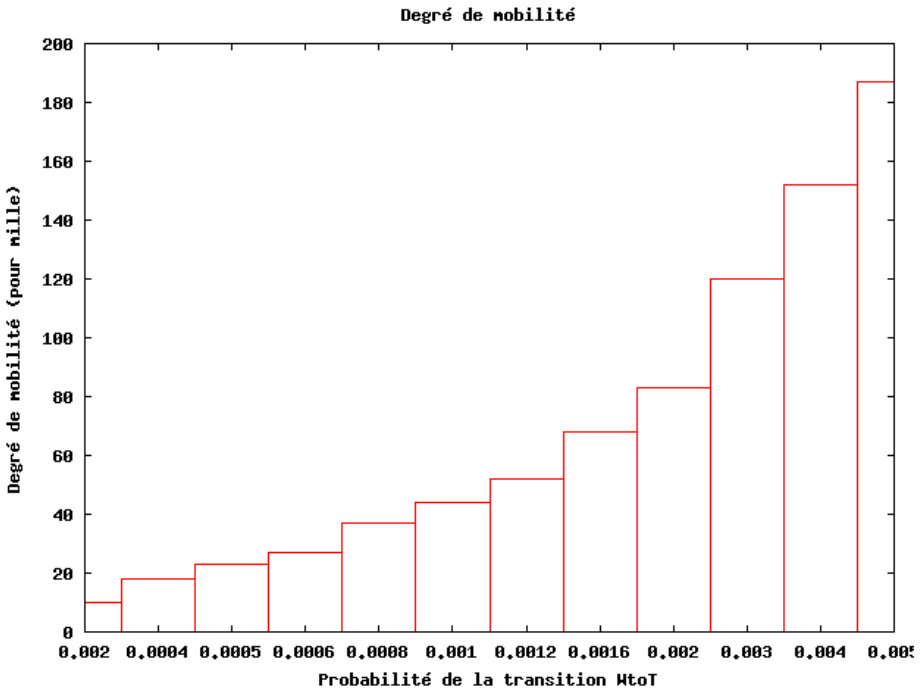

FIG. 11 – *Influence du changement de probabilité sur la transition WtoT de l'automate sur le degré de mobilité du Métavers*

Transition	Probabilité	État courant	Description
HtoH	0.85	Repos	Rester à l'arrêt
HtoT	0.0004	Repos	Commencer à voyager
HtoW	0.1496	Repos	Commencer à explorer
WtoH	0.01-x	Exploration	S'arrêter
WtoT	x	Exploration	Commencer à voyager
WtoW1	0.8	Exploration	Continuer à explorer dans la même direction
WtoW2	0.19	Exploration	Changer de direction d'exploration
TtoH	0.0002	Voyage	S'arrêter
TtoW	0.0005	Voyage	Commencer à explorer
TtoT	0.9993	Voyage	Continuer à voyager

TAB. 1 – *Tableau descriptif de l'automate de déplacement avec les probabilités associées à chaque transition. Les probabilités sont évaluées toutes les 100ms. x varie entre 0.0002 et 0.005 (cf. figure 11)*

Analyse probabiliste du modèle de mobilité

Le fort écart de probabilité des transitions s'explique par la fréquence de leur évaluation. Ainsi, un écart trop faible entre les transitions de changement d'état et celles qui concernent la conservation d'un état entraînerait une trop grande versatilité comportementale de l'avatar. En effet, considérons un avatar dans l'état W avec une probabilité de 0.0002 associée la transition $WtoT$. À chaque évaluation, soit $p = 0.0002$ la probabilité pour l'avatar de passer dans l'état T. De plus, notons $q = 1 - p = 0.9998$. L'évaluation correspond à une épreuve de Bernoulli de paramètre p, avec une probabilité de réussite très faible (0.0002). Ainsi, la répétition des épreuves de Bernoulli que constitue le modèle de mobilité du Métavers forme un schéma de Bernoulli. La probabilité de succès après n épreuves est alors de :

$$\boxed{P_n = p \times \sum_{k=1}^{n} q^k}$$

Il en résulte que même dans le cas où p=0.0002 (degré de mobilité le plus faible), la probabilité que l'avatar passe dans l'état T est supérieure à 0.5 après seulement 3465 cycles, soit environ 5 minutes et 48 secondes. Cela veut dire qu'un avatar en état d'-exploration dans une zone dense aura plus de 50% de chance de repartir en voyage après seulement 6 minutes, ce qui est très raisonnable pour une simulation réaliste du comportement.

Construction de l'overlay

Il convient maintenant de s'attarder sur la formation de l'overlay de support du Métavers. L'ensemble des algorithmes de maintenance de Solipsis ont été implémentées dans Peersim. En début de simulation, un algorithme de formation de la topologie est mis en route[14]. Une fois la topologie formée par notre algorithme, les protocoles de maintenance de Solipsis sont enclenchés, et la topologie de l'overlay converge vers un état stable. Cette convergence prend un temps non négligeable (quelques dizaines de secondes) et peut perturber les mesures, c'est pourquoi toutes les mesures commencent 100 secondes après l'enclenchement de la simulation.

Afin de surveiller la formation de la topologie, le respect des règles, ainsi que pour le debogage, un lecteur de traces a été mis au point (cf. figure 12. Il est alors possible, à chaque simulation, d'enregistrer toutes les données utiles sur l'overlay. Le fichier ainsi obtenu peut ensuite être interprété par le lecteur de traces, ce qui permet entre autres de vérifier le respect des regles sur chaque nœud de la topologie, et de leur évolution dans le temps (il est par exemple possible de mettre le Métavers en pause, revenir en arrière dans le temps, avancer, cliquer sur un avatar pour obtenir ses métadonnées, etc.).

Une fois la topologie stabilisée, les mesures sont effectuées à chaque cycle durant une période de 1000 cycles (soit 100 secondes). Cette procédure est répétée 10 fois, et une

[14] En effet, simuler l'arrivée simultanée de 1000 nœuds selon les régles de Solipsis serait trop long : il s'agit donc d'un algorithme dédié moins coûteux.

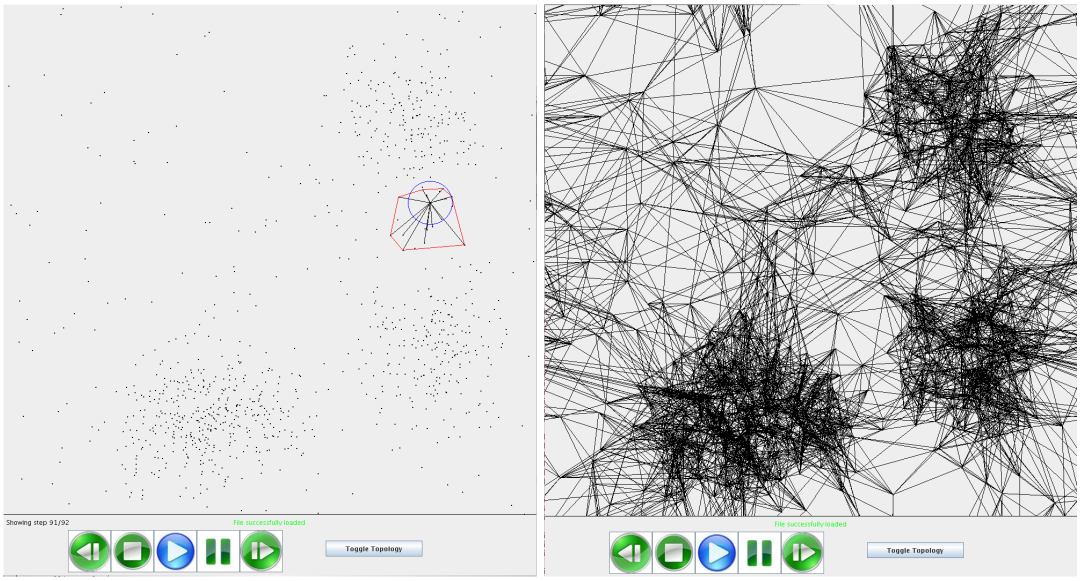

FIG. 12 – *Copies d'écran du lecteur de traces.* ***À gauche*** *: distribution du Métavers avec affichage des métadonnées (enveloppe convexe et zone d'intérêt) d'un avatar particulier.* ***À droite*** *: la topologie de Solipsis dans son ensemble*

moyenne de l'ensemble des mesures est calculée.

5.2 Résultats

Métriques utilisées

L'évaluation de notre mécanisme de pré-chargement est effectuée selon cinq paramètres :

1. **Maintien de la cohérence du Métavers** : permet de mesurer le nombre d'incohérences dans les vues du Métavers. On considère qu'il y a incohérence lorsqu'un nœud ne se trouve pas à l'intérieur de l'enveloppe convexe de ses voisins (violation de la règle de l'enveloppe convexe de Solipsis) ou bien qu'il existe un avatar situé dans sa zone d'intérêt avec lequel il n'a pas de lien de voisinage (violation de la règle de la zone de connaissance de Solipsis).
2. **Nombre de messages échangés** : permet de mesurer la charge réseau totale par cycle. Tous les messages sans distinction de type sont comptabilisés.
3. **Divergence moyenne entre les coordonnées des avatars vues par leur voisinage et les coordonnées réelles** : il existe toujours une divergence entre les coordonnées réelles d'un avatar à un instant donné, et les coordonnées que possède le voisinage de l'avatar. Ce décalage est du à la latence de propagation des messages de mise à jour. Dans notre cas, il est important de mesurer la divergence des coordonnées, car il faut quantifier l'efficacité du mécanisme de prédiction de

trajectoire (qui permet de prédire l'évolution des coordonnées d'un avatar à court terme en absence de mise à jour).

4. **Nombre moyen de nœuds voisins situés en avant de la trajectoire d'un avatar en mouvement** : Un avatar en déplacement doit constamment obtenir des données en avant de sa trajectoire afin de mettre à jour sa vue du Métavers. Plus le nombre de voisins situé en avant de sa trajectoire est grand, plus cette mise à jour sera répartie entre les nœuds et va se faire rapidement.
5. **Temps de connexion moyen d'un avatar en mouvement avec les nœuds situés sur sa trajectoire** : Cette métrique mesure l'ancienneté des contacts de l'avatar avec ses voisins situés en avant de sa trajectoire. Plus l'ancienneté des contacts est élevée, plus la taille des données transmises par anticipation à l'avatar est importante. Ainsi, un avatar qui possède un voisin dans la zone d'arrivée peut commencer à pré-charger les données relatives à la zone bien avant son arrivée effective. La durée est calculée à partir de l'ajout de du nœud dans l'ensemble des voisins jusqu'à son dépassement par l'avatar en mouvement, ou bien sa suppression de l'ensemble de voisinage.

L'évaluation des cinq métriques en fonction du dégré de mobilité du Métavers est proposée ci-dessous. Les systèmes évalués sont :

- La version de base de Solipsis
- Solipsis avec mécanisme de pré-chargement avec un coefficient d'expansion de 1 pour la propagation des requêtes (cf. section 4.2).
- Le même mécanisme de pré-chargement avec un coefficient d'expansion de 4.

Mesures effectuées

Rappelons qu'un coefficient d'expansion supérieur à 1 signifie qu'une requête de pré-chargement sera à chaque *hop* propagée à plusieurs nœuds dont les avatars se situent en aval du mouvement.

La figure 13 présente l'évolution de de la cohérence du Métavers (régle 1). On peut constater que l'algorithme de pré-chargement réduit nettement le nombre d'incohérences liées à la mobilité des avatars. En effet, le nombre d'incohérences constatées pour les degrés de mobilité moyens (entre 8 et 25 ‰) est entre 2 et 3 fois inférieur avec le mécanisme de pré-chargement. Cela signifie qu'en présence du pré-chargement, un avatar situé à l'intérieur d'une zone d'intérêt d'un autre avatar sera voisin de cet avatar avec une probabilité beaucoup plus élevée que sans pré-chargement. De plus, le mécanisme améliore grandement le respect de la règle de l'enveloppe convexe de Solipsis, facilitant le fonctionnement la collaboration spontanée entre les nœuds. Bien que l'amélioration a tendance à s'amoindrir avec l'augmentation du degré de mobilité, elle reste perceptible même pour le degré de mobilité maximal.

En revanche, l'augmentation du coefficient d'expansion n'influe quasiment pas sur la cohérence. Ceci est probablement dû à une forte redondance dans les réponses au pré-chargement : la requête est propagée à plusieurs nœuds simultanément. Ces nœuds choisissent les candidats au pré-chargement parmi leurs voisins avant d'en retourner

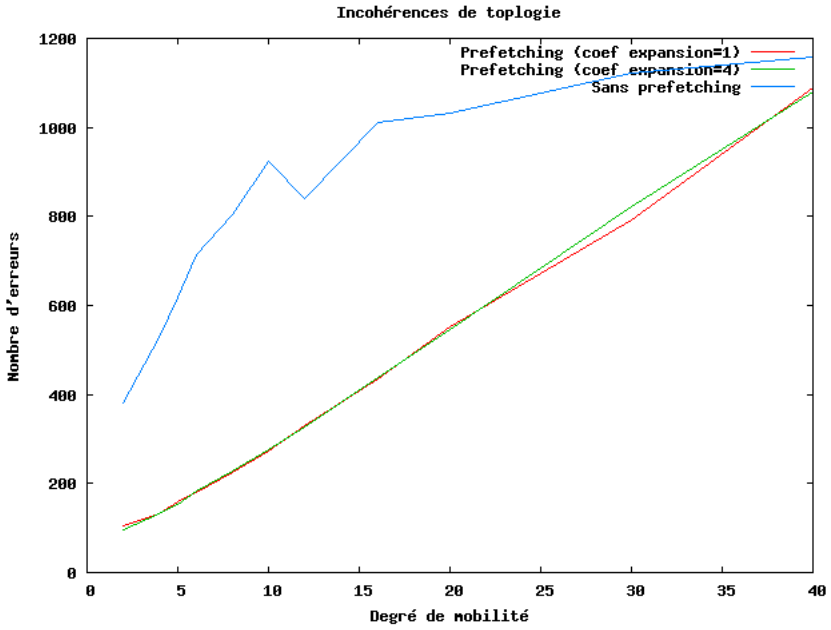

FIG. 13 – *Évolution du nombre global d'incohérences en fonction du degré de mobilité (métrique n^o 1)*

la liste au nœud source. Cependant, les nœuds situés en avant du déplacement auront probablement un nombre important de voisins communs. Il en résulte que l'information globalement retournée ne diffère sensiblement pas d'une requête propagée avec un coefficient de 1.

La figure 14 décrit l'évolution de la divergence des vues du Métavers (métrique n^o 2). On constate donc que la divergence moyenne est pratiquement doublée à cause du manque de mises à jour. Ceci est probablement imputable au caractère simpliste de l'algorithme de prédiction du mouvement (en effet, il ne prend même pas en compte l'accélération), et peut donc facilement être amélioré. De plus, la vitesse maximale (rapidement atteinte) d'un avatar en déplacement rapide étant de $100m.s^{-1}$ dans nos simulations, une divergence de vues de quelques dizaines de mètres au plus n'est pas si significative.

L'augmentation de la divergence avec le coefficient d'expansion est sans doute due au fait qu'en cas de redondance des requêtes (ie. un nœud est choisi plusieurs fois en tant que candidat), c'est sa position la plus obselète qui est retenue (car c'est la réponse qui arrive en dernier). Ce défaut peut donc facilement être corrigé en ne retenant que la première réponse.

La figure 15 montre l'évolution du nombre moyen de messages échangés par un nœud par période de 100ms. On constate sans surprise une croissance quasi-linéaire du nombre de messages échangés en fonction du degré de mobilité. De plus, le pré-chargement (avec

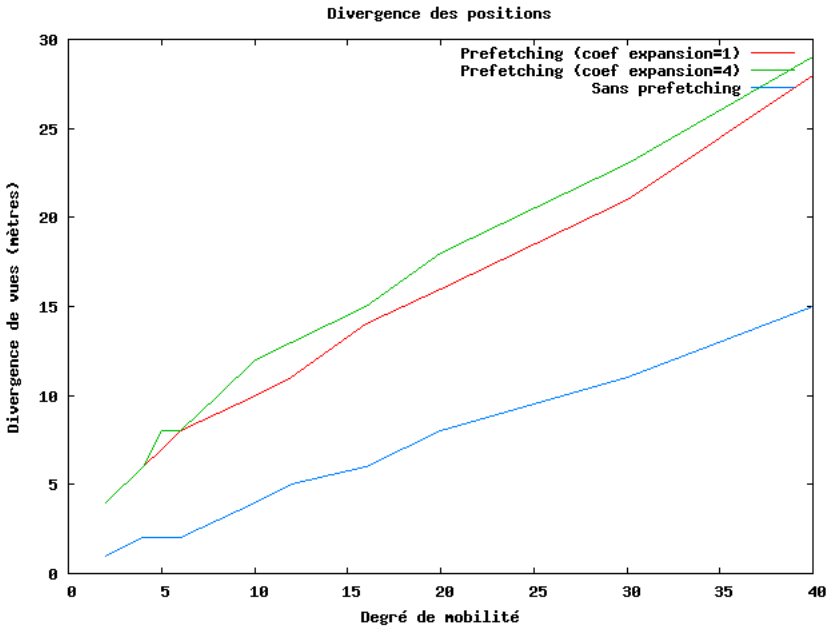

FIG. 14 – *Accentuation des divergences entre les positions des avatars vues par leur voisinage et leurs coordonnées réelles*

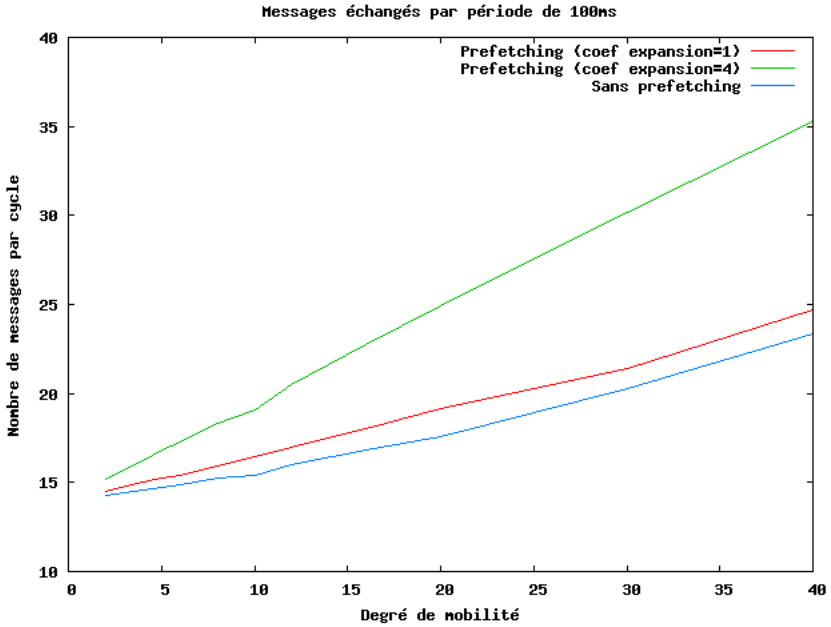

FIG. 15 – *Nombre de messages échangés (tous types confondus) en moyenne par un nœud par cycle de 100ms*

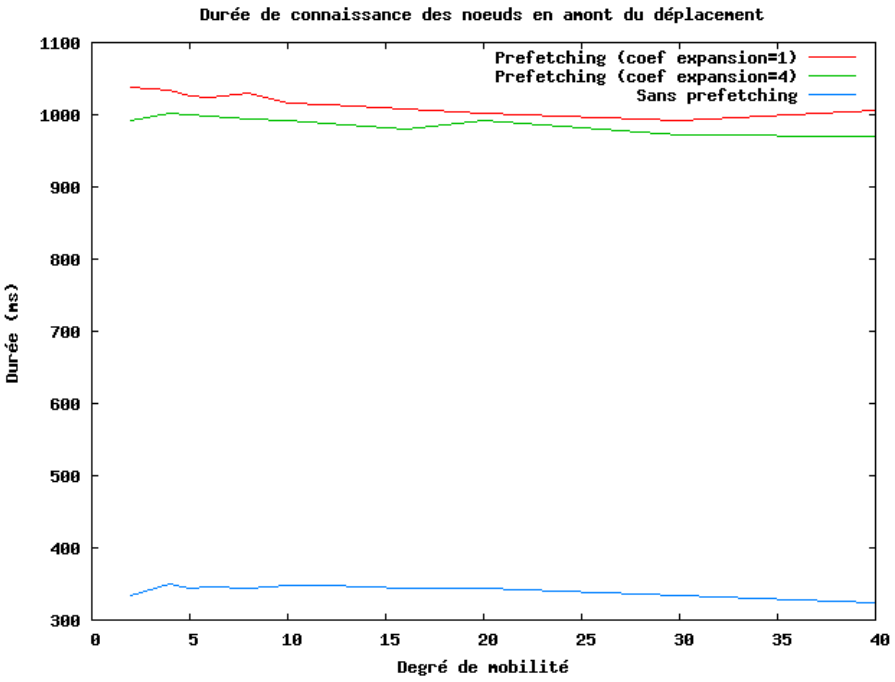

FIG. 16 – *Durée de connaissance des nœuds voisins situés en avant sur la trajectoire d'un avatar en déplacement*

un coefficient d'expansion égal à 1) n'apporte qu'un très faible surplus de messages, et n'influe donc que très peu sur la charge totale du réseau. En effet, le surplus est de l'ordre de 1 message par cycle (10 $msg.s^{-1}$) *quelque soit le degré de mobilité*, avec des échanges concernant uniquement la maintenance (ne contenant donc aucune donnée volumineuse). Enfin, il est possible de réduire encore la fréquence de mise à jour des avatars ayant des trajectoires linéaires[15], réduisant encore le nombre de messages transmis.

En revanche, le pré-chargement avec un coefficient d'expansion de 4 manifeste un accroissement significatif des échanges réseau.

L'évaluation du mécanisme de pré-chargement selon la métrique n° 5 montre que la durée moyenne de connaissance des nœuds situés en avant du mouvement d'un avatar est, avec un mécanisme de pré-chargement, environ 3 fois supérieure (figure 16) à la durée constatée avec le Solipsis de base. Il se situe en effet autour d'une seconde pour la version avec pré-chargement, pour un peu plus de 300ms pour la version sans optimisations. On constate par ailleurs que l'augmentation du coefficient d'expansion nuit légèrement à la durée de connaissance. Cela se produit car la propagation des requêtes se fait en largeur, et les nœuds ainsi pré-chargés seront donc plus vite dépassés par l'avatar en mouvement.

[15] en utilisant des mécanismes de prédiction de trajectoire évolués, comme le fait Colyseus [7].

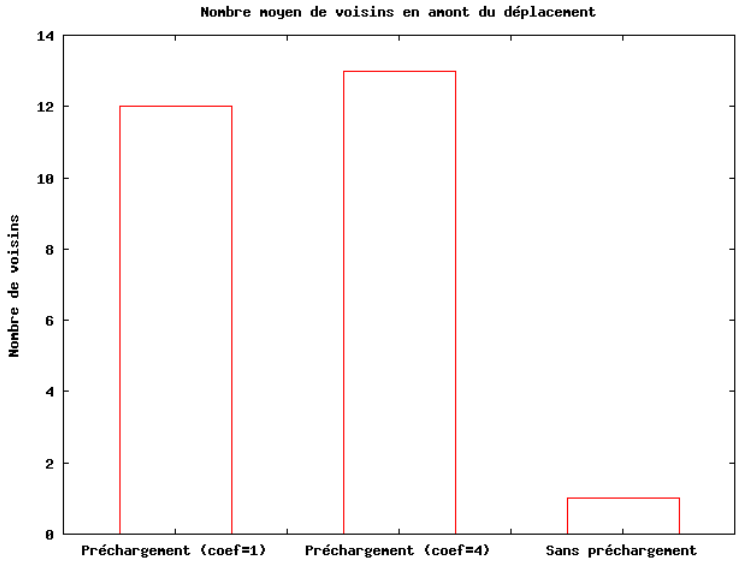

FIG. 17 – *Nombre de nœuds voisins situés en avant sur la trajectoire d'un avatar en déplacement*

Le nombre de nœuds situés en avant du mouvement de l'avatar en mouvement est lui aussi très largement supérieur avec le mécanisme de pré-chargement (figure 17). Il est en effet en moyenne de 12 ou 13 avec le pré-chargement contre 1 seulement pour Solipsis sans optimisation, indépendamment du degré de mobilité. Par conséquent, le nombre de sources pour le chargement de données an avant du déplacement de l'avatar est beaucoup plus important.

Conjugué à la durée de connaissance de ces nœuds presque 3 fois plus importante, ces deux propriétés permettent à l'application de pré-charger une quantité conséquente de données en avance sur le déplacement de l'avatar, réduisant très fortement la latence apparente du Métavers.

Il est intéressant de remarquer que ce nombre est à rapprocher de la quantité maximale de nœuds à pré-charger autorisée (fixée à 15 dans nos simulations). Cela signifie que l'algorithme de renouvellement des requêtes décrit en 4.2 (§ "Renouvellement des requêtes de pré-chargement") fonctionne bien et permet de réitérer à temps les requêtes au fur et à mesure du déplacement. Il est par conséquent possible d'accroitre ce nombre en fixant la quantité maximale de nœuds à précharger à une valeur supérieure (bien que cela entraînerait une charge réseau plus élevée).

Analyse

L'anticipation de la mobilité est un moyen très efficace pour améliorer l'adaptativité de l'overlay de Solipsis. Il s'avère en effet que grâce à notre mécanisme de pré-chargement, la cohérence du Métavers est grandement renforcée. On constate ainsi que le nombre

d'incohérences dues au retard d'adaptation de l'overlay est réduit au moins de moitié.

De plus, l'anticipation du mouvement permet à l'avatar en déplacement de rapidement disposer d'un nombre conséquent de sources situées en avant du mouvement et ainsi charger en avance les données concernant la portion de Métavers concernée. En effet, l'experimentation montre qu'avec notre mécanisme, l'avatar en mouvement dispose de sources en moyenne trois fois plus longtemps que sans pré-chargement. De plus, le nombre moyen de sources est paramétrable et peut facilement dépasser d'un facteur 10 le nombre de sources présent dans la version de base de Solipsis. Une rapide estimation permet de juger de la bande passante disponible dans les deux cas pour le nœud en charge de l'avatar en mouvement. Supposons que chaque source possède un débit de 1 $Mbit.s^{-1}$. Une source est disponible en moyenne durant un délai de 300ms environ avant d'être dépassée dans la version de base de Solipsis. Un avatar en déplacement peut donc esperer pouvoir charger environ 300Kbit (soit 37,5Ko) depuis cette source. Le mécanisme de pré-chargement met à disposition de l'avatar en déplacement en moyenne 12 sources pour une durée moyenne de 1000ms environ. L'avatar dispose ainsi de 12Mbit (soit 1500Ko) de données chargeables en avant du déplacement. Ce surplus plutôt conséquent de données est grandement bénéfique à l'application, qui pourra le mettre à profit pour présenter le Métavers à l'avatar avec bien plus de précision et de cohérence.

Par ailleurs, l'évaluation a permis de mettre en évidence que ce gain non négligéable des performances de l'overlay est réalisé sans surcharge importante du réseau. En effet, le surcoût en nombre de messages est peu élevé (inférieur à 10%) *quelque soit le degré de mobilité*. De plus, ce surcoût peut encore être réduit en utilisant des algorithmes de dead-reckoning avancés (et donc en espaçant encore un peu les messages de mise à jour).

Enfin, il est à noter que la variation du degré de propagation de l'algorithme de pré-chargement a permis de mettre en évidence que le degré le plus efficace est 1. Ainsi, le schéma de propagation exponentiel est plutôt inéfficace avec les hypothèses prises dans nos experiences. En effet, la diffusion de la requête à plusieurs voisins simultanément mène à une augmentation conséquente du nombre de messages (la charge réseau est ainsi presque doublée), pour un résultat quasiment égal à l'algorithme non-exponentiel.

6 Perspectives

Les performances du mécanisme de pré-chargement mises en évidence dans ces travaux en font une composante indispensable pour tout overlay malléable en charge du support d'un Métavers. Toutefois, de nombreux travaux doivent encore menés. Il est par exemple nécessaire d'élaborer un algorithme efficace et précis de prédiction de la trajectoire des avatars. On pourrait ainsi se baser sur les travaux effectués pour Colyseus, qui permettent d'espacer les mises à jour de plusieurs centaines de millisecondes (jusqu'à une seconde) sans défaut perceptible par l'utilisateur. Le gain en nombre de messages pourrait alors être réutilisé pour effectuer un pré-chargement plus important. Il serait par ailleurs utile passer à un modèle avec pannes et rendre l'algorithme tolérant aux fautes.

Il serait également intéressant d'implémenter un cache de voisinage, et de tester différentes politiques de gestion de cache dédiées aux mondes virtuels (telles que MRM,

qui n'a été exploité qu'avec un modèle client-serveur). Un tel cache serait en particulier très utile dans les zones à forte densité du Métavers, avec des avatars ayant une trajectoire désordonnée.

Enfin, dans un cadre plus général, la conception d'overlays fortement malléables reste un problème ouvert. La prise en compte de la mobilité au sein de Solipsis a en effet permis de mettre en lumière les bienfaits d'une anticipation de la dynamique du Métavers. Il s'agit cependant d'un cas particulier assez simple. Il conviendrait donc d'en tirer les enseignements nécessaires pour élargir le cadre d'étude et généraliser l'approche à l'ensemble des applications qui nécessitent une forte malléabilité de la part de leur overlay. Il faudrait donc concevoir des procédés efficaces pour déceler les tendances de la dynamique applicative et modifier la topologie en conséquence. Ces tendances seraient alors définies comme caractéristiques de l'application, ou bien déduites au fur et à mesure de son cycle de vie par les algorithmes de l'overlay. Il sera certainement nécessaire de prendre en compte des tendances globales, concernant un sous-ensemble des participants de l'application (comme l'estimation de l'évolution de la densité ou de la dynamique dans une zone donnée), ou bien locales, concernant le changement d'état de chaque entité de l'espace applicatif. Un overlay capable de prévoir assez précisément l'ensemble de ces tendances applicatives serait performant et adaptatif sans commune mesure avec les overlays existants, permettant l'essor du modèle pair-à-pair parmi les applications émergentes.

Conclusion

Ce document a présenté un mécanisme de prise en compte de la mobilité logique dans un Environnement Virtuel pair-à-pair particulièrement dynamique : le Métavers. L'ensemble des algorithmes proposés permettent la prédiction de la trajectoire des avatars du Métavers dans le but d'anticiper le mouvement et adapter en conséquence la topologie pair-à-pair. L'intégration de ce mécanisme à l'overlay de Solipsis via le simulateur Peersim a mis en évidence ses atouts. Son évaluation montre en effet qu'il permet d'accroitre sensiblement l'adaptabilité de l'overlay à la mobilité des avatars du Métavers. De plus, cette amélioration est réalisée sans augmentation significative du nombre de messages, prouvant définitivement l'utilité d'une telle approche. Cependant, de nombreux travaux, dont la généralisation de l'approche à l'ensemble des Environnements Virtuels, restent encore à réaliser.

Références

[1] O. Beaumont, A.-M. Kermarrec, L. Marchal, E. Rivière. Voronet : a scalable object network based on Voronoi Tessellations., in : Proceedings of 21st IEEE International Parallel & Distributed Processing Symposium (IPDPS)., Long Beach, CA, USA, March 2007

[2] D. Frey, J. Royan, R. Piegay, A.-M. Kermarrec, E. Anceaume, F. Le Fessant. Solipsis : A Decentralized Architecture for Virtual Environments. The 1st International Workshop on Massively Multiuser Virtual Environments (MMVE), 2008.

[3] R. Morales, S. Monnet, I. Gupta, G. Antoniu. MOve : Design and Evaluation of a Malleable Overlay for Group-Based Applications. IEEE Transactions on Network and Service Management, September 2007.

[4] J. Keller and G. Simon. Solipsis : A Massively Multi-Participant Virtual World. In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, NV, 2003.

[5] O. Beaumont, A.-M. Kermarrec, E. Rivière. Peer to Peer Multidimensional Overlays : Approximating Complex Structures. In OPODIS, 11th International conference on principles of distributed systems, 2007.

[6] J. Kleinberg. The Small-World Phenomenon : An Algorithmic Perspective. In : Proceedings of the 32nd ACM Symposium on Theory of Computing, Portland, OR, USA, pp. 163-170, May 2000.

[7] A. Bharambe, J. Pang, S. Seshan. Colyseus : A Distributed Architecture for Online Multiplayer Games. In : Proc. Synposium on Networked Systems Design and Implementation (NSDI), March 2006.

[8] A. Bharambe, J.R. Douceur, J. Lorch, T. Moscibroda, J. Pang, S. Seshan, X Zhuang. Donnybrook : Enabling Large-Scale, High-Speed, Peer-to-Peer Games. In : SIGCOMM'08, Seattle, Washington, USA, August 2008.

[9] A. Bharambe, M. Agrawal, S. Seshan. Mercury : Supporting Scalable Multi-Attribute Range Queries. In : SIGCOMM, Portland, OR, August 2004.

[10] S. Park, D. Lee, M. Lim, C. Yu. Scalable Data Management Using User-Based Caching and Prefetching in Distributed Virtual Environments. In : Proceedings of the ACM synposium on Virtual Reality Software and Technology (VRST), Baniff, Alberta, Canada, 2001.

[11] http ://peersim.sourceforge.net/

[12] C.-A. La, P. Michiardi. Characterizing User Mobility in Second Life. Technical Report RR-08-212, Institut Eurecom, August 2008.

[13] H. Liang, I. Tay, M. F. Neo, W. T. Ooi and M. Motani. Avatar Mobility in Networked Virtual Environments : Measurements, Analysis, and Implications. Technical Report, National University of Singapore, Singapore, 2008.

[14] B. Bollobas, Random Graphs, Second Edition. Cambridge University Press, 2001.

[15] F. Aurenhammer, Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure" ACM Computing Surveys 23(3) :345-405, Sept. 1991.

[16] L.A. Adamic, B.A. Huberman. Zipf's law and the internet. Glottometrics 3. 2002.

[17] D. Pittman and C. GauthierDickey. A measurement study of virtual populations in massively multiplayer online games. In Proc. of NetGames '07, pages 25-30, Melbourne, Australia, 2007.

[18] J. Chim, R. Lau, A. Si, H. Leong, D. To, M. Green, M. Lam. Multi-Resolution Model Transmission in Distributed Virtual Environments. In Proceedings of ACM Virtual Reality Software Technology (VRST'98), pages 25-34, Taipei, Taiwan, Nov. 1998.

[19] H. Hoppe. Progressive Meshes. In ACM Computer Graphics (SIGGRAPH'96), pages 98-108, August 1996.

[20] J. Chim, R. Lau, H. Leong, A. Si. Multi-Resolution Cache Management in Digital Virtual Library. In Proceedings of IEEE Advances in Digital Librairies Conference, pages 66-75. April 1998.

[21] L. Pantel, L.C. Wolf. On the suitability of dead reckoning schemes for games. In NetGames, pp. 79-84. April 2002.

[22] Statistiques Internet : http ://www.internetworldstats.com/stats.htm

[23] Neal Stephenson, Snow Crash, 1992.

[24] World Of Warcraft : http ://www.worldofwarcraft.com

[25] Second Life : http ://www.secondlife.com