Amélioration de la réactivité des réseaux pair à pair pour les MMOGs

Étude du code existant de Blue Banana

Xavier Joudiou

Encadré par: Sergey Legtchenko & Sébastien Monnet

1/06/10

Table des matières

Résumé

Depuis plusieurs années, un nouveau type d'architecture des systèmes est apparu. Il s'agit de l'architecture pair à pair, cette architecture est devenue populaire grâce à des applications de partage de fichiers. Nous allons nous intéresser aux jeux vidéos massivement multijoueur (MMOG pour Massively Multiplayer Online Games) qui sont de plus en plus populaires et qui font ressortir des problèmes que l'architecture pair à pair doit pouvoir corriger. Le problème du passage à l'échelle sera l'un des plus importants à résoudre pour permettre à un grand nombre de joueurs de participer simultanément. Nous verrons comment l'architecture pair à pair peut être une des solutions.
Pour remédier à cela, une solution consiste à remplacer le modèle client/serveur par un réseau logique pair à pair (overlay). Malheureusement, les protocoles pair à pair existants sont trop peu réactifs pour assurer la faible latence nécessaire à ce genre d'applications. Néanmoins, quelques travaux ont déjà été menés pour adresser ce problème. L'idée est d'adapter le voisinage de chaque pair afin que toute l'information dont il aura besoin dans l'avenir se trouve proche de lui dans le réseau. Il est alors nécessaire de correctement évaluer les futurs besoins de chaque pair, et de faire évoluer son voisinage à temps. Dans ce rapport bibliographique, nous expliquerons les différentes solutions existantes.

1 Introduction

Dans ce document, nous allons regrouper les différentes observations et les descriptions du code existant de Blue Banana [1], afin de faciliter les recherches et les modifications de celui-ci. Blue Banana est un mécanisme d'anticipation des mouvements des avatars dans les MMOG (Massively Multiplayer Online Game) utilisant une architecture pair à pair. Ce mécanisme d'anticipation permet aux nœuds d'avoir les données qui serviront aux prochaines itérations à disposition. Les performances du jeu, dans certaines conditions, se verront être améliorées de façon significative sans trop altérer le réseau.

Nous identifierons les fichiers et les fonctions importantes, et si nécessaire nous expliquerons quelques unes de ces fonctions. Tous les fichiers, que nous allons voir se trouve dans le répertoire *VirtualMobility/AMAM/simu/SolipsisPeersim*. Nous allons commencer par le fichier de configuration, ensuite nous verrons le fichier *SolipsisProtocol*, et plus précisément les différents traitements de la fonction *Receive*.

2 Le fichier de configuration

Le fichier de configuration va nous permettre de configurer comme nous le souhaitions les expérimentations, celui qui nous intéresse est : *params_CURRENT_2010.cfg*. Ce fichier de configuration se trouve dans le répertoire *Config*, d'autres fichiers de configuration sont présents dans ce répertoire mais nous n'en parlerons pas. Pour cela, il nous faut connaître au mieux l'utilité des différentes variables présentes dans ce fichier.

2.1 Les variables globales importantes

2.1.1 Variables du monde

Nous allons décrire les variables qui servent à définir le monde virtuel :

- *SIZE :* Nombre d'avatar ?
- *OUTOFZONENB :* Nombe d'avatar en dehors d'une zone ?
- *TSPEED :* Vitesse limite pour passer en état **T**, vitesse en mètre par seconde.
- *WSPEED :* Vitesse limite pour passer en état **W**, vitesse en mètre par seconde.
- *MAPSIZE :* Taille du monde virtuel, taille en centimètre.
- *ZONESIZE :* Taille d'une zone.
- *SMALLZONESIZE :* Taille d'une zone de petite taille.
- *SMALLZONENB :* Nombre de zone de petite taille.
- *ZONENB :* Nombre de zone.

2.1.2 Variables de la simulation

Les variables de la simulation permettent de faire varier les délais des messages, le nombre de *step*,...

- *MINDELAY :* Délai minimal pour la transmission d'un message
- *MAXDELAY :* Délai maximal pour la transmission d'un message
- *DROPTHRESHOLD :* Seuil de drop ? Sert à quoi ?
- *ANIMATESTEP :* Le pas des animations, 100 correspond à 100ms
- *SHOWSTEP :* Le pas de visualisation

RANDOM : 0 ? Distribution ?
SCEWED : 1 ?

2.1.3 Les différentes stratégies

Il est possible de choisir différentes stratégies, lors de la simulation. On verra ensuite comment affecter l'une des stratégies au simulateur.

- *BASIC :* La stratégie de base sans évolution.
- *ENHANCED :* La stratégie avec anticipation des mouvements.
- *SMALLWORLD :* Autre Stratégie avec des liens longs, ne pas regarder.
- *LA_MIENNE :* Une nouvelle stratégie qu'il faudra implémenté.

EXPAND_COEF : 4 ?
SHARP : 0 ?algo applicatif
EXPAND : 1 ?
STATIC : 0 ? evlatype ?
DYNAMIC : 1 ?
EFFICIENCY : 2 ?

2.1.4 Mobility state machine transition probabilities

Les différents états de l'avatar sont quelques peu différents de ceux énoncés dans Blue Banana, trois états sont définis : l'état **H** pour à l'arrêt, l'état **W** pour en exploration et l'état **T** pour les déplacements rapides. Voici la liste des transitions possibles entre les états et les probabilités correspondantes : (tableau rapport Sergey)

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

TABLE 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*

2.2 Les différentes couches, l'initialisation et les contrôleurs

Un des paramètres importants, est le temps de simulation. La variable suivante permet de définir le temps de fin de la simulation : *simulation.endtime*, le temps est en millisecond. Par exemple, 60000*60*120 correspond à une durée de 120 heures(432 000 000).

2.2.1 Les couches : applicative et protocolaire

La variable *protocol.transport* permet de définir le protocole de transport qui sera utilisé (*peersim.solipsis.GenericTransportLayer* dans notre cas). Ensuite différentes variables permettent de configurer la couche applicative :

- *protocol.applicative :* Le protocole applicative à mettre en place (*peersim.solipsis.SolipsisProtocol*).

- *protocol.applicative.tolerance_level :* Va servir pour le nombre de voisin.
- *protocol.applicative.exp :* Va servir pour le nombre de voisin.
- *protocol.applicative.prefetch_exp :* ? Dans le cas où on utilise l'anticipation ?
- *protocol.applicative.type :* Le type de stratégie utilisée (voir stratégies en 2.1.3).
- *protocol.applicative.algorithm :* Le type d'algorithme utilisé ?SHARP
- *protocol.applicative.expand_coef :* ?

2.2.2 L'initialisation

L'initialisation des différentes composantes du monde se fait grâce aux paramètres suivants :

- *init.populator.distribution :* Modèle de distribution ?
- *init.populator.mapSize :* Taille de la carte.
- *init.populator.zoneSize :* Taille des zones.
- *init.populator.zoneNb :* Nombre de zone.
- *init.populator.outOfZoneNb :* Nombre en dehors des zones.
- *init.populator.smallZoneNb :* Nombre petite zone.
- *init.populator.smallZoneSize :* Taille d'une petite zone.
- *init.populator.wanderingSpeed :* Vitesse pour état **W**
- *init.populator.travellingSpeed :* Vitesse pour état **T**
- *init.populator.quiet :* true utilisé dans statisticsGatherer

Et des paramètres pour fixer les probabilités de transition avec les variables vues en 2.1.4, les paramètres seront de la forme : *init.populator.hthTransitionProbability*

2.2.3 Module de contrôle

- *control.overview :* Classe pour le contrôleur de la vue d'ensemble .
- *control.overview.step :* Nombre d'étape.
- *control.animate :* Classe pour le contrôleur d'animation.
- *control.animate.step :* Nombre d'étape pour l'animation.

3 Le fichier SolipsisProtocol

Nous allons "analyser" le fichier *SolipsisProtocol*, nous commencerons par suivre les différents traitements de la fonction *Receive*, ce qui nous montrera déjà un début du fonctionnement global.

La fonction *Receive* va rediriger, en fonction de leur type, les différents messages. Dans la classe *Message*, nous pouvons voir les différents types de message disponibles. Cette fonction va gérer les types de message suivants :

- *CONNECT :* Provoque l'invocation de la méthode *processConnectMsg*, qui va traiter le message en ajoutant le nœud à la liste des voisins.
- *DELTA :* Provoque l'invocation de la méthode *processDeltaMsg*,
- *DETECT :* Provoque l'invocation de la méthode *processDetectMsg*, qui va regarder tout d'abord si l'entité est dans la liste de détection, et si c'est le cas la supprime.

Si l'entité ne fait déjà pas parti des voisins de l'entité réceptrice, alors on l'ajoute à la liste des voisins. Ensuite traitement pour SmallWorld ? ?

- *SEARCH :* Provoque l'invocation de la méthode *processSearchMsg*, qui va chercher un voisin et va créer un message *FOUND* à celui-ci. Ensuite si la source n'existe pas, elle va récupérer l'identité du nœud "étranger". Elle va envoyer le message créer ultérieurement.
- *FOUND :* Provoque l'invocation de la méthode *processFoundMsg*, si les nœuds ne sont pas déjà voisin alors on ajoute le nœud à sa liste de voisin (gestion cas mode REGULAR et PREFETCH). Si les nœuds sont voisins, on va analyser l'enveloppe convexe.
- *CLOSE :* Provoque l'invocation de la méthode *processCloseMsg*, va enlever le nœud de la liste des voisins et checker l'enveloppe convexe.
- *PREFETCH :* Provoque l'invocation de la méthode *processPrefetchMsg*, on teste si il est intéressant de prefetcher le nœud (distance trop courte ?), traitement et envoie vers nœuds.
- *FIND_NEAREST :* Provoque l'invocation de la méthode *processFindNearest*, recherche le plus près.

Références

[1] Sergey Legtchenko, Sébastien Monnet, and Gaël Thomas. Blue Banana : resilience to avatar mobility in distributed MMOGs. Research Report RR-7149, INRIA, 2009.