Titre: Title:	Architecture de rebalancement dynamique pour jeux massivement multijoueurs en ligne fonctionnant sur réseaux pair-à-pair
Auteur: Author:	Julien Gascon-Samson
Date:	2010
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Gascon-Samson, J. (2010). Architecture de rebalancement dynamique pour jeux massivement multijoueurs en ligne fonctionnant sur réseaux pair-à-pair [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/489/

UNIVERSITÉ DE MONTRÉAL

ARCHITECTURE DE REBALANCEMENT DYNAMIQUE POUR JEUX MASSIVEMENT MULTIJOUEURS EN LIGNE FONCTIONNANT SUR RÉSEAUX PAIR-À-PAIR

JULIEN GASCON-SAMSON
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION DU DIPLÔME DE
MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2010

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

ARCHITECTURE DE REBALANCEMENT DYNAMIQUE POUR JEUX MASSIVEMENT MULTIJOUEURS EN LIGNE FONCTIONNANT SUR RÉSEAUX PAIR-À-PAIR

présenté par : GASCON-SAMSON Julien.
en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées
a été dûment accepté par le jury constitué de :

M. PIERRE Samuel, Ph.D., président.
M. QUINTERO Alejandro, Doct., membre et directeur de recherche.
Mme BOUCHENEB Hanifa, Doctorat, membre.

À mes parents, ma famille, mes animaux de compagnie, ainsi qu'à toutes les personnes de mon entourage qui m'ont encouragées de près ou de loin à la réalisation de ce projet.

À mes amis, membres de la "Nation Deportivienne", qui ont su m'aider à me distraire un peu après une dure semaine de labeur avec leur sens de l'humour original et légèrement inhabituel.

Remerciements

Je tiens à remercier chaleureusement mon directeur de recherches, M. Alejandro Quintero, qui a su me fournir tout l'encadrement nécessaire, répondre à mes interrogation et me lancer sur de bonnes pistes lors de mon travail.

Je tiens également à remercier spécialement Mme LiJun ZHANG, M. Samuel Pierre, M. Yves Marthone, M. André Sambour, M. Zied Zaier et toute l'équipe de l'entreprise Geninov inc. qui m'ont fourni l'encadrement et les ressources (matérielles et financières) nécessaires à la réalisation de ce projet. Leur appui a été très apprécié et leur collaboration a été inestimable.

J'adresse également un merci spécial à toute l'équipe pédagogique, au cadre professoral et aux collègues du LARIM et de l'École Polytechnique de Montréal qui on su me fournir un environnement de travail adéquat pour la réalisation de ce travail ainsi que tous les outils et conseils nécessaires. J'adresse des remerciments particuliers à M. Olivier Gendreau pour ses conseils et son expérience lors de la rédaction du mémoire de maîtrise.

Je ne saurais mentionner l'appui financier important accordé par les organismes subventionnaires FQRNT et CRSNG, en partenariat avec l'entreprise Geninov inc., dans le cadre de la bourse "BMP Innovation", pour leur appui financier qui a permis de mener le travail à terme.

Résumé

L'industrie des jeux vidéos a connu une forte explosion ces dernières années. En particulier, un tout nouveau genre de jeux, les jeux massivement multijoueurs en ligne, a connu une forte popularisation. Ces jeux se caractérisent par un environnement virtuel immense et persistant qui est continuellement actif et évolutif. À la différence des autres jeux, les jeux massivement multijoueurs en ligne intègrent des milliers de joueurs qui participent au sein du même univers commun à tous.

À l'heure actuelle, la gestion informatique de tels univers nécessite beaucoup de ressources et constitue un défi de taille afin de s'adapter au nombre toujours croissant de joueurs. Le modèle client-serveur est actuellement utilisé, mais ce dernier s'avère éventuellement limité puisqu'il arrive un point où la puissance d'une seule machine ne suffit plus à assurer la prise en charge de tous les joueurs. L'utilisation d'un modèle pair-à-pair constitue une approche intéressante puisqu'il permet de redistribuer la charge de traitement requise pour assurer la maintenance du jeu aux noeuds-joueurs eux-mêmes.

Certaines approches pair-à-pair ont été proposées dans la littérature. Nous proposons d'aller un peu plus loin en proposant une approche hybride flexible capable de s'adapter automatiquement aux conditions actuelles en cours de jeu. Plus précisément, notre approche propose de découper dynamiquement le territoire du jeu et d'assigner chaque parcelle à un noeud serveur choisi arbitrairement parmi les joueurs participant actuellement au jeu. Le modèle propose ensuite d'analyser continuellement la charge réseau imposée à chaque noeud et d'appliquer automatiquement des opérations de rebalancement pour redistribuer la charge afin d'assurer une qualité de jeu optimale pour tous.

À cette fin, une plate-forme de simulation implémentant le modèle proposé a été construite afin d'évaluer le fonctionnement du modèle sous différentes conditions. Plusieurs simulations complètes de longue durée ont été réalisées. À partir des données produites par ces simulations, nous avons étudié la charge consommée par le serveur central par rapport au modèle client-serveur classique. Nous avons également étudié la charge imposée à chaque noeud serveur pour s'assurer que la capacité maximale n'était pas dépassée et que la latence demeurait dans des valeurs raisonnables.

Les simulations les plus restrictives au niveau des paramètres ont montré que dans certaines situations limites (un très grand nombre de joueurs sont rassemblées dans une région restreinte, ou alors la capacité des noeuds est trop faible), le modèle peut avoir de la difficulté à maintenir une bonne répartition de charge. À l'inverse, nous avons déterminé que sous des conditions typiques (capacité des noeuds, distribution des joueurs, taille du monde virtuel),

le modèle est pleinement fonctionnel et permet d'assurer une qualité de jeu très satisfaisante pour pratiquement l'ensemble des joueurs, et ce, malgré les fluctuations et changements à l'univers virtuel survenant continuellement en cours de jeu. Nous pouvons donc conclure en disant que le modèle proposé constitue une alternative fonctionnelle et efficace au modèle client-serveur actuellement mis en place.

Abstract

In the recent years, the video game industry has been given much attention. More specifically, a new game genre, massive multiplayer online games (MMOG), has emerged. MMOG games feature a very large and persistent virtual universe that always remains active and evolves continuously. Contrary to other traditional multiplayer games, massively multiplayer online games typically have thousands of simultaneous connected players.

As for the moment, managing such complex game universes takes a lot of system resources. Furthermore, adapting to the fluctuing number of game players is an important challenge. As of today, massively multiplayer online games use the so-called client-server model, but this model is limited by the fact that a single machine cannot handle more than a given number of players. A peer-to-peer model is a more interesting approach since it allows redistributing game maintenance load to all player nodes.

Some peer-to-peer approaches propose have been proposed in the litterature. We propose going farther by proposing a flexible hybrid architecture that can adapt itself to current game conditions. More precisely, our approach propose dynamically splitting the game territory into a given number of pieces and assigning each "piece" to an arbitrarily-chosen server node from the currently participating players. The proposed model then propose continuously analyzing the network load for each game zone and automatically applying rebalancing operations to redistribute the load, effectively leading to a better game quality for all players, in all conditions.

A simulation patform implementing the proposed model has been developped. This platform, called the "simulator", has been built to evaluate our model operates under different conditions. Many complete simulations have been performed. Those simulations have been run for a long time lapse. Data samples have then been produced from the performed simulations. We proceeded to study the load consumed by the central server using the proposed model and compared it against the client-server model. We also studied the load imposed to each server node to ensure the maximal allowable load wasn't exceeded and made sure that the latency was within an acceptable range.

The simulations with the more restrictive parameters demonstrated that under some critical situations (such as a very large number of players located in a small area or such as a very low node load threshold), the proposed model may have trouble maintaining a good load balancing. Inversely, we have determined that under typical conditions (node load threshold, player distribution, virtual world size), the model is fully functional and ensures an excellent game quality for pratically all players, despite the constantly-evolving nature

of MMOG games. Consequently, we can conclude by saying that the proposed model is a functional and efficient alternative to the traditional client-server typically used in today's applications.

Table des matières

Liste des tableaux

Liste des figures

Liste des sigles et abréviations

RPG	Role Playing Game
FPS	First Person Shooter
RTS	Real Time Strategy
MMOG	Massively Multiplayer Online Game
MMORPG	Massively Multiplayer Online Role Playing Game
MMOFPS	Massively Multiplayer Online First Person Shooter
MMORTS	Massively Multiplayer Online Real Time Strategy
CS	Client-Serveur
P2P	Peer-to-peer (pair-à-pair)
NPC	Non Playing Character (personnage non-joueur)
AOI	Area of Interest (zone d'intérêt)
DHT	Distributed Hash Table (table de hachage distribuée)
LR	Load Ratio (ratio de charge)
ZL	Zone Latency (latence de zone)
RW	Random Waypoint
HRW	Hotspots Random Waypoint

Chapitre 1

INTRODUCTION

L'univers des jeux informatiques est aujourd'hui un monde vaste. À l'heure actuelle, il existe une multitude de jeux variés de tous genres. Initialement, les jeux étaient joués presque exclusivement de façon individuelle où chacun des joueurs opérait sa propre copie du jeu et jouait de façon indépendante. L'univers du jeu, c'est-à-dire les différentes entités le composant tels que le monde, les personnages et les différents objets, existaient de façon totalement indépendante et étaient propres à chaque instance du jeu s'exécutant sur chacune des plate-formes.

Au cours des dernières années, le mode solo, bien que toujours existant, a peu à peu diminué pour faire place à un nouveau concept : les jeux multijoueurs. Ce nouveau concept permettait à quelques joueurs organisés en réseau local de participer au même jeu, mais à la différence que les différents joueurs participaient à la même partie. L'univers du jeu n'était donc plus propre à chaque joueur, mais bien partagé entre les différents joueurs qui participaient en réseau. Typiquement, le mode multijoueurs fonctionnait en déterminant un hôte pour la partie (habituellement celui qui amorçait ou déterminait le jeu) bien que dans certains cas, d'autres techniques aient été utilisées telles que le "broadcast" ou le mode pair-à-pair. Le concept a par la suite été étendu et popularisé et des serveurs dédiés à l'orchestration de parties ont été déployés. Ces serveurs permettaient à différents joueurs d'organiser entre eux, via Internet, des parties de différents jeux. Ces parties étaient alors gérées par le serveur central. Ce dernier avait donc pour but de gérer une multitude de parties indépendantes du jeu pour des groupes de joueurs. L'opérateur du jeu pouvait donc offrir une "plus value" à son jeu en offrant un service de rencontres virtuelles permettant ainsi aux différents joueurs de se retrouver et de démarrer une partie rapidement. Un autre avantage de ce principe était que l'opérateur du jeu était assuré que les différents utilisateurs jouaient selon les règles et principes propres au jeu et possédaient réellement une copie originale du jeu. Cela a donc permis de réduire les possibilités de tricher et donc de favoriser un jeu équitable pour tous. Cependant, certains joueurs ont tout de même découvert des façons de tricher en utilisant divers outils permettant d'avoir accès à des informations normalement non-autorisées dans certains contextes donnés du jeu ou en manipulant les données transmises par le réseau (proxy)(Webb et Soh, 2007).

Il y a quelques années, une nouvelle catégorie de jeux multijoueurs ont fait leur apparition : les jeux massivement multijoueurs en ligne (MMOG). Ces jeux se distinguent des jeux multijoueurs conventionnels puisqu'ils intègrent une très grande quantité de joueurs participant au sein d'une même partie. Contrairement aux jeux classiques qui permettent normalemement au maximum une dizaine de joueurs ou tout au plus une vingtaine, les jeux massivement multijoueurs peuvent normalement accueillir au-délà d'une centaine de joueurs et même souvent plusieurs miliers. Bien que certains jeux classiques possèdent un territoire (virtuel) assez large, une caractéristique importante des jeux massivement multijoueurs en ligne est qu'ils sont composés d'un monde virtuel extrêmmement vaste.

L'univers des jeux informatiques est suffisamment large à l'heure actuelle. Il existe plusieurs catégories de jeux informatiques. Mentionnons notamment les jeux de type *Role Playing Game* (RPG) où les utilisateurs y participant incarnent des rôles de personnages fictifs et doivent effectuer différentes actions telles qu'accomplir des quêtes, récupérer des artéfacts, interagir avec d'autres personnages (réels ou fictifs) au sein d'un environnement virtuel. Mentionnons également les jeux de type *First Person Shooter* (FPS) où les utilisateurs incarnent un personnage muni d'une ou plusieurs armes et doivent tirer des projectiles sur des joueurs opposants (réels ou fictifs). Ces jeux ont la caractéristique que l'environnement visuel et auditif est centré sur le personnage représenté par le joueur ; on ne voit habituellement que partiellement et bien souvent pas du tout la représentation visuelle du joueur à l'écran (avatar) puisqu'on considère que le personnage est le joueur réel. Mentionnons enfin les jeux de stratégie (*Real Time Strategy* ou RTS) dans lesquels les utilisateurs doivent mettre au point un plan, construire des unités, bâtiments et acquérir diverses ressources afin d'établir leur supériorité par rapport aux autres joueurs et éventuellement remporter la victoire. Plusieurs autres genres de jeux existent, notamment les jeux de simulations (conduite automobile, hockey, boxe, baseball, etc.). Certains jeux combinent également plus d'un genre.

Il est possible d'imaginer différents genres de jeux massivement multijoueurs ; bien que pour l'instant le genre dominant soit les jeux de rôle (RPG). En effet, ce genre se prête particulièrement bien aux jeux massivement multijoueurs puisque déjà, à la base, les jeux appartenant à cette catégorie intègrent un monde virtuel assez vaste. Dans cette optique, presque tous les jeux massivement multijoueurs en ligne sur le marché à l'heure actuelle sont de type RPG (*Massively Multiplayer Online Role Playing Game* ou MMORPG). Les éditeurs de tels jeux ont créé des mondes virtuels très vastes pouvant accueillir des centaines, voire même des miliers de joueurs simultanément.

1.1 Définitions et concepts de base

Les jeux massivement multijoueurs en ligne possèdent certaines caractéristiques et notions propres qu'il convient d'expliciter puisque plusieurs approches font appel à bon nombre de ces notions.

1.1.1 Zone d'intérêt

La zone d'intérêt (AOI) réfère à une région physique entourant chacun des joueurs. Cette zone, représentée par un disque d'un rayon donné autour de chaque joueur, modélise la capacité de perception du joueur et sert à discriminer les objets et événements qui sont dans la portée de perception du joueur. Ainsi, chaque joueur participant au jeu souhaitera être informé de l'ensemble des objets, actions, événements et changements d'état qui se produiront à l'intérieur de la région délimitée par la zone d'intérêt. À l'inverse, tout événement se produisant hors de la zone d'intérêt d'un joueur donné ne sera pas perçu par ce joueur. La figure 1.1 représente trois joueurs P_1, P_2 et P_3 et leur zone d'intérêt. Tel qu'illustré, dans cet exemple, les joueurs P_1 et P_2 seront mutuellement informés de la présence, de la proximité et des actions de l'un et l'autre. De plus, ni P_1 ni P_2 ne seront informés de l'emplacement de P_3 et P_3 ne sera informé de l'emplacement de P_1 et P_2.

Greenhalgh et Benford (1995) proposent un modèle plus avancé de gestion de la zone d'intérêt en définissant deux concepts : le nimbus et le focus. D'après les auteurs, tout sujet possède une capacité à être observé sous forme d'un aura définissant une certaine région donnée de l'espace (nimbus) et tout observateur possède une capacité à observer l'environnement l'entourant (focus). La zone dans laquelle l'observateur peut percevoir le sujet, appelée zone de perception, est donc fonction du nimbus et du focus. Les auteurs mentionnent également la possibilité de modéliser le nimbus et le focus par des fonctions arbitraires, permettant ainsi de représenter de façon plus précise la portée des différentes entités.

1.1.2 Événements

Un événement fait référence à la conséquence d'une action, directe ou indirecte, se produisant à un endroit donné au sein du monde virtuel. Un événement peut être généré volontairemet suite à une action posée par un joueur (un déplacement, une attaque, etc.) ou être déclenché de façon involontaire (par exemple, un scénario préprogrammé dans le jeu). Un événement affecte en principe tous les joueurs pour lesquels ce dernier se produit dans leur zone d'intérêt.

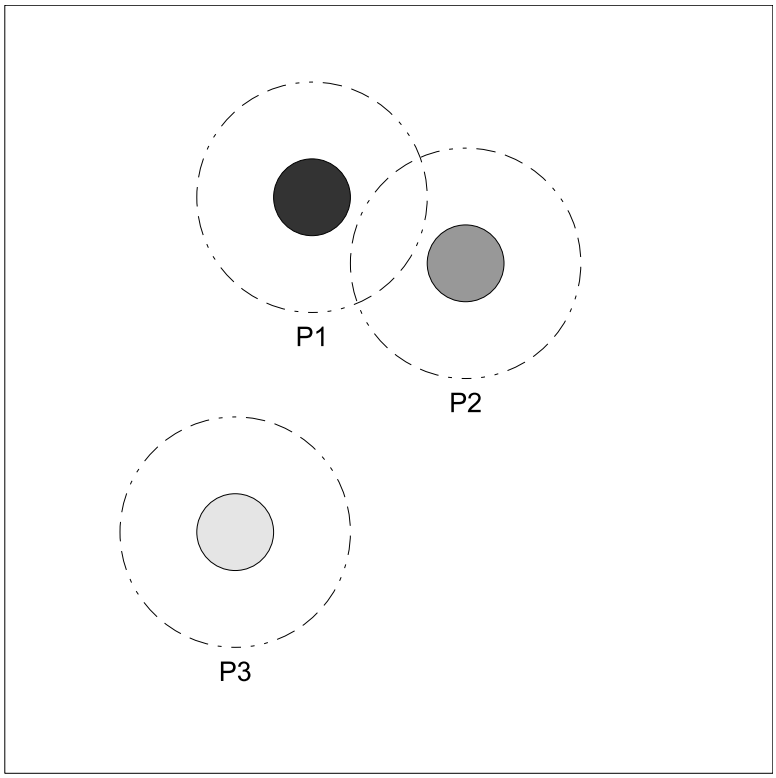


FIGURE 1.1 Zone d'intérêt de trois joueurs participant à un jeu multijoueurs

1.1.3 Découpage physique

Afin de répartir la charge de traitement requise entre les différents participants, certaines approches optent pour un découpage et une distribution physique du territoire tandis que d'autres proposent plutôt un découpage logique.

Un découpage physique consiste à répartir de façon géographique l'ensemble du territoire du jeu entre les différents serveurs [1]. Ainsi, le territoire global du jeu est subdivisé en une multitude de régions de forme régulière ou irrégulière qui sont gérées par un ou plusieurs serveurs différents.

1.1.4 Découpage logique

Un découpage logique consiste à répartir l'ensemble des entités du monde virtuel entre l'ensemble des serveurs sans égard à la disposition géographique du territoire. Par exemple, il est possible d'assigner un identificateur à chaque entité (objet, personnage non-joueur,

1. Dans le contexte d'une architecture intégrant une composante pair-à-pair, le terme *serveur* peut référer à la fois à un serveur dédié et à un serveur temporaire s'exécutant sur la machine hôte d'un joueur participant au jeu.

joueur, etc.) et de répartir l'ensemble des entités entre les différents serveurs.

1.2 Éléments de la problématique

De par leur nature, les jeux massivement multijoueurs en ligne nécéssitent d'énormes quantités de ressources (temps CPU, bande passante, mémoire). Pour être en mesure d'accueillir des centaines, voire des milliers de joueurs, il est nécessaire de mettre en place des infrastructures réseau énormes. À l'heure actuelle, les jeux massivement multijoueurs en ligne sont gérés par un seul serveur central [2]. Cette approche est plus facile à mettre en œuvre et permet de garantir une sécurité accrue auprès de l'opérateur du jeu en ligne.

Cependant, cela amène malheureusement certaines limitations quant à la quantité de joueurs pouvant être pris en charge simultanément au sein d'un même univers virtuel. Un serveur typique utilisé pour gérer de tels jeux peut prendre en charge un maximum de quelques miliers de joueurs simultanément. Au-délà d'un certain point, la puissance d'un seul serveur central (ou d'un groupe de serveurs) peut ne plus suffir à assurer la prise en charge de tous les joueurs participant. C'est le cas notamment du jeu *World of Warcraft* de Blizzard Entertainment, qui possède une multitude de serveurs indépendants (au-délà de 250), opérant en parallèle des copies indépendantes du jeu pour être en mesure de répondre aux besoins des joueurs [3].

1.3 Objectifs de recherche

L'objectif de ce projet de recherche est de proposer une nouvelle architecture de gestion de jeux massivement multijoueurs en ligne. Cette architecture fonctionnera selon un paradigme pair-à-pair (P2P). L'objectif est de répartir la majeure partie de la charge de traitement entre les noeuds joueurs. Ainsi, un joueur participant au jeu pourrait être appelé à consacrer une partie de ses ressources système pour gérer le jeu pour d'autres joueurs.

Le principe de découpage géographique, qui consiste à répartir le territoire géographique de l'univers virtuel en morceaux distincts, sera utilisé. Ce travail de recherche répondra à deux éléments :

– découpage du territoire : proposer un schéma de découpage du territoire global permettant d'assigner une parcelle ("zone") à différents noeuds

2. Il peut s'agir d'un groupe de serveurs (grappe ou *cluster*), mais d'un point de vue logique, nous pouvons le considérer comme une seule machine.

3. La technique de la fragmentation ("sharding") est utilisée ; cette dernière sera décrite dans la revue de littérature.

– rebalancement en temps réel : proposer un algorithme de rebalancement permettant de rebalancer dynamiquement la charge, de façon à s'assurer que les noeuds-serveurs ne soient pas surchargés tout en répondant aux fluctuations survenant en cours de partie

Initialement, les approches existantes de la littérature seront étudiées afin d'identifier dans quelle mesure elles répondent à la problématique actuelle. Les points forts et les points faibles de chaque approche seront étudiés, pour déterminer à quels besoins elles répondent et identifier quelles améliorations pourraient y être apportées.

Par la suite, un nouveau modèle sera proposé. Pour valider ce nouveau modèle, différents paramètres de simulation seront définis (jeux de données en entrée). Il serait illogique d'assigner des valeurs fixes aux différents paramètres puisque ceux-ci dépendent fortement du type de jeu, et même de chaque jeu développé, qui nécessitent des quantités différentes de ressources. L'objectif est plutôt de déterminer si le modèle proposé arrive à s'adapter à différents scénarios pouvant s'appliquer à différents jeux. Pour cette raison, la démarche proposée préconisera plutôt de varier les différents paramètres pour avoir des jeux de données représentant des contextes de jeux plus légers et plus gourmands en ressources.

À l'heure actuelle, suite aux recherches effectuées, aucun outil de simulation existant ne s'est revelé adéquat pour ce projet. Une des tâches consistera donc à réaliser une plate-forme de simulation implémentant le modèle proposé pour effectuer des simulation de jeux en ligne.

Enfin, les simulations seront effectuées avec les différents paramètres identifiés. Les résultats seront analysés et comparés pour déterminer si l'approche proposée est efficace, et dans quels contextes il pourrait être souhaitable de l'appliquer.

1.4 Plan du mémoire

Le chapitre suivant effectuera une revue de littérature des différents modèles proposés dans la littérature pour optimiser la gestion de jeux massivement multijoueurs. Certaines notions et concepts dignes d'intérêt qui sont couramment utilisés seront décrits. Plusieurs stratégies sont possibles pour optimiser de tels jeux (gestion statique/dynamique, découpage physique/logique, etc.). Les modèles proposés seront classifiés selon la stratégie utilisée.

Le troisième chapitre portera sur l'architecture proposée. Les différents concepts utilisés seront décrits. Les opérations de rebalancement ("load balancing") pour rééquilibrer la charge seront décrites ainsi que les différents algorithmes conçus. La fonction de coût servant à déterminer si un rebalancement est nécessaire sera également détaillée.

Le quatrième chapitre décrira la plate-forme de simulation développée, effectuera un survol des différentes fonctionnalités offertes. Les jeux de données utilisés et le résultat des simulations réalisées avec ces jeux de données seront décrits afin d'analyser l'efficacité de la

méthode proposée.

En conclusion, le dernier chapitre résumera l'approche proposée, en quoi elle diffère des autres approches. Il sera également mentionné pour quelles raisons elle s'avère efficace (si les résultats le démontrent) et dans quels contextes elle pourrait être employée. Des pistes d'améliorations possibles seront présentées.

Chapitre 2

REVUE DE LITTÉRATURE

2.1 Notions mathématiques et informatiques

Puisque les jeux massivement multijoueurs incorporent habituellement de grands territoires, on a souvent recours au modèle mathématique et à la géométrie du plan pour modéliser différentes approches. Dans ce contexte, certaines notions mathématiques peuvent s'avérer utiles en ce qui a trait à certains aspects de l'organisation géographique.

2.1.1 Triangulation de Delaunay

La triangulation de Delaunay [1] (Chew (1987)) consiste à relier un ensemble de points P dans le plan de manière à former uniquement des triangles, le tout en respectant certaines conditions. La triangulation de Delaunay subdivisera donc la région délimitée par P en triangles adjacents et interconnectés : une *triangulation DT*. Toutefois, pour chacun des triangles, l'agencement doit être construit de telle sorte qu'aucun point de P ne se retrouve à l'intérieur du cercle passant par les trois points de ce triangle.

2.1.2 Diagramme de Voronoi

Un diagramme de Voronoi [2] (Aurenhammer (1991)) est une subdivision d'un plan contenant un ensemble de points P en plusieurs régions distinctes en fonction de la position et des distances entre les différents points. Chacune des régions, qui regroupe un seul point P_i de l'ensemble P, est formée de l'ensemble des points dans le plan dont la distance par rapport à P_i est inférieure à la distance par rapport à tous les autres points de l'ensemble P. Par conséquent, les arêtes sont constituées de l'ensemble des points dans le plan qui sont équidistants à deux points de l'ensemble P et les sommets sont l'ensemble des points dans le plan qui sont équidistants à un minimum de trois points de l'ensemble P.

Les notions de triangulation de Delaunay et de diagramme de Voronoi sont fortement liées : pour un même ensemble de points P, chacun des sommets des régions délimitées par le

1. Il est question ici de triangulation bidimensionnelle ; il est toutefois également possible d'adapter le processus à un espace n-dimensionnel

2. Dans le plan

diagramme de Voronoi correspond au centre de l'un des cercles regroupant trois des points de la triangulation de Delaunay. Partant de ce lien, il est possible de passer d'une représentation à l'autre facilement.

Dans la pratique, les diagrammes de Voronoi peuvent s'avérer fort utiles pour subdiviser géographiquement un territoire en plusieurs sous-régions lorsque l'on cherche à optimiser (minimiser) les distances par rapport à certains points de repère.

2.1.3 Enveloppe convexe

L'enveloppe convexe d'un ensemble de points est le polygone convexe de périmètre minimal qui englobe tous les points de l'ensemble. Par conséquent, chacun des sommets du polygone est formé de l'un des points. Ces points forment la bordure du polygone.

Les points qui ne peuvent faire partie de l'ensemble des sommets du polygone sans briser l'exigence de convexité sont contenus à l'intérieur du polygone, mais ne servent pas à en définir la bordure.

2.1.4 Tables de hachage distribuées

Une table de hachage *simple*(Litwin, 1980; Tanenbaum *et al.*, 1990) est une structure de données qui permet de storer des paires d'éléments selon le principe "clé-valeur" où la clé est le résultat de l'application d'une fonction de hachage sur la valeur. Les valeurs sont stockées et récupérées en utilisant leur clé comme mécanisme d'indexation.

Le principe de la table de hachage distribuée (DHT)(Balakrishnan *et al.*, 2003; Ratnasamy *et al.*, 2001) est que le stockage n'est pas confiné à une seule entité, mais bien à plusieurs entités différentes reliées en réseau. L'ensemble des entrées de la table sont réparties entre plusieurs noeuds distincts. Les DHT ne nécessitent pas de serveur central ; elles fonctionnent en utilisant une couche réseau pair-à-pair uniquement.

Le principe de fonctionnement repose sur le fait qu'il peut être possible de calculer la *distance* mathématique entre deux clés. Chacun des noeuds participant à la table de hachage possède alors un identificateur donné, et incluera l'ensemble des paires clé-valeur dont la distance entre la clé et l'identificateur du noeud est la moins élevée par rapport aux autres noeuds.

Une conséquence de l'architecture pair-à-pair est que le stockage et la récupération fonctionnent par propagation. Ainsi, afin de découvrir quel noeud devra contenir une paire clé-valeur donnée, la requête devra être relayée d'un noeud à l'autre jusqu'au destinataire final. Certaines implémentations opèrent uniquement de manière circulaire où chaque noeud connaît seulement ses deux voisins immédiats, et leur identificateur. D'autres implémentations

autorisent les noeuds à maintenir une liste de noeuds (antémémoire) et leurs identificateurs afin d'acheminer la requête au destinataire plus rapidement.

Pastry

Pastry (Rowstron et Druschel (2001); Castro *et al.* (2002)) est une implémentation d'une table de hachage distribuée possédant une infrastructure de routage. Chaque utilisateur (ou noeud) membre d'un réseau Pastry est muni d'un identificateur choisi arbitrairement parmi un espace de clés circulaire à 128 bits. Un noeud qui souhaite joindre le réseau Pastry n'a besoin de connaître que l'adresse réseau (par exemple, l'adresse IP) d'un autre noeud.

Chaque noeud contient et maintient à jour trois listes : une liste des noeuds terminaux, une liste de voisins (non utilisé par l'algorithme de routage ainsi qu'une table de routage. Un noeud peut générer et transmettre un message ou relayer (et éventuellement modifier) un message reçu d'un autre noeud vers son destinataire final. À l'instar des autres implémentations de tables de hachage distribuées, Pastry exploite le principe de *distance numérique* en tant que mécanisme de routage. Ainsi, pour qu'un message donné atteigne le noeud final voulu, chaque noeud-relais transmettra toujours ce dernier vers le noeud connu se rapprochant le plus numériquement du destinataire.

La liste des noeuds terminaux de tout noeud est un tableau associatif reliant les n (variable paramétrable) identificateurs de noeuds les plus près numériquement de l'identificateur du noeud courant à leur adresse réseau. Cette liste permet au noeud de router les messages directement vers le destinataire si ce dernier est suffisament rapproché numériquement du noeud courant. Cependant, si le destinataire n'est pas dans la liste des noeuds terminaux, la table de routage sera utilisée afin de repérer le prochain noeud intermédiaire le plus près du destinataire final.

Scribe

Scribe (cite) est une autre implémentation d'une table de hachage distribuée. Cette implémentation permet de mettre en oeuvre un modèle "publication-souscription" (*publish-subscribe*) de façon distribuée en optimisant le routage des messages entre les noeuds.

Un noeud peut effectuer trois actions : amorcer la création d'un flux, publier des messages dans un flux précédemment créé par ce dernier et s'abonner à un flux créé par un autre noeud (et éventuellement y recevoir les message transmis). Les flux sont organisés selon des arbres multicast : un noeud (appelé racine) est responsable du flux et une hiérarchie de noeuds fils servent à transmettre les messages reçus du noeud racine vers les destinataires intéressés. Notons qu'un noeud fils peut avoir ou non des enfants.

Le principe de *distance numérique* est utilisé par Scribe afin d'associer un flux particulier à un noeud donné. Lors de la création d'un nouveau flux, un identificateur est créé en calculant le résultat du hachage de la concaténation du nom du flux et de l'utilisateur responsable de sa création. La responsabilité du flux est alors attribuée au noeud dont la distance entre l'identificateur du noeud et l'indentificateur du flux est minimale.

Lorsqu'un noeud donné souhaite joindre un flux existant, il calcule l'identificateur du flux selon la fonction mentionnée précédemment et achemine un message à destination de cet identificateur. Le message suivra donc un trajet en étant routé à travers plusieurs noeuds intermédiaires, et l'identificateur de chaque noeud intermédiaire parcouru se rapprochera toujours davantage de l'identificateur du flux. Toutefois, si un noeud sur le chemin est lui-même inscrit au flux demandé, ce dernier arrêtera la propagation du message d'inscription et ajoutera le noeud à sa liste d'enfants.

Le processus de publication d'un message suit le même principe : le créateur du flux transmet d'abord le message au noeud racine. Le message est par la suite propagé récursivement d'une hiérarchie d'enfants à la suivante jusqu'à la base de l'arbre.

2.2 Répartition statique

Les jeux massivement multijoueurs en ligne regroupent bien souvent une grande quantité de joueurs jouant simultanément au sein du même monde virtuel. Pour un jeu dont le nombre de joueurs n'est pas trop élevé, il peut arriver qu'un seul serveur arrive à gérer l'ensemble du déroulement du jeu, sachant qu'un serveur performant peut gérer au plus quelques miliers de joueurs participant simultanément au jeu. Cependant, dans la pratique, il est souvent nécessaire de mettre en place un réseau de serveurs dédiés.

La présente section présente des approches permettant de répartir de façon statique la charge réseau requise en utilisant plusieurs machines. Une répartition statique consiste à définir initialement de quelle façon le jeu sera séparé entre les différentes serveurs. Contrairement à une répartition dynamique, une répartition statique ne modifiera pas la répartition en cours de jeu. Cette approche peut s'appliquer en utilisant le modèle client-serveur (CS) classique ou en utilisant le modèle pair-à-pair.

2.2.1 Fragmentation ("sharding")

Une technique couramment utilisée est la "fragmentation" (*sharding*), qui consiste à créer plusieurs copies indépendantes du même jeu (et donc par conséquent du même environnement virtuel). Cette technique ne nécessite pas d'architecture réseau spécifique pour être mise

en œuvre puisqu'elle consiste simplement à exécuter plusieurs fois de manière indépendante le même logiciel. Chacune des copies est gérée par un serveur dédié, s'exécute de façon autonome et complètement indépendante des autres et possède son propre lot d'utilisateurs. Cela signifie que lorsqu'un nouvel utilisateur souhaite s'inscrire afin de participer au jeu, il doit déterminer dans quelle instance propre il souhaite se joindre. Il ne pourra communiquer qu'avec les autres joueurs ayant également rejoint cette instance et ne pourra interagir qu'avec les éléments de l'environnement virtuel spécifiques à cette instance. Bien sûr, certains mécanismes peuvent être mis en place pour permettre le transfert de comptes utilisateurs entre instances ; ces mécanismes sont toutefois utilisés uniquement sur une base ponctuelle et occasionnelle. L'utilisation de cette technique a pour conséquence que le monde virtuel, bien qu'initialement identique, évoluera de façon différente sur chacune des instances. Bien que cette technique permette de répartir la capacité de traitement totale requise sur plusieurs serveurs distincts, elle possède l'inconvénient que les utilisateurs d'une instance donnée ne peuvent entrer en communication avec les entités d'une autre instance. Un exemple de jeu qui utilise abondamment la technique de la fragmentation est le jeu *World Of Warcraft* qui est présentement le jeu massivement multijoueurs en ligne le plus populaire. Le nombre total de joueurs dépasse 11 millions ; pour être en mesure d'offrir une expérience de jeu de qualité à tous les joueurs, le jeu est présentement fragmenté en 238 instances indépendantes.

2.2.2 Grille informatique

Une autre technique souvent utilisée est la technique de la grille informatique. Cette technique consiste à répartir la charge de traitement totale entre plusieurs serveurs dédiés. Chaque serveur devient donc responsable d'une partie (géographique ou logique) du monde virtuel total.

Second Life

Le jeu *Second Life*[3] tire parti de l'architecture en grille en répartissant le traitement requis entre différents serveurs dédiés. Chaque serveur (appelé simulateur) prend en charge une partie du territoire physique du jeu. Chaque simulateur gère une zone de superficie de 256 mètres carrés (par rapport au jeu) et est relié aux quatre autres simulateurs voisins selon les repères cardinaux. Chaque joueur participant au jeu (avatar) se retrouve donc connecté à l'un des simulateurs selon la zone dans laquelle il se trouve ; lorsque ce dernier change de zone, il est automatiquement transféré au simulateur de la zone adjacente.

3. `http://www.gamasutra.com/resource_guide/20030916/rosedale_pfv.htm`

Architecture distribuée pour jeux MMORPG

Assiotis et Tzanov (2006) proposent une méthode permettant de répartir la charge de traitement totale d'un jeu massivement multijoueurs en ligne entre plusieurs serveurs et gérer le déroulement du jeu. Leur méthode propose de diviser géographiquement l'espace physique total du jeu W en plusieurs zones $w_1, w_2, \ldots, w_n \subset W$. Chacune des zones w_i est assignée à un serveur dédié (géré par l'opérateur du jeu). L'ensemble des entités du jeu, appelées objets (pouvant inclure les avatars des joueurs eux-mêmes, les personnages non-joueurs (NPC) ainsi que les différents éléments avec lesquels il est possible d'interagir), sont situées dans l'une ou l'autre des zones. Chaque joueur possède une zone d'intrêt donnée s'exprimant sous la forme d'un disque de rayon donné autour de son avatar.

Chacune des zones w_i du jeu regroupera éventuellement un certain nombre de joueurs. Chaque joueur interagira avec son environnement immédiat, sous le contrôle du serveur responsable de la région dans lequel le joueur est situé. Ainsi, chacun des joueurs est "abonné" aux événements publiés par son serveur responsable. Cependant, advenant le cas où un joueur serait localisé à la frontière de sa zone w_i, il se pourrait que sa zone d'intérêt intègre une partie de la zone adjacente w_j gérée par un ou plusieurs autres serveurs. Dans un tel cas, ce joueur recevrait également des événements en provenance des autres serveurs (pour la région recoupante uniquement), en utilisant son serveur courant comme relais. Le serveur courant responsable de ce joueur, étant déjà interconnecté aux quatres serveurs adjacents, agirait donc comme passerelle et retransmettrait les événements reçus de la part des serveurs adjacents concernant le joueur.

La figure 2.1 illustre un scénario avec deux joueurs P_1 et P_2 et leur zone d'intérêt répartis au sein d'un territoire découpé en quatre zones rectangulaires gérées par quatre serveurs dédiés S_1, S_2, S_3 et S_4. Dans cette situation, le joueur P_1 est géré exclusivement par le serveur S_1 puisque ce dernier et sa zone d'intérêt sont contenus entièrement dans le territoire desservi par S_1. À l'inverse, le joueur P_2 est géré par le serveur S_2 puisqu'il est situé dans le territoire desservi par ce dernier, mais reçoit également des événements en provenance du serveur S_4 puisque la zone d'intérêt de P_2 recoupe une partie du territoire de S_4.

Le mécanisme de relais en conditions frontières et d'interconnexions entre les serveurs pose certains défis au niveau de la consistence du jeu puisque plus d'une entité peuvent être appelées à interagir sur un même fragment de territoire ou sur un même objet. Pour palier à ce problème, les auteurs proposent un mécanisme de verrouillage pour s'assurer qu'un seul serveur à la fois n'accède aux données d'une région donnée. Deux types de verrous sont proposés : les verrous de régions, qui permettent à un serveur donné de demander un accès exclusif à une région donné de l'espace physique afin d'exécuter un événement ayant un impact sur une grande quantité de territoire. Le second type est le verrou d'objet, qui

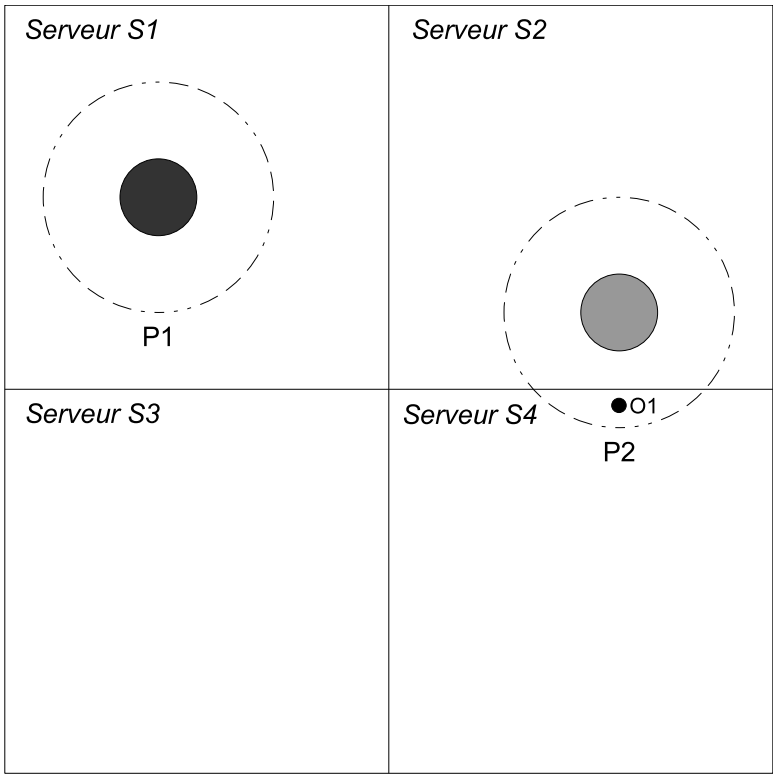


FIGURE 2.1 Assignation de joueurs aux serveurs selon leur zone d'intérêt

permet à un serveur donné de demander un accès exclusif à un objet situé hors de son territoire de couverture (mais situé dans la zone d'intérêt d'un joueur sous sa responsabilité) afin d'exécuter un événement sur cet objet (par exemple, le ramasser, l'utiliser, etc.). Ce scénario est exprimé dans la figure 2.1 : dans cette situation, on suppose que le joueur P_2 géré par le serveur S_2 veut accéder à l'objet O_1. Puisque l'objet O_1 est situé dans le territoire géré par le serveur S_4, le serveur S_1 devrait demander au serveur S_4 d'obtenir un verrou sur l'objet O_1 afin que le joueur P_2 puisse exécuter l'événement approprié et empêcher que tout autre événement concernant l'objet O_1 ne soit généré simultanément.

Les auteurs proposent différents algorithmes tirant parti des verrous afin de gérer correctement et de façon synchronisée les événements affectant plusieurs serveurs ainsi que les transferts d'objets et les déplacements de personnages entre plusieurs serveurs.

2.2.3 Assignation dynamique de microcellules

De Vleeschauwer *et al.* (2005) proposent une façon intéressante de répartir le territoire du jeu entre plusieurs hôtes. L'architecture ne précise pas, cependant, si les différents hôtes sont des serveurs dédiés ou des micro-serveurs s'exécutant sur les machines des joueurs. Or,

même si l'architecture présentée ici n'a pas été définie spécifiquement pour utilisation au sein d'un réseau pair-à-pair, elle pourrait facilement être adaptée à ce contexte.

Tel que vu précédemment, le jeu *Second Life* répartit de façon géographique le territoire en une multitude de zones carrées contigüées de superficie fixe. Cette méthode, bien que plus simple, possède l'inconvénient majeur qu'une distribution inégale de la charge peut survenir. En effet, dans les jeux massivement multijoueurs en ligne, il existe des zones de forte population où il peut y avoir une très grande quantité de joueurs concentrés sur un espace très réduit (par exemple, une ville). À l'inverse, il existe également des zones vastes dénuées d'intérêt pour lesquelles il risque d'y avoir très peu de joueurs. En se basant sur la cartographie géographique du monde virtuel, il est possible pour l'opérateur du jeu de prévoir à l'avance les zones qui susciteront plus et moins d'intérêt et de prévoir un découpage initial en zones (appelées *cellules*) distinctes. Chacune des cellules peut donc être dimensionnée de façon variable afin afin d'avoir une densité de population constante. Ainsi, chaque serveur attitré à la gestion d'une cellule aurait une charge équivalente.

Un problème important avec cette approche découle de la nature dynamique du jeu. En effet, puisque le jeu et le monde virtuel évoluent selon les actions posées par les utilisateurs, il est impossible de prévoir exactement les positions de chacun des joueurs et les événements qui seront posés par ces derniers. Par exemple, dans une zone où il y a habituellement une faible densité de population, un événement auquel participeraient une très grande quantité de joueurs pourrait avoir lieu, ce qui pourrait potentiellement surcharger le serveur responsable de la cellule appropriée. Dans leur article, les auteurs proposent une façon de découpler le territoire avec une granularité beaucoup plus fine. Il s'agit de diviser le territoire global en zones de tailles assez petite appelées *microcellules*. Chacune de ces microcellules se retrouve assignée à un des serveurs disponibles. Bien entendu, considérant la taille réduite et la quantité importante de microcellules, un serveur sera appelé à en gérer plusieurs. Il s'agit donc ici d'optimiser l'assignation des cellules aux serveurs afin de réduire les délais et la charge éprouvée par chaque serveur [4]. La figure 2.2 représente un exemple de monde virtuel divisé en microcellules assignées à différents serveurs fictifs.

Les auteurs proposent ici un modèle mathématique permettant d'exprimer la charge sur un serveur donné (c'est-à-dire la charge totale de l'ensemble des cellules gérées par ce serveur) en fonction de différents paramètres. Ces paramètres prennent en compte, pour chaque cellule, le niveau d'activité moyen des joueurs et le nombre de joueurs ainsi que la charge due aux différentes intéractions et déplacements entre les cellules voisines (pour les cellules gérées par le même serveur et gérées par un serveur différent). Les auteurs étudient ensuite

4. Ce problème est très similaire au problème d'assignation des cellules aux commutateurs dans un réseau de téléphonie cellulaire

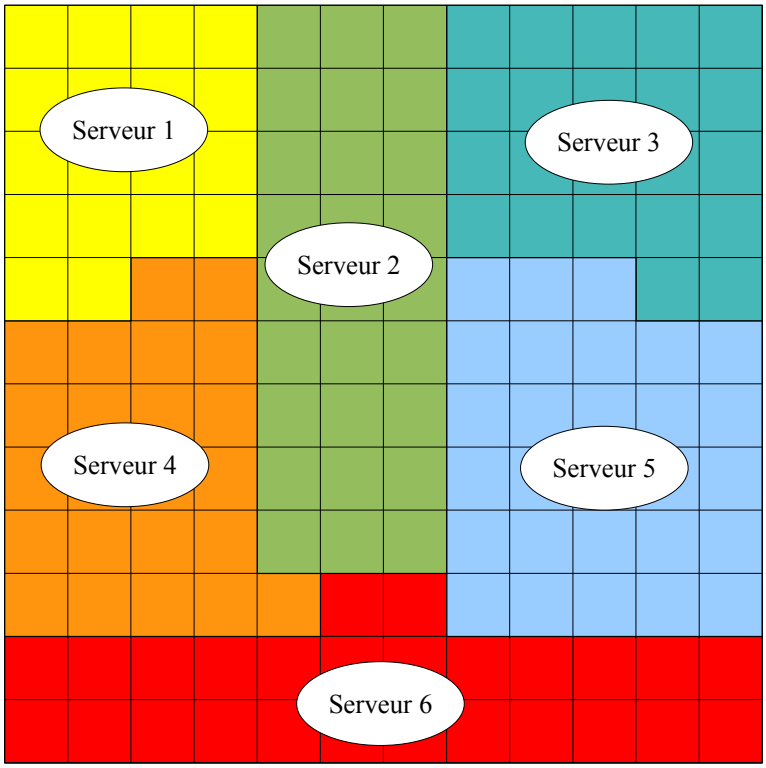


FIGURE 2.2 Répartition et assignation de microcellules

plusieurs algorithmes permettant d'optimiser la répartition des cellules aux commutateurs afin de minimiser la fonction de charge sur l'ensemble des serveurs. Ces différentes approches incluent d'abord des stratégies d'optimisation voraces et de déploiement en grappes. Puis, afin d'éviter le piège des minima locaux, un algorithme de recuit simulé est présenté. Enfin, un algorithme faisant appel à la des techniques de programmation linéaire est donné. Notons toutefois que le dernier cas est le seul à garantir une solution optimale, mais il s'avère impraticable dans la réalité puisqu'il s'agit d'un problème NP-complet (ne pouvant être résolu en temps polynomial). Les auteurs comparent ensuite chacun des algorithmes présentés afin de comparer la solution obtenue par rapport aux différents paramètres intiaux. Cependant, les différentes approches présentées dans cet article ne s'appliquent qu'à des paramètres statiques. Dans le contexte d'un véritable jeu, les différents paramètres propres à chaque cellule risquent fort probablement de varier en fonction des actions et événements posés par les joueurs. Les auteurs proposent en ouverture d'étudier comment il pourrait être possible de rebalancer dynamiqement les assignations de microcellules aux différents serveurs en prenant en compte la nature dynamique des paramètres.

2.3 Répartition dynamique

Une architecture réseau intégrant une répartition dynamique du monde virtuel implémente tout d'abord un schéma de répartition statique afin de définir un découpage initial avant l'exécution du jeu. Par la suite, durant l'exécution du jeu, le schéma d'assignation initial sera modifié au besoin selon les changements survenant au sein du monde virtuel, afin d'optimiser les paramètres et de rebalancer la charge réseau à chaque intermédiaire. Les sections qui suivent présentent différentes approches qui intègrent une répartition dynamique.

2.3.1 Plateforme de redéploiement dynamique de microcellules

Dans un second article publié sur le sujet, Van Den Bossche *et al.* (2006) proposent une architecture logicielle permettant d'étendre le concept de répartition géographique du territoire à un ensemble de serveurs gérant des microcellules. Ils proposent les bases architecturales d'un système et de protocoles permettant de relocaliser en temps réel les microcellules d'un serveur à l'autre afin d'accomoder les variations au niveau de la charge des serveurs. Les auteurs ont opté pour la plate-forme J2EE puisqu'elle est bien adaptée à la conception modulaire d'applications réseau distribuées.

L'architecture proposée par les auteurs intègre différents modules logiciels. Au sein de chaque serveur (qui gère un certain nombre de microcellules), on retrouve un contrôleur de microcellules qui est le principal responsable de la gestion de la cellule. Le contrôleur de microcellules traites tous les événements et actions se produisant à l'intérieur de chacune des microcellules assignées au serveur. Au sein de chaque serveur, il y a également un contrôleur d'acteurs qui gère l'ensemble des communications avec les différents participants au jeu (joueurs). Ce module gère entièrement les messages échangés, mais n'effectue aucun traitement. Il transmet plutôt les requêtes au controlleur de microcellules qui effectue le traitement et retourne ensuite la réponse.

Une caractéristique importante concernant les microcellules est qu'elles sont dotées d'une taille très réduite, ce qui implique que les joueurs auront à entrer en relation ou se déplaceront fréquemment vers les cellules voisines. Le fait de gérer plusieurs microcellules en utilisant un même contrôleur permettra donc de réduire les échanges de messages, en supposant que les microcellules avoisinantes soient regroupées au sein du même serveur [5].

L'architecture proposée possède également une entité centrale, le gestionnaire de microcellules, responsable de superviser le bon déroulement du jeu et de balancer la charge réseau

5. Notons qu'avec les algorithmes de répartition discutés précédemment proposés par De Vleeschauwer *et al.* (2005), les critères d'optimisation feront en sorte qu'un regroupement de cellules avoisinantes sera fréquemment réalisé.

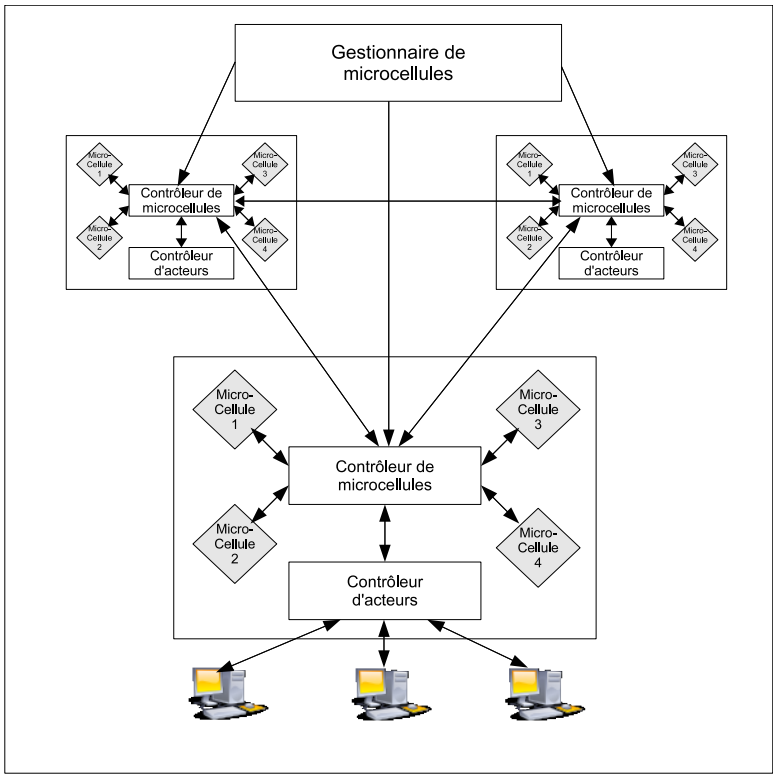


FIGURE 2.3 Architecture logicielle de redéploiement dynamique de microcellules

afin d'accomoder les variations pouvant survenir en cours de partie. Ainsi, si un serveur devient trop surchargé (par exemple, si plusieurs joueurs se déplacent dans le territoire des microcellules gérées par ce serveur), le gestionnaire de microcellules peut procéder à une réorganisation, c'est-à-dire migrer une ou plusieurs microcellules d'un serveur à un autre. De cette façon, la charge pourra être rééquilibrée de façon dynamique et en temps réel par rapport aux événements survenant dans le monde virtuel. Tel que le stipule les auteurs, il convient de préciser que le gestionnaire de microcellules ne participe pas à la gestion du jeu en tant que tel. Il joue uniquement un rôle de régulation et d'équilibrage. Par conséquent, dans l'éventualité où un problème concernant ce composant surviendrait, le jeu continuerait tout de même de fonctionner ; seul l'auto-rééquilibrage des microcellules serait inhibé (statique).

La migration des microcellules d'un serveur à un autre soulève certains défis au niveau du maintien de la consistence et de la synchronisation. Les auteurs proposent un processus de relocalisation en deux étapes : la relocalisation et le reroutage. Afin de mieux illustrer ce processus, considérons une microcellule C étant transférée d'un serveur S_A à S_B. La relocalisation consiste d'abord à créer un réplicata de la microcellule C sur S_B. Toutefois, il est possible que des événements continuent de se produire durant l'intervalle de temps associé à la relocalisation. Pour palier à ce problème, S_A, qui est toujours responsable de

la microcellule, continuera de recevoir et traiter les événements ; il avisera simplement S_B de l'arrivée de ces nouveaux événements. Une fois la relocalisation complétée, S_B appliquera simplement l'ensemble des événements [6] et sera considéré à jour. À ce moment, S_B prendra le relais et le contrôle de C sera transféré de S_A à S_B. La phase de reroutage consiste à informer les cellules avoisinantes ainsi que les joueurs situés géographiquement dans le territoire de C de se reconnecter au nouvel hôte. Ce processus n'est pas instantané ; il faut par conséquent considérer le fait que les utilisateurs se connecteront graduellement au nouveau serveur S_B. Il faut donc, le temps de la phase de reroutage, que l'ancien serveur S_A [7] continue de recevoir les événements des joueurs n'ayant pas encore effectué la transition. Toutefois, le contrôleur de microcellules de S_A ne traitera aucun événement ; il les transmettra plutôt au contrôleur de microcellules de S_B. À l'inverse, le contrôleur de microcellules de S_B, connaissant la liste des joueurs n'ayant pas effectué la transition, transmettra les messages destinés à chacun de ces joueurs au contrôleur de microcellules de l'ancien serveur S_A (qui le transmettra à son tour au moyen de son contrôleur d'acteurs). Une fois l'ensemble des joueurs migrés et l'ensemble des messages en transit acheminés, la phase de reroutage sera considérée complétée, terminant ainsi la migration de la microcellule C. Le serveur S_A pourra donc disposer des données relatives à C puisque S_B aura pris le relais entièrement.

2.3.2 Découpage basé sur la zone d'intérêt

Ahmed et Shirmohammadi (2008) proposent une approche différente : un modèle hybride dans lequel le territoire global du monde virtuel n'est aucunement découpé initialement. Les auteurs font plutôt appel ici à la notion de zone d'intérêt pour induire un découpage dynamique selon la position des joueurs. La zone d'intérêt peut être différente pour chaque entité puisque par exemple, dans le contexte d'un jeu, un joueur dans un hélicoptère pourrait voir plus loin qu'un joueur régulier.

Si plusieurs joueurs sont situés à proximité, il est fort possible que leur zone d'intérêt se chevauche. Les auteurs suggèrent de regrouper dynamiquement les joueurs dont la zone d'intérêt mutuelle se superpose afin de former une région délimitée par la position de chacun de ces joueurs. Pour chaque région ainsi délimitée, selon la prémisse que la très grande majorité des communications et échanges seront réalisés au sein de la région délimitée (puisque cette région englobe la zone d'intérêt et par conséquent la zone d'intéraction de l'ensemble des joueurs la composant), les auteurs proposent d'intégrer un modèle de communication pair-à-pair au sein de ladite zone. Les joueurs pourraient donc négocier entre eux les événements,

6. En réalité, l'ensemble des événements à appliquer constitue la différence (ou *delta*) entre l'état reçu de S_A et l'état *actuel*.

7. Au moyen de son contrôleur d'acteurs qui redirige ensuite les requêtes au contrôleur de microcellules

réduisant ainsi la charge serveur. Un membre élu serait alors responsable de notifier le serveur central des changements d'états.

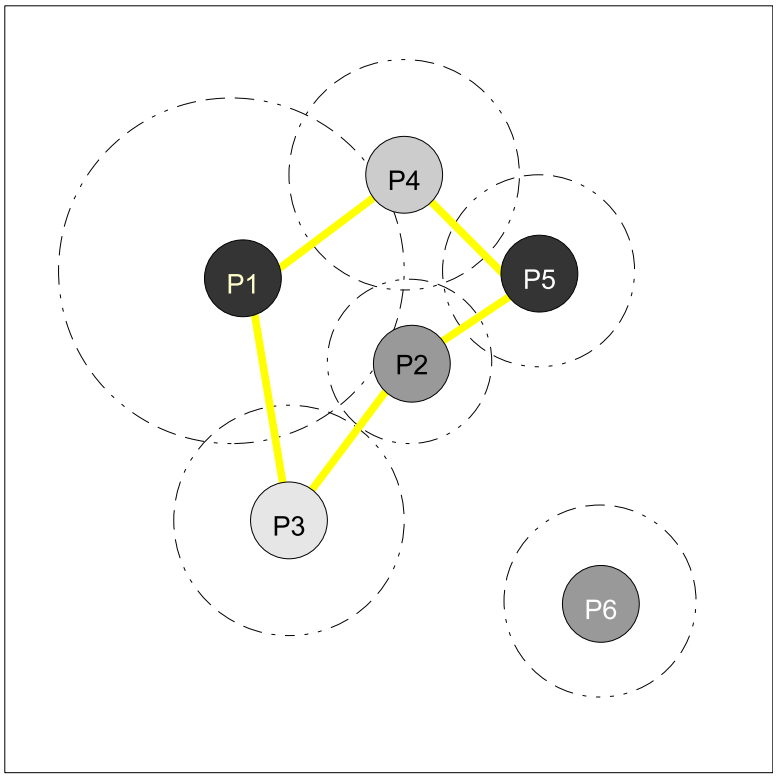


FIGURE 2.4 Région délimitée par des joueurs dont la zone d'intérêt se chevauche

Ainsi, pour l'ensemble des joueurs présents au sein du jeu, le serveur central générera un ensemble d'enveloppes convexes mettant en commun les différents joueurs dont la zone d'intérêt se recoupe. La figure 2.4 illustre cette approche pour six joueurs au sein d'un monde virtuel, P_1 à P_6. Dans cette figure, puisque les zones d'intérêt pour les joueurs P_1 à P_5 se recoupent, une enveloppe convexe englobant tous ces joueurs est formée. Les intéractions entre ces joueurs seront donc négociées en mode pair-à-pair et un noeud sera choisi pour rapporter les changement d'état concernant les membres de l'enveloppe au serveur, économisant ainsi les ressources.

Pour chaque enveloppe convexe générée, les joueurs (noeuds) situés sur la bordure auront la responsabilité de réajuster la frontière selon leurs déplacements. Toutefois, la génération d'une enveloppe demande une grande quantité de ressources. Pour éviter un trop grand nombre de recalculs si les joueurs ne se déplacent que sur de courtes distances, les auteurs proposent l'utilisation d'une marge de sécurité (*safety-edge space*) qui "élargit" légèrement la bordure. Ainsi, pour chaque joueur, un recalcul de l'enveloppe convexe ne sera nécessaire que si le joueur en question traverse complètement la bordure. À cette restriction, les auteurs

ajoutent également un intervalle temporel minimal (*time-span*) qui empêche les recalculs trop fréquents.

Considérant que la latence influence grandement la qualité d'un jeu multijoueurs et que les connexions pair-à-pair sont plus à risque d'en subir les impacts, les auteurs proposent un algorithme *min-max* permettant d'obtenir une configurtion qui minimise la latence maximale afin d'optimiser l'organisation de la couche pair-à-pair au sein de chaque zone (enveloppe convexe).

2.3.3 Hydra

Chan *et al.* (2007) présentent une architecture hybride de jeux MMORPG qui vise à abstraire complètement la composante pair-à-pair et à exposer les services réseau en simulant le modèle traditionnel. Hydra fonctionne selon un modèle de découpage basé sur les régions physiques du jeu. Un serveur donné (appelé simulateur) est responsable de chacune des régions et reçoit en continu des messages de la part des clients. Chaque message possède un "timbre temporel" (*timestamp*) exprimé en unité discrètes (*tick*) de temps du jeu. Afin de garantir un état du jeu cohérent, chaque simulateur Hydra doit implémenter trois principes fondamentaux :

1. **Mise en pause du simulateur.** Le simulateur, qui exécute en continu une boucle de traitement des messages, traitera tous les messages dont le timbre temporel est inférieur au timbre temporel actuel. Les messages reçus dont le timbre temporel est supérieur au timbre temporel actuel sont placés en attente, ce qui peut provoquer la mise en pause du simulateur.
2. **Déterminisme de simulation.** L'ordre temporel est cohérent, ce qui implique que tout message transmis par le simulateur sera transmis immédiatement après le traitement en lot de tous les messages à traiter, et sera cohérent avec le nouvel état du simulateur.
3. **Interface de sauvegarde/restauration.** Cette exigence assure que l'état exact du simulateur peut être restauré à un instant donné en cas de panne.

L'état de chacune des régions du territoire global est encapsulé au sein de composants appelés "tranches" (*slices*), qui sont gérées par un serveur Hydra (un serveur peut gérer plusieurs tranches). L'architecture inclut également un noeud de rendez-vous qui permet aux noeuds de repérer les tranches correspondant à leur position actuelle afin de s'y connecter. L'abstraction du réseau pair-à-pair est réalisée par l'ajout d'un composant supplémentaire, le proxy client. Ce composant joue le rôle de médiateur entre le client et les infrastructures réseau de gestion (tranches). Les clients (noeuds joueurs) transmettent et reçoivent leurs

messages uniquement à partir du proxy, ce qui permet de simuler le paradigme client-serveur existant. Afin de garantir un niveau de fiabilité accru, deux types de tranches sont définies :

Tranche primaire (*primary slice*). Cette entité est responsable de la gestion des messages et de la mise à jour du timbre temporel au sein de la tranche. Les messages reçus ainsi que timbre temporel actuel sont automatiquement retransmis aux tranches de sauvegarde correspondantes.

Tranche de sauvegarde (*backup slice*). Cette entité maintient une copie de sauvegarde de l'état du jeu. Elle reçoit les messages à la fois du proxy client et de la tranche principale. Elle peut alors détecter si une déconnexion ou une perturbation a eu lieu et maintenir un état cohérent, synchronisé avec la tranche principale. Elle ne maintient pas à jour automatiquement son timbre temporel : elle utilise uniquement le timbre reçu de la tranche principale.

Une région donnée est associée à une tranche principale et peut être associée à un nombre arbitraire de tranches de sauvegarde selon la fiabilité voulue. Grâce à ce mécanisme, la récupération en cas de défaillance est possible. Ainsi, advenant l'échec d'une tranche (détecté par un dépassement de temps (*timeout*)), cette dernière peut automatiquement être reconstruite à partir de l'état synchronisé d'une autre tranche. Dans le cas de l'échec d'une tranche primaire, une nouvelle tranche primaire est simplement sélectionnée parmi les tranches de sauvegarde au moyen un protocole d'élection.

2.3.4 Micro-architectures pair-à-pair

Knutsson *et al.* (2004) présentent une architecture hybride de gestion de jeux MMOG. Cette architecture intègre une composante peer-to-peer pour assigner les tâches de micro-gestion aux joueurs eux-mêmes, ainsi qu'un serveur centralisé pour réaliser les tâches de gestion (facturation, persistence des profils des joueurs) et l'organisation des micro-réseaux pair-à-pair. Cela permet ainsi à l'opérateur de déléguer certaines tâches qui consomment beaucoup de ressources sans toutefois perdre le contrôle sur le jeu.

Les auteurs définissent les classes de gestion suivantes :

Gestion l'état des joueurs. La gestion de l'état des joueurs traite de tout ce qui a trait aux actions effectuées par les joueurs eux-mêmes (déplacements, échanges, discussions, etc.). Les intéractions s'effectuent généralement entre les joueurs concernés (unicast). Certaines actions peuvent cependant concerner plusieurs joueurs, tel que la mise à jour de la position (déplacement d'un joueur) ; dans un tel cas, les messages sont diffusés à tous les joueurs avoisinants (multicast), selon un intervalle de temps donné. Bien que la garantie de livraison des messages multicasts ne soit pas assurée, la perte de

certains messages est mitigée par le fait qu'un message subséquent viendra "corriger" la situation. La diffusion des messages multicasts utilise l'infrastructure Scribe.

Gestion des objets. Un objet réfère à toute entité du jeu avec laquelle un joueur peut entrer en intéraction. La gestion des objets est réalisée avec l'infrastructure Pastry. Chaque objet (possédant un identificateur donné) est assigné à un coordonateur, qui est le joueur dont l'identificateur est le plus près [8]. Lorsqu'un noeud intéragit avec un objet, il passe par son coordonateur. Ce dernier résoud également les conflits, advenant le cas un objet serait mis à jour simultanément par plus d'un joueur.

En ce qui concerne la carte géographique du jeu et tous les élément multimédias (sons, modèles, textures, etc.), les auteurs supposent que ces derniers feront partie de l'installation de base du jeu et ne seront pas transférés dynamiquement.

Dans l'implémentation proposée et réalisée par les auteurs, ces derniers proposent d'utiliser Pastry pour associer chacune des régions du jeu à un joueur donné qui deviendra le coordonateur de cette région. La probabilité que le coordonateur ne soit pas situé dans la région attitrée est très élevée, ce qui peut réduire considérablement la propension à tricher. Pour fins de simplification, tous les objets présents dans ladite région seront également sous le contrôle du même coordonateur. Néanmoins, les auteurs mentionnenent qu'il pourrait être possible d'assigner des identificateurs Pastry distincts aux objets afin de les assigner à des coordonateurs différents si la charge devenait trop lourde.

Certains problèmes de perte de données peuvent survenir puisque la fiabilité des noeuds est moindre que celle de serveurs dédiés. Toutefois, les auteurs supposent que le taux de défaillance sera considérablement réduit par certains facteurs tels que (1) l'indépendance de la répartition logique des noeuds au sein de Pastry par rapport à leur réelle répartition géographique et (2) les capacités de récupération en cas de défaillance de Pastry qui feront en sorte qu'un message arrivera avec une grande probabilité à sa destination même si plusieurs noeuds sont brusquement déconnectés.

Un problème potentiel concerne l'éventualité où le coordonateur d'une région donnée subirait une défaillance, puisque ce dernier est responsable de la gestion centralisée de la région au complet. Pour mitiger ce problème, les auteurs suggèrent l'ajout d'un second coordonateur (ou de n coordonateurs supplémentaires selon la fiabilité voulue). Le coordonateur-réplique contiendra et maintiendra à jour l'état complet de la région et tous ses objets. Ce rôle sera attribué au second joueur dont l'identificateur sera le plus près numériquement (selon Pastry) de l'identificateur de la région. Les auteurs discutent de trois scénarios possibles :

1. **Le coordonateur subit une défaillance.** Les messages destinés au coordonateur seront automatiquement acheminés au coordonateur-réplique puisque ce dernier est

8. En termes de distance numérique selon Pastry

maintenant le noeud dont l'identificateur est situé le plus près numériquement de l'identificateur de la région. Le coordonateur-réplique se retrouve donc informé qu'il doit prendre la relève. Un nouveau coordonateur-réplique est alors choisi selon le principe mentionné précédemment.

2. **Un nouveau noeud devient un coordonateur.** Cette situation peut se produire si un nouveau noeud rejoignant le réseau Pastry possède un identificateur numérique plus près de l'identificateur de la région par rapport à l'identificateur numérique du coordonateur actuel ; afin de préserver le principe de routage basé sur la distance de Pastry, le coordonateur actuel doit transférer son rôle au nouveau noeud. Le coordonateur actuel devient le coordonateur-réplique et le coordonateur-réplique actuel perd son rôle.
3. **Un nouveau noeud devient un coordonateur-réplique.** Cette situation peut se produire si un nouveau noeud rejoignant le réseau Pastry possède un identificateur numérique plus près de l'identificateur de la région par rapport à l'identificateur numérique du coordonateur-réplique actuel (sans toutefois être *suffisament près* pour devenir coordonateur). Le coordonateur-réplique actuel perd alors son rôle.

Un problème important subsiste toutefois en utilisant cette approche : la possibilité de perte de consistence advenant la perte du coordonateur et des coordonateurs répliques. Cette situation pourrait se produire si plusieurs défaillances successives font en sorte que tous les noeuds associés à la gestion d'une région donnée subissent une déconnexion sans avoir le temps de passer le contrôle à d'autres noeuds Pastry. Cette situation est toutefois extrêmment rare, mais des simulations effectuées ont démontré qu'elle demeure toutefois possible et qu'il est important d'envisager des solutions pour y rémédier. Une solution proposée par les auteurs consiste à faire en sorte que le serveur central soit impliqué dans la sélection des coordonateurs et puisse prendre en charge les joueurs advenant un tel scénario, même si cela pourrait réduire les performances du réseau pair-à-pair "autosuffisant" de Pastry.

2.3.5 MOPAR

Mopar est une architecture de jeux MMOG entièrement pair-à-pair proposée par Yu et Vuong (2005). Elle utilise simultanément les modèles pair-à-pair structuré (tables de hachage distribuées) et non-structuré. Le monde virtuel est initialement découpé en régions hexagonales de taille constante. Il n'y a aucun serveur central ; le jeu est géré uniquement par les noeuds-joueurs. Les auteurs définissent trois rôles possibles pour chacun des noeuds :

Noeud domicile (*home node*)

Un noeud domicile est assigné à chaque région hexagonale, selon un réseau pair-à-pair structuré utilisant Pastry. Le noeud domicile est le noeud dont l'identificateur numérique est le plus près numériquement de l'identificateur d'une région donnée. De ce fait, un noeud domicile pourrait être assigné à plusieurs régions différentes. Ce noeud a pour fonction de conserver et maintenir à jour le noeud maître associé à chacune des régions attitrées à ce noeud.

Puisque l'assignation du noeud domicile est basée sur une répartition logique (Pastry) et est indépendante de la position physique du noeud dans le monde virtuel, la probabilité que le noeud soit physiquement situé dans sa région attitrée est très faible. Une conséquence du fonctionnement de Pastry est qu'un nouveau noeud joignant le monde virtuel pourrait à tout moment remplacer un noeud domicile existant si son identificateur numérique se rapproche davantage de l'identificateur d'une région donnée.

Les auteurs prévoient également la mise en place possible d'un mécanisme de redondance. Ce mécanisme consiste à conserver un nombre donné k (constant) de noeuds domicile secondaires qui contiennent les mêmes informations que le noeud domicile principal. Ces noeuds secondaires sont les k noeuds dont les identificateurs sont les plus près numériquement de l'identificateur de la région, et seront utilisés en cas de défaillance du noeud domicile principal. Ce mécanisme peut diminuer significativement le risque d'incohérence par isolation dans le monde virtuel.

Noeud maître (*master node*)

Chaque région possède un noeud maître qui est un des noeuds présentement situés dans la région. Ce noeud est responsable de la gestion de la région. Il gère l'arrivée et le départ de noeuds. Il peut prédire le déplacement des autres noeuds puisqu'il connait la fonction de déplacement de chacun des noeuds. Il est également relié à tous les autres noeuds maîtres des régions avoisinantes, ce qui permet de faciliter les transferts entre régions et de gérer les zones d'intérêt chevauchant plusiers régions.

Lorsqu'un noeud joint une région qui ne contient pas de noeud maître, ce dernier devient automatiquement le noeud maître. Si noeud maître quitte une région donnée, il délègue la responsabilité à un des noeuds esclaves et avise le noeud domicile afin que ce dernier puisse mettre à jour ses enregistrements.

Noeud esclave (*slave node*)

Chaque noeud rejoignant une région donnée du monde virtuel qui possède déjà un noeud maître devient un noeud esclave. Le nouveau noeud reçoit du noeud maître la liste des autres noeuds présentement situés dans sa zone d'intérêt [9]. Les échanges entre noeuds dans une même zone d'intérêt sont réalisés de façon pair-à-pair, sans l'intermédiaire du noeud maître [10].

Le déplacement des noeuds suit une fonction donnée qui permet de prédire la position du noeud à un instant donné. Puisque le noeud maître connaît la fonction de déplacement des noeuds esclaves, ces derniers n'ont pas besoin d'aviser à chaque changement de position. Ils transmettent au noeud maître uniquement les changements à la fonction de déplacement.

9. Par souci de simplification, les auteurs assument que la zone d'intérêt de chaque noeud est de taille constante.

10. L'article ne précise pas de détails supplémentaires concernant l'organisation de ce sous-réseau pair-à-pair non-structuré.

Chapitre 3

ARCHITECTURE P2P PROPOSÉE

Tel que vu précédemment, plusieurs architectures ont été proposées pour améliorer la performance et optimiser la répartition de la charge au sein de jeux massivement multi-joueurs en ligne. Cependant, aucune des approches de répartition géographique présentées précédemment ne permet d'ajuster dynamiquement la taille des zones pour s'adapter en temps réel au flot des joueurs.

L'architecture proposée dans ce travail peut être qualifiée de *hybride* puisqu'il y a présence d'un serveur central pour coordonner à haut niveau le fonctionnement de la sous-couche peer-to-peer. L'objectif du projet est de proposer une architecture pair-à-pair permettant de prendre en charge un nombre significatif de joueurs au sein d'un même univers virtuel tout en s'assurant que la qualité de jeu ne soit pas affectée par ce mode opératoire.

3.1 Hypothèses

Afin d'améliorer la compréhension et la concision, nous définissons ici certaines hypothèses qui permettront de simplifier certains aspects du modèle proposé.

3.1.1 Territoire de jeu

Nous supposons que le territoire de jeu possède une forme rectangulaire. Il pourrait être possible sans beaucoup d'efforts d'appliquer notre modèle à des territiores de d'aspects différents ou plus complexes puisque toute surface géométrique peut se diviser en un ensemble de primitives (triangles). Toutefois, à notre sens, cela n'apporterait pas beaucoup de valeur ajoutée au travail pour la complexité générée.

Mentionnons également que le territoire du jeu ne permet pas de se déplacer d'une extrémité à l'autre (ie. la superficie du jeu n'est pas "sphérique"). Par exemple, un joueur situé à l'extrémité droite qui souhaite aller à l'extrémité gauche doit traverser le territoire de droite à gauche ; il ne peut pas aller vers la droite pour réapparaître à gauche. Cette simplification induira une concentration de joueurs un peu plus importante vers le centre puisque les joueurs devront passer par là pour aller d'une extrémité à l'autre. Cependant, dans nos simulations, nous modélisons déjà des zones de répartition inégale à l'aide des régions d'intérêt

qui seront explicitées à la section 3.2. Par conséquent, nous pensons que cette simplification est parfaitement acceptable.

Une autre hypothèse est que toutes les zones ont le même coût de gestion par unité d'aire. Normalement, dans le cadre d'un vrai jeu, on pourrait s'attendre à ce que certaines zones contenant plus d'éléments de jeu aient un coût de maintenance plus élevé et inversement. Dans notre modèle, nous proposons d'uniformiser ces coûts puisque modéliser des coûts non-uniformes demanderait une part significative de paramètres supplémentaires à intégrer pour peu de valeur ajoutée.

3.1.2 Noeuds joueurs

Nous supposons que les joueurs se déplacent selon le modèle *Hotspots Random Waypoint*. Ce modèle, qui sera décrit à la section 3.2, admet uniquement des déplacements et des temps de pauses aléatoires en prenant en compte les régions d'intérêts, ce qui simplifie beaucoup le comportement réel de joueurs qui peuvent normalement effectuer beaucoup d'actions autres en cours de jeu. Modéliser ces actions n'est toutefois pas essentiel à notre modèle.

Nous supposons également que les noeuds joueurs conservent la même capacité de charge assignée initialement et que les valeurs de latence entre les paires de noeuds ne changent pas au fil de la simulation. Cette simplification simplifie beaucoup l'expression mathématique et la validation du modèle.

3.1.3 Rebalancement

Nous supposons un temps de rebalancement nul. Les rebalancements des zones au sein de la sous-couche pair-à-pair s'effectuent de façon instantanée. Normalement, il faudrait considérer un temps de relève pour tous les noeuds concernés, mais nous en faisons abstraction dans notre modèle par souci de simplification.

Nous assumons que les déconnexions des noeuds joueurs sont "souples" et non "dures", ce qui implique que nous considérons que les noeuds joueurs peuvent transmettre leur état avant de quitter. La prise en charge de déconnexions dures aurait demandé d'implémenter des mécanismes de redondance et de recouvrement en cas d'erreur, ce qui est hors de la portée de ce travail.

3.2 Joueurs et mobilité

Les jeux massivement multijoueurs en ligne prennent en charge une très grande quantité de joueurs au sein d'un même univers virtuel. Afin d'acroître le réalisme, il est important de

modéliser correctement le comportement typique d'un joueur, puisque ce comportement sera simulé en milliers d'exemplaires.

3.2.1 Mobilité

La mobilité d'un noeud est souvent modélisée grâce au patron de mobilité *Random Waypoint* Ariyakhajorn *et al.* (2006), notamment au sein des réseaux mobiles ad-hoc (MANET). Ce patron sert à modéliser un schéma de mobilité possible pour le noeud. L'algorithme 1 illustre son mode de fonctionnement. Les paramètres à considérer sont décrits au tableau 3.1.

Algorithme 1 (Fonctionnement du modèle *Random Waypoint*)

1. *Sélectionner un point de destination aléatoire et se déplacer vers ce point à une vitesse constante choisie aléatoirement (distribution uniforme)*
2. *Effectuer une pause d'une durée aléatoire (distribution uniforme)*
3. *Retourner à l'étape 1*

FIGURE 3.1 Fonctionnement du modèle *Random Waypoint*

TABLEAU 3.1 Paramètres du modèle *Random Waypoint*

Paramètre	Description
A	Extrémités de l'univers (x_{min}, x_{max}, y_{min}, y_{max})
v_{min}	Vitesse minimale pour la génération de action aléatoires
v_{max}	Vitesse maximale pour la génération de action aléatoires
p_{min}	Temps de pause minimal suivant chaque action de déplacement
p_{max}	Temps de pause maximal suivant chaque action de déplacement

Il a été pensé d'adopter ce patron de mobilité dans le contexte de jeux massivement multijoueurs en ligne. Cependant, puisque les points de destination des joueurs sont choisis aléatoirement, il y aura une distribution à peu près constante de joueurs sur tout le territoire. La densité sera toutefois légèrement plus élevée vers le centre, puisque le modèle proposé prend en compte une surface de territoire planaire et non sphérique ; les joueurs ne peuvent pas traverser d'un côté à l'autre du territoire par les extrémités, tel que stipulé à la section 3.1. Dans la réalité, la densité de joueurs est inégale puisqu'il y a des régions qui suscitent plus d'intérêt que d'autres. Par exemple, une région associée à un "village" virtuel pourrait avoir beaucoup plus de joueurs qu'une région désertique. Van Den Bossche *et al.* (2009) proposent un modèle de mobilité admettant des densités de joueurs plus élevées dans certains endroits clés du jeu appelés "*Hotspots*". Ce modèle de mobilité constitue une extension du modèle de mobilité *Random Waypoint*. Pour fins de simplification, on considère un

nombre donné de régions circulaires (régions d'intérêt ou *Hotspot*) qui auront une probabilité *significativement* plus élevée d'attirer les joueurs. Les régions d'intérêt sont situées à des emplacements générés aléatoirement et possèdent un rayon également généré aléatoirement, le tout selon une distribution uniforme. Deux catégories de régions d'intérêt sont définies :

Statiques : ces régions d'intérêt sont fixes et demeurent en vigueur tout le long de la durée de la simulation. Ces régions peuvent modéliser des villages, quêtes, etc.

Dynamiques : ces régions d'intérêt apparaissent, demeurent en vigueur pour un temps donné et disparaissent aléatoirement. Ces dernières peuvent permettre de simuler des événements spéciaux attirant beaucoup de joueurs pour un laps donné.

Les paramètres propres à chacune des régions d'intérêt sont définis au tableau 3.2. Lorsqu'un noeud termine une action du mouvement (déplacement suivi du temps de pause), ce dernier génère une autre action. Avec l'utilisation du modèle *Hotspot Random Waypoint*, le choix du point de destination est biaisé de manière à avoir une probabilité de se retrouver à l'intérieur de l'une des régions d'intérêt définies. Le paramètre de poids (*weight*) propre aux régions d'intérêt sert à déterminer la *pondération* de cette région par rapport aux autres régions. Cela sert à introduire une gradation pour indiquer que certaines régions pourraient être plus importantes que d'autres. Il y a également un poids $W_{outsideHotspot}$ qui représente la pondération selon laquelle la prochaine destination ne sera pas située dans aucune région d'intérêt ; il s'agit d'un point aléatoire sur la carte (à la manière du modèle *Random Waypoint*).

Tableau 3.2 Paramètres propres aux régions d'intérêt

Paramètre	Description
P_i	Position dans l'univers virtuel
r_i	Rayon de la région circulaire
W_i	Poids relatif qui influence la probabilité que cette région d'intérêt soit sélectionnée lors de déplacements
t_i^e	Moment où cette région d'intérêt entrera en vigueur
t_i^d	Moment où cette région d'intérêt cessera d'être en vigueur

Lorsqu'un noeud termine un déplacement à l'intérieur d'une zone d'intérêt, il y a une forte probabilité ($p_{stayInsideHotspot}$) que le noeud choisisse un prochain point de destination situé à l'intérieur de cette même zone. Cela modélise le fait qu'un joueur peut vouloir rester plus longtemps à l'intérieur de la région d'intérêt, tout en se déplaçant vers un autre point situé dans cette même région. L'algorithme 2 modélise le modèle *Hotspot Random Waypoint* implémenté.

Algorithme 2 (Fonctionnement du modèle *Hotspot Random Waypoint*)

1. **Si** *le noeud est dans une région d'intérêt*
   (a) **Si** *le noeud demeure dans la région d'intérêt (déterminé aléatoirement selon la probabilité $P_{stayInsideHotspot}$)*
      i. *Générer un point de destination aléatoire dans la région d'intérêt*
   (b) **Sinon** *(le noeud ne demeure pas dans la région d'intérêt)*
      i. *Aller à l'étape 2a*
2. **Sinon** *(si le noeud n'est pas dans une région d'intérêt)*
   (a) *Déterminer le poids total (le poids de l'ensemble des régions d'intérêt et le poids associé à la sélection d'un point extérieur) :*
   $W_{total} = W_1 + W_2 + W_3 + \ldots + W_n + W_{outsideHotspot}$
   (b) *Sélectionner aléatoirement une région d'intérêt ou aucune région (point aléatoire). Pour toute région i de poids W_i, la probabilité de sélection est W_i/W_{total}. La probabilité de ne choisir aucune région est $W_{outsideHotspot}/W_{total}$.*
   (c) **Si** *Si une région d'intérêt a été sélectionnée*
      i. *Générer un point de destination aléatoire dans la région d'intérêt choisie*
   (d) **Sinon** *(point extérieur)*
      i. *Générer un point de destination aléatoire dans l'univers virtuel*
3. *Se déplacer à une vitesse constante aléatoire (distribution uniforme) en direction du point de destination déterminé*
4. *Effectuer une pause d'une durée aléatoire (distribution uniforme)*
5. *Retourner à l'étape 1*

FIGURE 3.2 Fonctionnement du modèle *Hotspot Random Waypoint*

3.2.2 Connectivité

Bien que les jeux massivement multijoueurs en ligne soient passionnants, les joueurs ne jouent pas en continu. Un joueur joue typiquement quelques heures, se déconnecte pour un laps donné et reprend éventuellement le jeu. Le but du projet n'est pas de modéliser de façon exacte le patron de connectivité de tels joueurs, mais bien de prendre en compte une certaine variation dans le temps. Une étude (Tarng *et al.* (2008)) a analysé différentes métriques de jeu auprès de 34 524 joueurs du jeu *World of Warcraft* (le plus populaire) sur une période de deux ans. Les auteurs ont notamment tiré une modélisation décrivant le nombre moyen de joueurs connectés en fonction de l'heure du jour. On constate un nombre de joueurs au plus bas durant les heures matinales (de 6h à 12h), qui augmente lentement durant l'après-midi et croît rapidement en soirée, pour atteindre son plafond autour de 22h. Par la suite, le nombre

de joueurs décroit au long de la nuit pour retourner à son minimum vers 6h. Sans chercher à reproduire de façon exacte ce schéma, il peut être intéressant de s'en inspirer pour voir dans quelle mesure l'algorithme proposé peut s'adapter aux connections / déconnections de joueurs.

La connectivité s'exprime comme un ajout au modèle de mobilité. L'extension apportée se situe au niveau de la génération d'actions de mouvement pour le noeud. Deux types d'actions deviennent possibles : une action de déplacement (déplacement typique d'un noeud selon le modèle *[Hotspot] Random Waypoint*) ou une action de *déconnexion* selon laquelle le noeud "quitte" le jeu pour une période donnée. Ainsi, lorsqu'une nouvelle action est générée, le noeud a une probabilité P_d de générer une action de déconnexion et une probabilité $1 - P_d$ de générer une action de déplacement standard. Cette probabilité est modulée par un facteur de pondération selon l'heure du jour (W_d^i où $i \in [0, 23]$). La durée de la période de déconnexion est choisie aléatoirement selon une distribution normale de moyenne σ_d et d'écart type μ_d. Le tableau 3.3 résume les paramètres des actions de déconnexion.

Tableau 3.3 Paramètres des actions de déconnexion

Paramètre	Description
σ_d	Durée moyenne durant laquelle un joueur reste déconnecté
μ_d	Écart type de la durée durant laquelle un joueur reste déconnecté
P_d	Probabilité de génération d'une action de déconnexion
W_d^i	Facteur modulant la probabilité de déconnexion selon l'heure du jour i

À l'état initial, tous les joueurs sont déconnectés, c'est-à-dire que la première action est obligatoirement une action de *déconnexion* (dont la durée sera variable pour chaque joueur). Les actions subséquentes sont aléatoires.

3.2.3 Capacité de charge et latence

Puisqu'il s'agit d'une architecture pair-à-pair, tout noeud joueur participant pourrait être appelé à jouer un rôle de serveur. Vu la grande diversité au niveau des capacités offertes par les machines clientes des joueurs (CPU, RAM, réseau), il convient de proposer une modélisation qui prend en compte ces variations. Pour exposer cette modélisation, chaque joueur se voit attribuer une capacité de charge, C_{max} , qui est la charge maximale que peut supporter ce joueur si ce dernier agit en tant que serveur. Par convention, on suppose que tant et aussi longtemps que la charge actuelle demeure en-dessous de la capacité de charge, il n'y aura aucune dégradation de performance.

De plus, de par la nature internationale de tels jeux, des joueurs provenant de différents lieux géographiques peuvent se retrouver à jouer ensemble, simultanément. Le délai de latence

entre les différents joueurs (communément référé sous l'appelation *ping*) est une métrique qui sera également considérée lors de l'optimisation.

3.3 Modèle de répartition

Le modèle de répartition pair-à-pair repose sur une répartition géographique. Cela signifie que le territoire global du jeu doit être subdivisé en un nombre donné de régions distinctes, qui seront ici appelées *zones*. Tel que décrit à la section 3.1, nous assumons que le territoire de jeu possède une forme rectangulaire. L'implémentation de la sous-couche pair-à-pair propose d'assigner la responsabilité de chaque zone à un noeud joueur. Le noeud deviendra alors responsable de gérer l'ensemble des activités des autres joueurs situés dans cette zone et l'ensemble des événements externes qui peuvent altérer l'état de la zone. Ces événements peuvent inclure, par exemple :

- la génération de nouvelles unités ennemies à combattre ;
- la modification du paysage ;
- l'apparition d'objets spéciaux.

Le noeud serveur rapporte périodiquement au serveur central les modifications apportées à l'état des joueurs et à l'état de la zone. Le fait de céder la gestion de la zone à un noeud externe permettra de réduire la charge imposée au serveur pour la gestion de cette zone. Ce dernier n'aura plus à gérer de façon microscopique les activités des joueurs et les événements de la zone ; il ne fera que recevoir de temps à autres des mises à jour sur les différences entre l'état précédent et l'état actuel. En obtenant un découpage de l'ensemble du territoire du jeu, il est possible d'assigner chaque parcelle distincte du territoire à un noeud-joueur donné. Le serveur central n'aura donc comme rôle que de coordonner chaque microserveur et de mettre à jour la base de données de l'état du jeu et des profils joueurs sur réception des mises à jour des différents coordonateurs.

Tel que vu au chapitre précédent, la plupart des modèles classiques de répartition géographique fonctionnent selon un découpage en zones prédéterminées (Assiotis et Tzanov (2006), De Vleeschauwer *et al.* (2005), Van Den Bossche *et al.* (2006)). Pour mieux tirer parti de ces modèles, il faut prévoir à l'avance la répartition globale de la charge de traitement requise au sein des différentes régions du jeu. Notre modèle propose plutôt d'effectuer un découpage géographique totalement dynamique pouvant être capable de se reconfigurer automatiquement pour s'adapter aux variations de la charge en cours de partie. Pour ce faire, des zones de forme triangulaire seront utilisées. En termes géométriques, la subdivision du territoire global en zones triangulaires revient donc à effectuer une *tesselisation* d'une surface (bidimensionnelle).

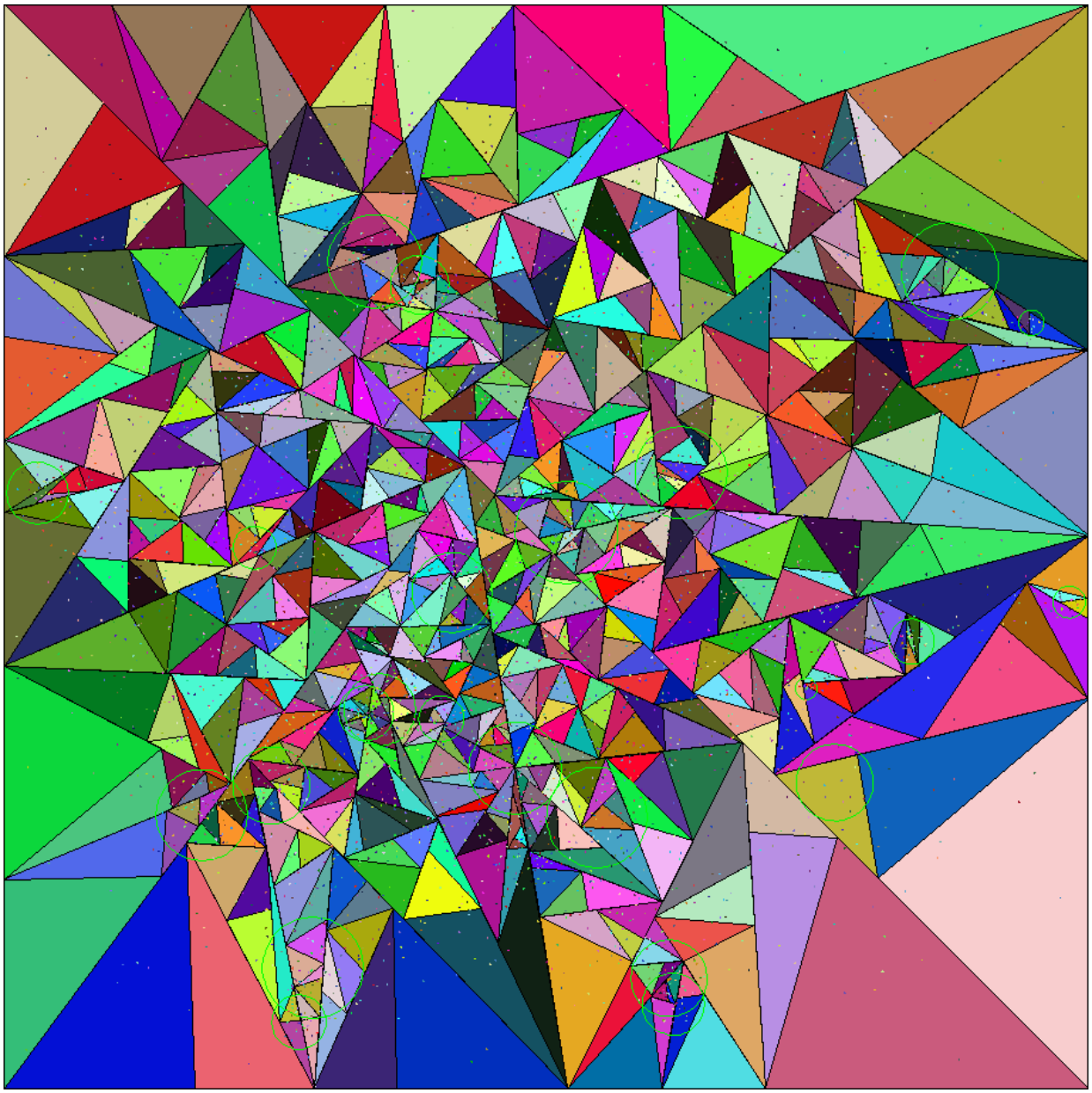


FIGURE 3.3 Exemple de décomposition de la surface de jeu en zones triangulaires

Tel que vu au chapitre précédent, plusieurs modèles à base de figures géométriques hexagonales (inspirés des réseaux de téléphonie cellulaire) ont été décrits dans la littéarture. La forme hexagonale a pour avantage de se rapprocher de la forme d'un cercle, garantissant ainsi une géométrie symétrique tout en permettant un agencement sans chevauchement (ie. il est possible de trouver un agencement tel que pour tout point P du plan, P ne sera contenu que dans une et une seule zone hexagonale). L'inconvénient majeur de la forme hexagonale est qu'il n'est pas possible de redimensionner aisément certaines régions. Pour obtenir des régions de taille variable, il faut altérer toutes les régions, sinon la propriété de tesselisation sans chevauchement n'est pas respectée.

Le modèle en zones triangulaires, inspiré des principes de tesselisation de modèles virtuels, a été retenu puisqu'il s'agit de la primitive géométrique qui offre le plus de flexibilité quant à l'agencement. En effet, non seulement il est possible de subdiviser une surface en une multitude d'agencements à base de triangle, mais en plus il est possible de modifier l'agencement *localement* (ie. pour une seule zone) sans qu'il n'y aie de répercussions sur les autres zones [1].

Chaque parcelle du jeu est associée à une zone triangulaire de forme variable. La taille de chacune des zones dépend de plusieurs facteurs (notamment du nombre de joueurs présents) qui seront discutés plus en détails dans les sections 4.2 et 3.5. Chaque zone est assignée à un noeud serveur donné. Chaque noeud responsable d'une zone donnée rapporte au serveur central sur une base régulière les modifications à l'état des joueurs et à l'état de la zone. L'intervalle de propagation, dénoté B_i ("broadcast interval"), sera discuté plus en détails à la section suivante. Lorsqu'un noeud joueur ne peut plus assurer la maintenance d'une zone donnée, le serveur central désigne un autre noeud pour en prendre charge.

La disposition en zones triangulaires est très flexible dans le but d'assurer un agencement permettant de s'adapter le plus fidèlement possible aux conditions de jeu. Toutefois, une contrainte fondamentale doit être respectée en tout temps : *l'agencement de triangles doit couvrir l'entièreté du territoire de jeu sans qu'il n'y aie aucun chevauchement.* La section 3.5 présentera les différents mécanismes possibles pour altérer l'agencement.

3.4 Évaluation de la charge

Afin d'assurer la stabilité de la sous-couche pair-à-pair et de garantir des performances optimales à tous les joueurs malgré l'utilisation d'une telle topologie architecturale, le serveur central doit continuellement superviser l'état du jeu. Pour ce faire, le serveur effectue à chaque seconde une évaluation du coût de chaque zone triangulaire et calcule la charge

1. Il pourrait même être possible de subdiviser une surface de jeu non-rectangulaire puisque n'importe quelle forme géométrique peut se découper en un ensemble de triangles.

correspondante imposée sur le noeud serveur afin de déterminer si cette dernière est adéquate. Le cas échéant, des mesures de rebalancement seront appliquées. La formule d'évaluation de coût prend en charge plusieurs paramètres qui seront décrits dans les sections qui suivent.

3.4.1 Maintenance des joueurs

Chaque joueur est domicilié dans une seule zone. Le noeud serveur de cette zone est entièrement responsable de la gestion du joueur. Cependant, il peut arriver que la zone d'intérêt du joueur recoupe une (ou plusieurs) autres zones. Dans le but d'éviter les conflits de synchronisation, la connexion multiple d'un noeud à plus d'un serveur de zone n'est pas autorisée. Il est nécessaire de mettre en oeuvre un mécanisme de relais en conditions frontières. Notre modèle propose trois classes de joueurs :

Joueur domicile : tout joueur J ayant une appartenance à une zone Z est dit *joueur domicile* de cette zone. Cette zone est la *zone primaire* du joueur J.

Joueur visiteur : tout joueur J dont la zone d'intérêt chevauche une zone Z mais qui est toutefois domicilié dans une autre zone est dit *joueur visiteur* de la zone Z. Cette zone est une *zone secondaire* du joueur J. Notons que plus d'une zone secondaire peuvent être associées à un joueur donné.

Joueur relais : tout joueur domicile J ayant une appartenance à une zone Z, dont la zone d'intérêt recoupe une ou plusieurs autres zones Z_i est dit *joueur relais* pour ces autres zones Z_i.

Le principe de relais fonctionne comme suit. Supposons un joueur J situé dans une zone Z_p ayant pour noeud serveur S_P dont la zone d'intérêt chevauche les zones Z_1 et Z_2, respectivement gérées par les noeuds S_1 et S_2. Pour tout événement dont la portée ne dépasse pas la région délimitée par Z_p, S_P assura le traitement via des échanges avec le noeud joueur J. Pour tout événement concernant Z_1, le joueur J ne communiquera avec S_1 que via son serveur S_P. Autrement dit, S_P agit comme *agent* ou relais auprès de S_1. Si le serveur S_1 désire transmettre un message concernant le joueur J, il le fera également via son *agent* S_P. Il en va de même pour S_2. Évidemment, si le joueur J continue sa traversée de sorte qu'il soit maintenant situé dans S_1, le serveur précédent S_P transfèrera le contrôle de J à S_1. Si la zone d'intérêt de J continue de chevaucher la zone Z_P, S_1 agira en tant qu'agent. Nous supposons que les serveurs de zones maintiennent des connexions aux serveurs des zones avoisinantes puisque le mécanisme de relais sera fréquemment utilisé.

La figure 3.4 illustre un exemple de situation avec des joueurs domicile, visiteurs et relais. Cette figure illustre les joueurs 1 à 5 et les zones A, B, C, D, E. Supposons que ces zones sont gérées respectivement par les noeuds serveurs S_A, S_B, S_C, S_D, S_E. Dans cette situation,

les joueurs 1, 2, 3 et 4 sont des joueurs domiciles du serveur S_A. Parmi ces derniers, le joueur 1 est également un joueur visiteur de la zone B ; toute communication avec S_B s'effectue via son *agent* S_A uniquement. Il est donc joueur relais pour la zone B puisque l'établissement d'un relais est nécessaire pour que le joueur 1 puisse intéragir avec S_B. Finalement, le joueur 5 est un joueur domicile de la zone B. Il est également un joueur visiteur des zones A, C et D et un joueur-relais pour ces mêmes zones puisque S_B doit gérer les communications avec S_A, S_C et S_D pour le joueur 5.

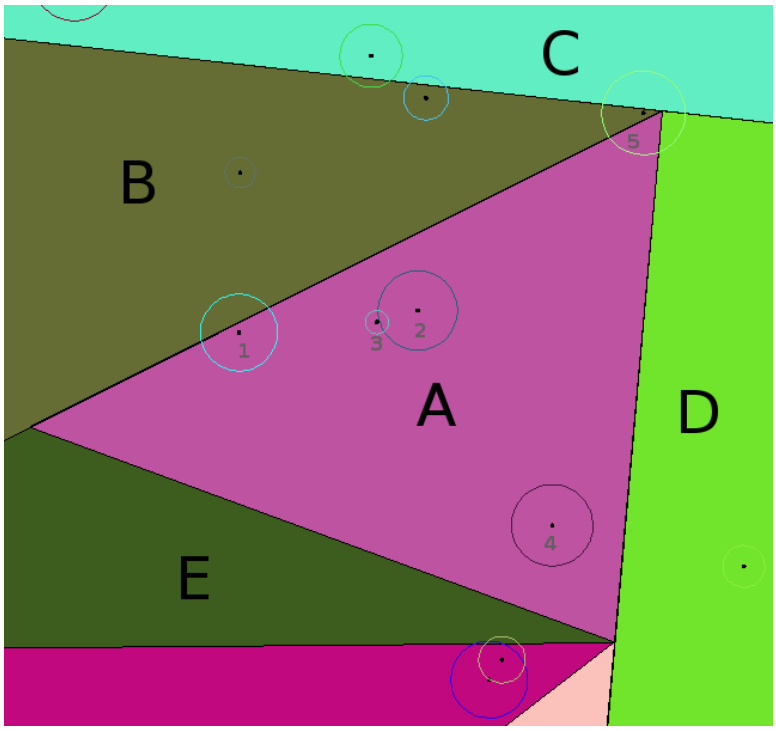


FIGURE 3.4 Exemple de localisation de joueurs au sein de zones triangulaires

Un poids est associé à la maintenance de chaque joueur par une zone donnée. Le tableau 3.4 détaille les paramètres des poids associés à la maintenance des joueurs pour une zone donnée, au temps t. Les coûts sont calculés pour toute les zones.

Le coût total pour assurer la maintenance des joueurs pour une zone donnée au temps t est donné par la formule 3.1. Dans la figure 3.4, pour la zone A nous aurions donc 4 joueurs domicile, 1 joueur visiteur et 1 joueur relais ($n_h^t = 4$, $n_v^t = 1$, $n_r^t = 1$). Pour la zone B, $n_h^t = 3$, $n_v^t = 2$, $n_r^t = 4$.

$$C_m^t = n_h^t W_h + n_v^t W_v + n_r^t W_r \tag{3.1}$$

Tableau 3.4 Paramètres des coûts de maintenance des joueurs pour une zone donnée

Paramètre	Description
n_h^t	Nombre de joueurs domicile
W_h	Poids associé à la maintenance d'un joueur domicile
n_v^t	Nombre de joueurs visiteurs
W_v	Poids associé à la maintenance d'un joueur visiteur (la communication s'effectue via son agent)
n_r^t	Nombre de joueurs relais, qui représente, pour chaque joueur domicile, le nombre de zones se chevauchant avec la zone d'intérêt du joueur.
W_r	Poids associé à la maintenance d'un relais (pour un joueur dont la zone d'intérêt regroupe une autre zone). Il peut y avoir plus d'un relais pour un même joueur.

3.4.2 Migration des joueurs

En plus du coût de maintenance, nous considérons également un coût imputé à l'entrée et à la sortie d'un joueur d'une zone donnée. La prise en charge d'un nouveau joueur au sein d'une zone donnée demande une allocation de ressources (peu importe que ce dernier soit un joueur domicile ou visiteur). Cette allocation de ressources se modélise par un coût ponctuel, le *coût de connexion*, de poids W_c. À l'inverse, il y a également présence d'un coût dû à la fin de prise en charge d'un joueur, le *coût de déconnexion*, de poids W_d. Le coût de déconnexion s'applique pour chaque joueur qui quitte une zone donnée (à titre de joueur domicile ou visiteur). Ce paramètre de modélisation sert à mettre en relief les zones où beaucoup de migrations surviennent. Ainsi, dans un jeu où il y a beaucoup de joueurs qui conservent leur position, le coût de maintenance pourrait être élevé alors que le coût de migration serait quasi-nul. À l'inverse, dans certaines zones où il y a peu de joueurs mais beaucoup de déplacements, le coût de maintenance serait faible, alors que le coût dû aux migrations serait élevé, ce qui est un facteur important à considérer pour l'évaluation de performance. Le coût de migration se calcule par unité de temps pour une zone donnée grâce à l'équation 3.2, où n_c^t représente le nombre de joueurs qui sont *entrés* dans la zone et n_d^t, le nombre de joueurs qui sont *sortis* de la zone, au temps t.

$$C_g^t = n_c^t W_c + n_d^t W_d \tag{3.2}$$

3.4.3 Superficie du territoire

Bien que les coûts les plus importants pour un noeud serveur soient les coûts reliés à la maintenance et à la migration des joueurs, il y a tout de même un coût imputé à la *gestion* du territoire de la zone au temps t (C_s^t). Ce coût modélise les ressources systèmes nécessaires (CPU, mémoire) à la gestion des différents événements et scripts survenant dans la zone. Ce coût est le produit du coût par unité de superficie (W_a) par l'aire de la zone au temps t (A_z^t). Ainsi, une zone de plus grande superficie exigera un coût de traitement supérieur par rapport à une zone de superficie plus petite. Le poids de la superficie a donc un impact dans le calcul du coût total de la zone. La formule 3.3 illustre le calcul du coût de gestion du territoire pour une unité de temps.

$$C_s^t = A_z^t W_a \tag{3.3}$$

3.4.4 Coût total et coût amorti

Les équations 3.4 et 3.5 donnent le coût total d'une zone donnée au temps t. Rappel : t représente un nombre d'unités de temps écoulées depuis le début de la simulation.

$$C_z^t = C_m^t + C_g^t + C_s^t \tag{3.4}$$

$$C_z^t = n_h^t W_h + n_v^t W_v + n_r^t W_r + n_c^t W_c + n_d^t W_d + A_z^t W_a \tag{3.5}$$

Toutefois, il peut arriver que des changements fassent varier le coût de façon drastique d'une unité de temps à l'autre. Dans un tel cas, l'algorithme de rebalancement pourrait effectuer une série d'opérations inutiles en réaction aux changements soudains. Pour palier à ce problème, nous proposons d'amortir le calcul du coût sur une période de quelques secondes. Le coût amorti, C_A^t, est fortement pondéré par le coût de l'unité de temps précédente $(t-1)$ et faiblement pondéré par l'unité de temps actuelle (t). Il s'exprime par l'équation 3.6.

$$C_A^t = \begin{cases} C_A^t = C_z^t & \text{si} \quad C_z^{t-1} = 0 \\ C_A^t = \frac{9}{10} C_z^{t-1} + \frac{1}{10} C_z^t & \text{si} \quad C_z^{t-1} > 0 \end{cases} \tag{3.6}$$

L'utilisation du coût amorti permettra d'adoucir les opérations de rebalancement qui seront décrites plus en détails à la section 3.5.

3.4.5 Ratio de charge

Le ratio de charge est le rapport entre le coût amorti de la zone (à un temps t donné) et la capacité de charge du noeud joueur qui en est responsable. Le ratio de charge, LR_t au temps t, est défini par l'équation 3.7. Il sert à déterminer si la zone est sous-chargée, normalement chargée ou surchargée.

$$LR_t = \frac{C_A^t}{C_{max}} \tag{3.7}$$

On considère que toute zone dont le ratio de charge excède 1 peut être sujette à des dégradations de performance ; si cette situation survient, il est approprié d'y remédier rapidement. On considère également que toute zone dont le ratio de charge se situe dans l'intervalle $[0, 1]$ ne devrait pas subir de dégradations de performance.

3.4.6 Latence moyenne

Tel qu'énoncé précedemment, en raison de la sous-couche pair-à-pair, il est possible que la latence entre un noeud serveur et les joueurs domicile ne soit pas optimale (exemple : si tous les joueurs viennent de l'Amérique du Nord alors que le noeud serveur est localisé en Chine). Une évaluation de la latence moyenne est périodiquement effectuée afin de s'assurer que cette dernière demeure en bas d'un certain seuil. La latence moyenne est définie comme la moyenne de l'ensemble des latences entre les joueurs et le noeud serveur. Le calcul de cette latence moyenne est formalisé par l'équation 3.8, où n est le nombre de joueurs dans la zone et L_{s-i} la latence entre le noeud serveur s de la zone et le joueur i.

$$\overline{L_s} = \sum_{i=1}^{n} \frac{L_{s-i}}{n} \tag{3.8}$$

3.4.7 Comparaison avec le modèle client-serveur classique

Afin d'évaluer l'efficacité de l'approche pair-à-pair par rapport à une approche traditionnelle client-serveur, notre modèle propose une méthode pour estimer les coût si une telle approche était utilisée. Plus spécifiquement, notre modèle propose de comparer la charge au serveur central par unité de temps en mode pair-à-pair par rapport au mode client-serveur pour vérifier quelle économie est réalisée. Les sections précédentes ont traité de l'évaluation de performance pour les noeuds serveurs de zones, c'est-à-dire quelles métriques étaient importantes pour effectuer ce calcul. La charge au niveau du serveur central n'a pas été évaluée à présent.

En mode pair-à-pair, notre modèle considère que chacun des noeuds serveurs responsables d'une zone donnée propagent une mise à jour de leur état au serveur central selon un intervalle temporel défini, B_i, tel que défini à la section 3.3. Ainsi, à chaque B_i unités de temps, chaque serveur de zone envoie les donnés pertinentes concernant l'état de sa zone afin que le serveur central puisse sauvegarder l'état du jeu dans la base de données centrale persistante. Selon notre modèle, l'état du jeu regroupe :

- le profil de chacun des joueurs, qui contient toutes les informations pertinentes pour *persister* l'état et la cohérence de chaque joueur inscrit au jeu ;
- l'état du territoire du jeu, qui contient toutes les informations pertinentes servant à assurer la persistance et la cohérence de l'ensemble de l'univers virtuel.

En raison de la subdivision géographique en zones triangulaires, l'état du jeu est séparé en un ensemble de zones. Chaque zone contient donc une parcelle des deux éléments mentionés précédemment concernant l'état global du jeu. Il est proposé que le noeud responsable de chaque zone transmette à chaque B_i unités de temps les changements reliés à l'état des joueurs domicile de la zone et au territoire de la zone. Puisque chaque joueur est géré par un et un seul serveur de zone, une couverture complète de l'état de chaque joueur sans redondance est assurée. Pour fins de simplification, on considère que le poids des données de la mise à jour sont les mêmes que ceux définis dans les sections précédentes pour l'évaluation de la charge des zones. Ainsi, à chaque intervalle de temps B_i, chaque serveur de zone transmettra un volume de données correspondant au coût de maintenance des joueurs domicile et du territoire de la zone. L'équation 3.9 illustre le coût imputé au serveur central lié à la mise à jour de l'état du jeu à chaque intervalle de propagation, pour l'ensemble des zones 1 à n de la carte du monde.

$$C_b^{p2p} = \sum_{j=1}^{n} \left(n_h^j W_h + A_z^j W_a \right) \tag{3.9}$$

De plus, en raison de l'exigence selon laquelle le serveur central doit toujours connaître la localisation de chaque joueur du jeu, on suppose qu'un mécanisme de mise à jour avertit le serveur central lorsqu'un noeud pénètre ou quitte une zone donnée. Pour fins de simplification, on suppose que les poids de connexion (W_c) et de déconnexion (W_d) discutés dans la section des coûts de migration (3.4.2) sont réutilisés. Le coût imputé au serveur central par unité de temps est simplement la somme de tous les coûts de migration (connexion et déconnexion) dans toutes les zones de la carte du jeu. L'équation 3.10 illustre les coûts imputés au serveur central en raison de la migration des joueurs, pour l'ensemble des zones 1 à n de la carte du monde.

$$C_g^{p2p} = \sum_{j=1}^{n} \left(n_c^j W_c + n_d^j W_d \right) \tag{3.10}$$

Les coûts imputés au serveur central en mode pair-à-pair par unité de temps correspondent à $\frac{1}{B_i}C_b^{p2p}$ puisque C_b^{p2p} représente les coûts imputés à chaque intervalle de propagation B_i. Les équations 3.11 et 3.12 représentent respectivement sous forme compacte et étendue le coût total imputé au serveur central par unité de temps lors du fonctionnement en mode pair-à-pair selon le modèle décrit.

$$C^{p2p} = \frac{1}{B_i}C_b^{p2p} + C_g^{p2p} \tag{3.11}$$

$$C^{p2p} = \sum_{j=1}^{n}\left(\frac{1}{B_i}\left(n_h^j W_h + A_z^j W_a\right) + n_c^j W_c + n_d^j W_d\right) \tag{3.12}$$

Le calcul du coût total imputé au serveur par unité de temps si le modèle client-serveur classique était utilisé est beaucoup plus simple. Selon ce paradigme, le serveur effectue la gestion individuelle de chaque joueur et du territoire global du jeu. Encore une fois, pour fins de simplification, on considère que les mêmes paramètres de pondération sont utilisés. Calculer le coût total se résume à calculer le coût de maintenance de chaque joueur [2] et le coût de maintenance du territoire global selon sa superficie. L'équation 3.13 détaille le calcul du coût total imputé au serveur central par unité de temps. Remarquons qu'il s'agit de la même équation que celle utilisée pour le calcul du coût de propagation C_b^{p2p} si le serveur s'exécute en mode pair-à-pair (3.9). La différence est que le coût de propagation est imputé au serveur central à chaque intervalle de propagation B_i, alors que le coût total de gestion en mode client-serveur est imputé au serveur central à chaque unité de temps t.

$$C^{cs} = \sum_{j=1}^{n}\left(n_h^j W_h + A_z^j W_a\right) \tag{3.13}$$

Bien évidemment, d'autres paramètres peuvent s'avérer pertinents pour le calcul de la charge serveur. Ces autres paramètres n'ont pas été considérés et ont été négligés pour des fins de simplification.

Connaissant le coût imposé au serveur en mode client-serveur et en mode pair-à-pair, il est possible de calculer le rapport des coûts requis entre le mode client-serveur et le mode pair-à-pair. Les équations 3.14 et 3.15 détaillent le rapport des coûts avec un serveur central fonctionnant sous le modèle pair-à-pair proposé et sous le modèle client-serveur classique.

$$R_{\mathrm{cs/p2p}} = \frac{C^{cs}}{C^{p2p}} \tag{3.14}$$

2. On considère ici que chaque joueur est un joueur *domicile* du serveur central ; les notions de joueur visiteur et joueur relais ne sont pas pertinentes dans ce contexte.

$$R_{\text{cs/p2p}} = \sum_{j=1}^{n} \frac{n_h^j W_h + A_z^j W_a}{\frac{1}{B_i}\left(n_h^j W_h + A_z^j W_a\right) + n_c^j W_c + n_d^j W_d} \tag{3.15}$$

Au prochain chapitre, les rapports des coûts seront calculés pour les différentes simulations. En fonction de l'économie réalisée, il pourra être possible de tirer des conclusions sur le nombre de joueurs qui pourraient être pris en charge par le modèle pair-à-pair.

3.5 Rebalancement dynamique

Suite à l'évaluation du ratio de charge de chacune des zones à chaque unité de temps, le module analyseur de rebalancement détermine si certaines actions doivent être entreprises. Avant d'introduire les différentes opérations et algorithmes de rebalancement, il convient de définir certains paramètres que le module analyseur du serveur central utilisera pour prendre les décisions relatives aux opérations de rebalancement. Le tableau 3.5 liste les paramètres pertinents.

Tableau 3.5 Paramètres de rebalancement

Paramètre	Description
LR_{high}	Ratio de charge à partir duquel une zone est considérée en surcharge.
LR_{low}	Ratio de charge à partir duquel une zone est considérée en sous-chargé.
L_{high}^{avg}	Latence moyenne entre le noeud serveur et l'ensemble des noeuds joueurs d'une zone à partir de laquelle on considère que la qualité de jeu pourrait être affectée.
LT_{max}^{R}	Ratio de charge maximal admissible lors de l'affectation d'un nouveau noeud serveur à une zone donnée. Autrement dit, seul un noeud serveur S tel que le ratio de charge serait en délà de ce ratio maximal si S était assigné à une zone donnée serait considéré.
n_σ	Seuil à partir duquel une opération de séparation surviendra obligatoirement en cas de surcharge de zone (au-délà de ce seuil, une réassignation ne sera pas considérée). En considérant que la distribution des capacités de charge des joueurs suit une loi normale de moyenne μ_c et d'écart-type σ_c, une opération de séparation surviendra obligatoirement si la charge de la zone excède $\mu_c + n_\sigma \sigma_c$.
n_r^{high}	Nombre maximal d'opération de rebalancement effectuées par unité de temps, pour des zones surchargées.
n_r^{low}	Nombre maximal d'opération de rebalancement effectuées par unité de temps, pour des zones sous-chargées.
n_r^{total}	Nombre total maximal d'opération de rebalancement effectuées par unité de temps. $n_r^{total} = n_r^{high} + n_r^{low}$
t_b	Lorsqu'une opération de rebalancement est effectuée sur une zone donnée, cette valeur représente l'intervalle de temps pendant lequel cette zone ne pourra subir d'autre rebalancement ("blacklist").

Lorsque le ratio de charge d'une ou plusieurs zones excèdent une certaine valeur seuil, il convient de prendre des mesures pour faire diminuer la charge. À l'inverse, lorsque le ration de charge arrive en déça d'une autre valeur seuil, il peut être avantageux d'opérer un réagencement pour mieux réassigner les ressources. Il se peut également que suite à certaines migrations de joueurs, la latence moyenne d'une ou plusieurs zones dépasse un seuil considéré "acceptable". L'ensemble des manipulations visant à établir des réajustements à la structure de la sous-couche pair-à-pair sont appelées *opérations de rebalancement*. Notre modèle définit trois opérations :

- réassignation d'une zone ;
- séparation d'une zone ;
- fusion de deux zones.

Ces opérations qui permettent de faire des réorganisations à l'échelle locale (ie. affectant seulement une ou deux zones, sans que ce ne soit nécessaire de remodeler la structure pair-à-pair au complet). Elles seront explicitées dans les sous-sections suivantes. En premier lieu, l'algorithme 3 indique le traitement à effectuer lorsqu'il est nécessaire de déterminer le noeud optimal pour une zone donnée (ie. le noeud qui serait le plus apte à devenir serveur).

3.5.1 Réassignation d'une zone

L'opération de réassignation consiste à réassigner le contrôle d'une zone par un noeud serveur donné à un autre noeud serveur. Le serveur central peut effectuer cette opération si l'une des deux conditions suivantes surviennent pour une zone donnée :

1. le ratio de charge dépasse LR_{high}, le serveur central peut alors nommer un noeud serveur remplaçant qui possède une meilleure capacité de charge ;
2. la latence moyenne entre le noeud serveur et l'ensemble des joueurs de la zone dépasse LT_{max}, le serveur central peut alors nommer un noeud serveur remplaçant dont la latence moyenne est inférieure.

L'algorithme 4 formalise le fonctionnement de l'opération de rebalancement. Tel que décrit, il est possible qu'aucun noeud adéquat ne puisse être trouvé, selon l'une des deux situations suivantes :

1. aucun noeud ne possède une capacité de charge suffisante pour prendre en charge la zone ;
2. aucun noeud ne possède une latence moyenne par rapport aux joueurs qui ne dépasse un certain seuil.

Dans un tel cas, l'opération de réassignation ne peut pas être exécutée sur la zone. Une autre opération sera probablement nécessaire, comme une séparation par exemple.

Algorithme 3 (Détermination du noeud optimal)

1. *`MeilleurNoeud` = nul*
2. **Pour chaque** *noeud de la partie*
   (a) **Si** *le noeud est déjà assigné à une zone* **ou** *est présentement déconnecté*
      i. *Passer au prochain noeud (retourner à la ligne 2)*
   (b) *Calculer le ratio de charge de la zone si ce noeud en devenait le serveur (en utilisant l'équation 3.6).*
   (c) **Si** *le ratio* $\leq LT_{max}$ *(ce noeud a une capacité de charge suffisante pour assurer la prise en charge de la zone)*
      i. *Calculer la latence moyenne entre ce noeud joueur et tous les noeuds de la zone (en utilisant l'équation 3.8).*
      ii. **Si** *la latence moyenne est moins élevée que celle du meilleur noeud trouvé*
         A. *`MeilleurNoeud` = le noeud actuel*
3. **Si** *`MeilleurNoeud` est nul*
   (a) **Fin:** *Aucun noeud ne peut prendre en charge la zone (aucun noeud n'a de capacité suffisante OU aucun noeud ne peut assurer une latence moyenne acceptable)*
4. **Sinon**
   (a) **Fin:** *Retourner `MeilleurNoeud`*

FIGURE 3.5 Détermination du noeud optimal

3.5.2 Séparation d'une zone

L'opération de séparation consiste à séparer une zone triangulaire en deux zones triangulaires distinctes. À l'échelle du monde virtuel, cette opération n'altère que la zone visée par cette opération. Le serveur central peut décider de procéder à une séparation si une zone donnée devient surchargée, et qu'il est difficile ou impossible d'assigner un noeud serveur alternatif qui soit suffisament performant pour réduire la charge de manière significative.

Dans les périodes d'accroissement du nombre de joueurs, par exemple lorsque l'état actuel du jeu passe d'une plage de faible achalandage à une plage de fort achalandage ou lors de de la mise en service de ce dernier, les opérations de séparation seront fréquentes à travers l'ensemble du territoire du jeu. Les opérations de séparation seront également nombreuses dans les régions d'intérêt afin de s'adapter au flot important de joueurs.

Normalement, l'opération de séparation éliminerait une zone donnée pour en créer deux nouvelles. Cela induirait inévitablement un transfert de l'ensemble des données de la zone et de la prise en charge des joueurs vers deux nouveaux noeuds serveurs, ce qui est une opération coûteuse. Nous proposons l'optimisation qui suit pour nous permettre de mitiger

Algorithme 4 (Algorithme de réassignation)

1. *Exécuter l'algorithme* `Déterminer le noeud optimal` *(3) pour déterminer le noeud le plus apte à devenir serveur de la zone*
2. **Si** *aucun noeud n'a été trouvé*
   (a) **Fin:** *Aucune opération de réassignation n'est possible pour cette zone*
3. **Sinon**
   (a) **Fin:** *Le noeud trouvé peut prendre le contrôle de la zone*

FIGURE 3.6 Algorithme de réassignation

en partie ce problème. Au lieu de *remplacer* la zone initiale par deux nouvelles zones, il est proposé de *redimensionner* (réduire) la zone initiale pour former une des deux nouvelles zones et, évidemment, de créer l'autre zone. De cette façon, seul la recherche d'un noeud serveur et le transfert d'une partie (autour de 50%, mais ce pourcentage peut varier) de la zone vers ce nouveau noeud serveur seront nécessaires. Puisque la zone initiale aura plutôt été redimensionnée (et non détruite), le noeud serveur original conservera son rôle, mais en délaissant une part de ses responsabilités. Ce noeud devrait normalement être apte à gérer la nouvelle zone réduite. Advenant l'éventualité où ce dernier ne serait plus en mesure d'en assurer la gestion, une autre opération de rebalancement sera invoquée par le serveur ; à savoir, un rebalancement ou une autre séparation.

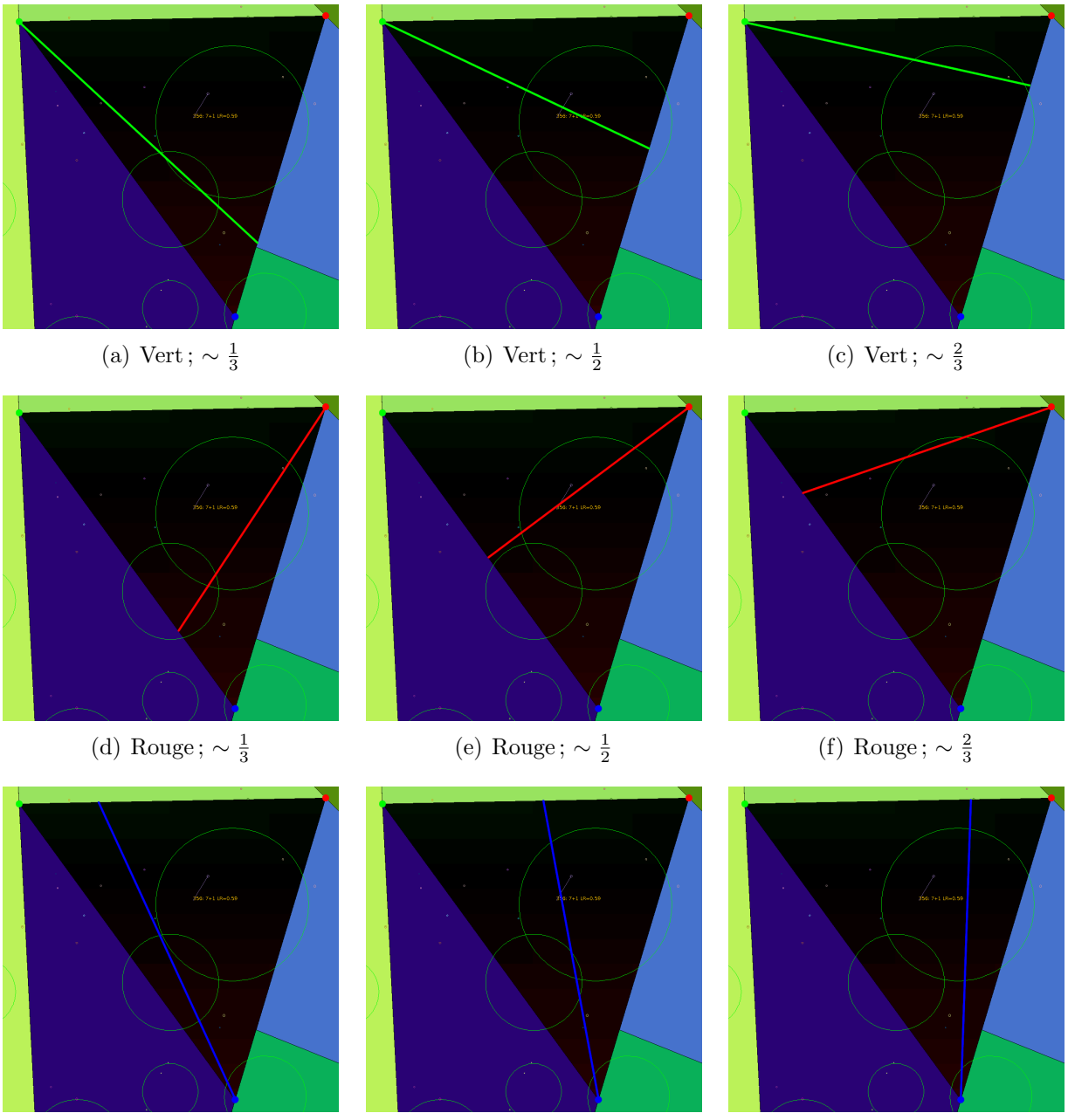
Il est essentiel de respecter la contrainte selon laquelle seules des zones triangulaires sont permises. Pour y parvenir, toute nouvelle zone formée par une opération de séparation doit être formée d'un des sommets de la zone initiale. Par géométrie, ce sommet sera commun aux deux nouveaux triangles générés. Le principe général de la division consiste à déterminer une ligne de coupure qui servira à délimiter les deux nouvelles zones. Cette ligne de coupure constituera un côté commun à ces deux triangles. Elle part de l'un des trois sommets du triangle initial, et se prolonge jusqu'à n'importe quel point situé sur le côté opposé à ce sommet. Évidemment, cela conduit à une infinité de possibilités de coupure et ce, pour les trois sommets de la zone initiale. Il s'agit ici de choisir une ligne de coupure qui garantira un rebalancement optimal. Une telle ligne de coupure visera donc à équilibrer le mieux possible le nombre de joueurs situés de part et d'autre de cette ligne. Autrement dit, une ligne de coupure optimale visera, dans la mesure du possible, à avoir un nombre égal de joueurs dans chacune des deux zones délimitées par cette ligne. Si une répartition *strictement égale* n'est pas possible (par exemple, si le nombre de joueurs dans la zone est impair ou s'il n'est pas possible de tracer une ligne de coupure qui permet une répartition égale), on choisira une répartition qui se rapproche le plus de l'égalité.

La figure 3.7 illustre des exemples de lignes de coupure pour une zone triangulaire donnée. Pour chacun des trois sommets du triangle (respectivement vert, rouge et bleu), trois exemples de lignes de coupure sont donnés. Les triangles formés de part et d'autre des lignes de coupure illustrent différents résultats qu'une opération de séparation pourrait permettre d'atteindre.

L'algorithme 5 formalise l'opération de rebalancement. À haut niveau, la procédure de séparation consiste en les trois étapes suivantes :

1. Déterminer la ligne de coupure qui génère une division optimale.
2. Redimensionner la zone initiale en deux zones selon la ligne de coupure. La première zone créée est assignée au noeud qui était responsable de la zone initiale.
3. Déterminer le meilleur noeud apte à prendre en charge la seconde zone créée et lui transférer le contrôle de la zone.

La figure 3.9, produite par le simulateur qui sera décrit au prochain chapitre, illustre un exemple réel d'une opération de séparation. Dans la figure (a), la zone bleue devient surchargée en raison du nombre de joueurs connectés. Le serveur central décide alors de séparer la zone en deux zones distinctes (bleue et verte, tel qu'illustré dans la figure (b)). La séparation choisie tend à rééquilibrer la charge entre les deux nouvelles zones ainsi crées. Le noeud qui était responsable de la zone initiale bleue demeure responsable de la nouvelle zone bleue créée.



(a) Vert ; $\sim \frac{1}{3}$ (b) Vert ; $\sim \frac{1}{2}$ (c) Vert ; $\sim \frac{2}{3}$

(d) Rouge ; $\sim \frac{1}{3}$ (e) Rouge ; $\sim \frac{1}{2}$ (f) Rouge ; $\sim \frac{2}{3}$

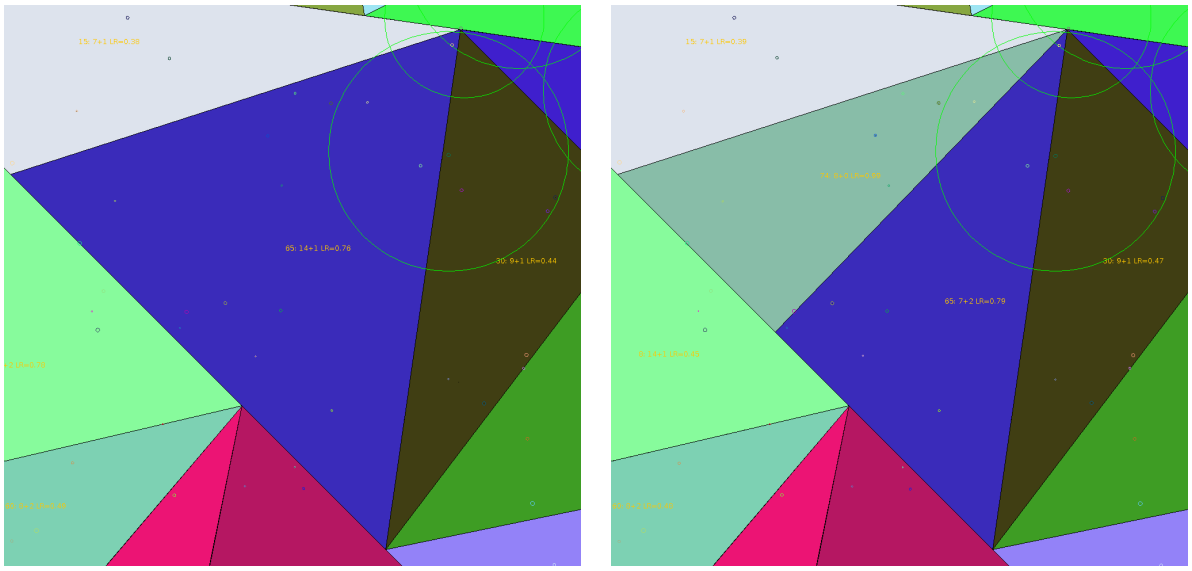
(g) Bleu ; $\sim \frac{1}{3}$ (h) Bleu ; $\sim \frac{1}{2}$ (i) Bleu ; $\sim \frac{2}{3}$

FIGURE 3.7 Exemples de lignes de coupure possibles pour une zone

Algorithme 5 (Opération de séparation d'une zone initiale `zone0`)

1. `point1` = *sommet associé à l'angle* le plus grand *du triangle*
2. `minDiff` = ∞
3. `point2` = `nul`
4. **Pour chaque** *point sur le côté opposé*
   (a) `diff` = |(*nb joueurs zone 1*) − (*nb joueurs zone 2*)|
   (b) **Si** `diff` <= `minDiff` *(le point actuel réduit ou conserve l'écart, on accepte ce point)*
      i. `minDiff` = `diff`
      ii. `point2` = *le point actuel*
   (c) **Sinon** *(le point actuel fait augmenter l'écart)*
      i. `point2` = *le point milieu entre le point actuel et* `point2`
      ii. **Sortie:** *Le point sur le côté opposé qui répartit de la meilleure façon possible les joueurs entre les deux nouvelles zones a été trouvé.*
5. *Séparer la zone en deux parties (*`zone1`, `zone2`*) selon la ligne définie par* `point1` *et* `point2`
6. *Assigner le noeud responsable de* `zone0` *en tant que noeud serveur pour la zone* `zone1`
7. *Exécuter l'algorithme* ***`Déterminer le noeud optimal`*** *(3) pour déterminer le noeud le plus apte à devenir serveur de la zone* `zone2`
8. **Si** *aucun noeud n'a été trouvé*
   (a) $LT_{max} = \infty$ *(relaxer la contrainte de charge maximale de réassignation ; le noeud sera choisi uniquement sur la base de minimiser sa latence moyenne avec les autres noeuds de la zone.)*
   (b) *Réexécuter l'algorithme* ***`Déterminer le noeud optimal`*** *(3) (une autre opération de séparation (sous-division) s'avérera fort possiblement nécessaire par la suite)*
   (c) **Si** *aucun noeud n'a été trouvé*
      i. **Fin:** *L'opération de séparation est annulée et n'est pas possible pour cette zone (ne devrait se produire qu'en raison de circonstances exceptionnelles)*
   (d) **Sinon**
      i. **Fin:** *Le noeud trouvé peut prendre le contrôle de la zone* `zone2`*, complétant ainsi l'opération de séparation*

FIGURE 3.8 Opération de séparation d'une zone initiale `zone0`



(a) Avant la séparation (zone bleue) (b) Après la séparation (zones bleue et verte)

FIGURE 3.9 Illustration du processus de séparation d'une zone

3.5.3 Fusion de deux zones

L'opération de fusion consiste à fusionner deux zones triangulaires pour n'en former qu'une seule. Cette opération n'altère que les deux zones concernées par la fusion. Le serveur central peut effectuer cette opération si les deux zones concernées ont un ratio de charge très faible. Évidemment, il faut s'assurer que le ratio de charge de la zone résultant de la fusion des deux zones concernées sera *acceptable*.

Lorsqu'une surcharge d'un noeud serveur survient, il est indispensable d'effectuer une opération (réassignation ou séparation) pour empêcher que les joueurs affectés à ce noeud serveur ne subissent des impacts de performance négatifs. Contrairement aux autres opérations, l'opération de fusion est facultative dans le sens où les noeuds serveurs ne seront pas impactés négativement par un ratio de charge trop faible. Cette opération est toutefois requise pour conserver un nombre cohérent de noeuds serveurs par rapport aux noeuds joueurs. De plus, fusionner des zones peu chargées permet de réduire le nombre de zones par lesquelles les joueurs circuleront lors de leurs déplacements, réduisant ainsi le nombre de connexions et déconnexions nécessaires. Le nombre de joueurs visiteurs relais (réf. section 3.4.1) sera fort possiblement réduit en raison de zones plus grandes puisque les noeuds joueurs seront connectés simultanément à moins de zones.

Contrairement à l'opération de séparation qui peut s'exécuter sur n'importe quelle zone triangulaire, l'opération de fusion est plus restreinte puisqu'elle ne peut opérer que sur deux

zones adjacentes qui, une fois fusionnées, formeraient une autre zone triangulaire. De plus, il est requis que la charge des deux zones candidates à la fusion n'excède pas une valeur seuil donnée. L'algorithme 6 décrit de façon détaillée le processus de fusion.

Algorithme 6 (Opération de fusion de la zone `zone0`)

1. `RatioMinimal` $= \infty$
2. `ZoneFusion` $=$ `nul`
3. **Pour chaque** *zone adjacente* `zoneA` *à la zone* `zone0`
   (a) **Si** $C_A^t < LR_{low}$ *(le ratio de charge de* `zoneA` *est inférieur au seuil)*
      i. **Si** *le résultat de la fusion n'est pas de forme triangulaire*
         A. *Passer à la prochaine zone (fusion impossible)*
      ii. **Si** $C_A^t <$ `RatioMinimal` *(zone avec un ratio de charge plus faible que le minimum trouvé)*
         A. `RatioMinimal` $= C_A^t$
         B. `ZoneFusion` $=$ `zoneA` *(la zone actuelle)*
4. **Si** `ZoneFusion` *n'est pas nulle*
   (a) *Effectuer la fusion entre* `zone0` *et* `ZoneFusion`
   (b) *Assigner le contrôle de la nouvelle zone créée au noeud responsable de* `zone0`
5. **Sinon** *(*`ZoneFusion` *demeure nulle)*
   (a) **Fin:** *Il n'est pas possible de fusionner la zone* `zone0` *avec aucune autre zone.*

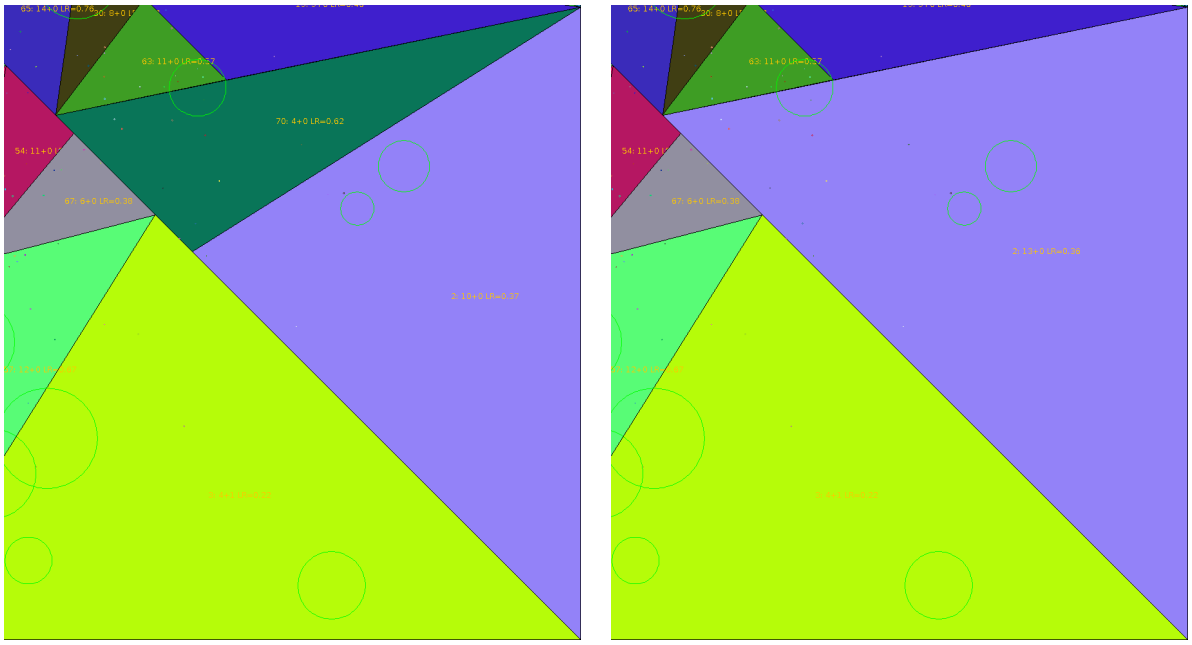
FIGURE 3.10 Opération de fusion de la zone `zone0`

La figure 3.11, produite par le simulateur, illustre un exemple réel d'opération de fusion. Dans la figure (a), les zones mauve et verte sont sous-chargées puisque peu de joueurs ne s'y trouvent. Le serveur central décide alors de fusionner les deux zones pour n'en former qu'une seule (mauve, tel qu'illustré dans la figure (b)). Le noeud qui était responsable de la zone mauve initiale demeure responsable de la zone mauve résultant de la fusion.

3.5.4 Génération d'opérations optimales

Les sections précédentes ont décrit de façon détaillée chacune des trois opérations de rebalancement possibles. L'algorithme permettant de générer la combinaison de paramètres optimaux pour chaque opération, pour une zone donnée, a été décrit. Ces algorithmes sont les suivants :

– pour l'opération de réassignation, l'algorithme permettant d'obtenir le meilleur noeud candidat à la réassignation ;



(a) Avant la fusion (zones mauve et verte) (b) Après la fusion (zone mauve)

FIGURE 3.11 Illustration du processus de fusion de deux zones

– pour l'opération de séparation, l'algorithme permettant d'obtenir la meilleure *ligne de coupure* et le meilleur noeud candidat pour la seconde zone créée ;
– pour l'opération de fusion, l'algorithme permettant d'obtenir la zone adjacente la plus propice à la fusion.

Ces algorithmes prennent en entrée une zone donnée du monde virtuel. Conséquemment, il est présupposé qu'un autre mécanisme préalable (algorithme) sélectionne une zone adéquate à l'exécution de l'algorithme en question. Par exemple, pour l'exécution de l'algorithme de fusion, on suppose qu'une zone peu chargée est passée en entrée. Bien sûr, il est possible que l'exécution d'un algorithme donné ne soit pas possible. Par exemple, il se pourrait que l'algorithme de fusion ne trouve aucune zone adjacente avec un ratio de charge suffisament faible pour assurer une fusion adéquate. Si une telle situation survient, l'exécution de l'opération de rebalancement sélectionnée sur la zone en question est simplement annulée. Si possible, une opération alternative peut alors être sélectionnée, ou bien le traitement de la zone est annulé.

Cette section comblera les éléments n'ayant pas encore été explicités en décrivant, dans un premier temps, l'algorithme permettant de sélectionner l'opération optimale pour une zone donnée (réassignation, séparation ou fusion). Dans un second temps, l'algorithme permettant de déterminer quelles zones doivent être traitées à chaque itération temporelle sera explicité.

Génération d'une action optimale

L'algorithme de génération d'une action optimale permet de déterminer, pour une zone donnée et à un instant donné t, si une opération de rebalancement est nécessaire. Si tel est le cas, l'algorithme permet de déterminer quelle opération est la plus optimale selon le contexte actuel. Cet algorithme fait appel aux trois algorithmes définis précédemment pour générer les paramètres optimaux pour l'opération de rebalancement choisie.

L'algorithme vérifie initialement si la zone en entrée est surchargée. Si tel est le cas, l'algorithme vérifie si une opération de réassignation serait propice où s'il est nécessaire de procéder à une séparation. Tel que décrit précédemment, nous supposons que la distribution de la capacité de charge des joueurs C_{max} est connue à l'avance, demeure statique (section 3.1) et suit une loi normale de moyenne μ_c et d'écart-type σ_c (sous-section 3.2.3). Le paramètre n_σ spécifie à partir de combien d'écart-types (en haut de la moyenne) une valeur de coût total pour une zone donnée est considérée comme étant suffisament élevée pour nécessiter obligatoirement une séparation. L'équation 3.16 formalise cette condition. Dans le cas contraire, l'algorithme tentera de vérifier en premier si une réassignation est possible, et procèdera par la suite à une séparation si une réassignation s'avère impossible.

$$C_A^t > \mu_c + n_\sigma \sigma_c \tag{3.16}$$

Si la zone n'est pas surchargée, l'algorithme vérifie ensuite si la latence moyenne de la zone $\overline{L_s}$ est trop élevée, c'est-à-dire si la valeur de cette dernière se trouve au-délà du seuil défini par le paramètre L_{high}^{avg}. Si tel est le cas, l'algorithme générera une action de réassignation pour trouver le noeud remplaçant apte à prendre en charge la zone qui minimise la latence moyenne. Dans l'éventualité où aucun noeud remplaçant ne puisse être trouvé (si aucun noeud remplaçant possédant une capacité de charge suffisante ne permet de minimiser la latence moyenne), aucune action n'est générée (action "nulle"). Enfin, si la zone n'est pas surchargée et la latence moyenne de la zone est en-délà du seuil, l'algorithme vérifie si la zone est en sous-charge. La zone sera considérée en sous-charge si son ratio de charge au temps t (LR_t) est inférieur au seuil de ratio de charge faible (LR_{low}). Si tel est le cas, l'algorithme générera une action de fusion, si possible. L'algorithme 7 formalise la procédure de génération d'une action optimale pour une zone donnée `zone0` au temps `t`.

Génération des actions optimales à un temps donné

L'algorithme de génération d'une action optimale décrit à la sous-section 3.5.4 agit sur une seule zone. Cela présuppose qu'un mécanisme préalable ait identifié les zones potentiellement candidates au rebalancement. Il s'agit de l'une des tâches principales du serveur central, qui

Algorithme 7 (Génération d'une action optimale pour une zone donnée `zone0`)

1. `action` = `nul`
2. **Si** $LR_t > LR_{high}$ *(ratio de charge de* `zone0` *élevé)*
   1
   (a) **Si** $C_A^t > \mu_c + n_\sigma \sigma_c$ *(équation 3.16)*
      i. `action` = *Exécuter l'algorithme Générer séparation optimale*
   (b) **Sinon**
      i. `action` = *Exécuter l'algorithme Générer réassignation optimale*
      ii. **Si** *action est nulle (réassignation impossible))*
         A. `action` = *Exécuter l'algorithme Générer séparation optimale*
3. **Sinon** *:* **si** $\overline{L_s} > L_{high}^{avg}$ *(latence moyenne de* `zone0` *trop élevée)*
   (a) `action` = *Exécuter l'algorithme Générer réassignation optimale*
4. **Sinon** *:* **si** $LR_t < LR_{low}$ *(ratio de charge de* `zone0` *faible)*
   (a) `action` = *Exécuter l'algorithme Générer fusion optimale*
5. **Fin: Retourner** `action` *(l'action optimale générée pour la zone* `zone0`*; il se peut que l'action soit nulle si aucune action n'est requise ou n'a été trouvée pour la zone concernée)*

FIGURE 3.12 Génération d'une action optimale pour une zone donnée `zone0`

est exécutée à chaque unité de temps t. Suite à l'évaluation de la performance au temps t de chacune des zones actuelles de l'univers du jeu, le serveur doit décider quelles opérations de rebalancement sont nécessaires. Il faut noter que les opérations de rebalancement sont coûteuses en terme de ressources. Deux mesures sont prises pour mitiger ce problème :

- Dans le but d'empêcher un nombre trop important de changements simultanés, notre modèle limite le nombre d'opérations de rebalancement effectuées par unité de temps.
- Dans le but d'empêcher que des zones ayant récemment subi une opération de rebalancement ne soient rebalancées à nouveau advenant de multiples variations au niveau de la charge (mouvement oscillatoire provoquant un gaspillage important de ressources), notre modèle propose un intervalle de temps "tampon" t_b (ou temps de liste noire). Le principe est le suivant : toute zone ayant subi une opération de rebalancement ne peut subir d'autre opération avant l'expiration du délai t_b. Autrement dit, si une zone donnée est rebalancée, le seveur ne pourra ordonner d'autre rebalancement pour la ou les zone(s) concernée(s) qu'après que t_b unité de temps ne se soient écoulées.

Cette section présente l'algorithme permettant de générer la liste des actions optimales à effectuer à un temps t donné. L'algorithme part initialement avec l'ensemble des zones qui

ne peuvent être rebalancées puisqu'elles ont déjà été rebalancées dans les t_b dernières unités de temps ("liste noire"). L'algorithme trie ensuite les zones par ordre croissant de ratio de charge (LR_t), de façon à pouvoir traiter prioritairement les zones les plus chargées et les moins chargées. L'algorithme se décompose en trois étapes principales :

1. Générer séquentiellement les rebalancements pour un maximum de n_r^{high} zones, en partant de la zone la **plus** chargée.
2. Générer séquentiellement les rebalancements pour un maximum de n_r^{low} zones, en partant de la zone la **moins** chargée.
3. Si le nombre total maximal de zones pouvant être rebalancées par unité de temps est inférieur à n_r^{total}, (rappel : $n_r^{total} = n_r^{high} + n_r^{low}$), continuer à générer les rebalancements pour les zones les plus chargées jusqu'à l'atteinte de n_r^{total} zones traitées.

Lors de l'exécution des étapes précédentes, les zones sur la "liste noire" et les zones pour lesquelles un rebalancement n'est pas possible sont simplement ignorées et l'algorithme passe à la zone suivante. Les zones ignorées ne comptent pas dans le décompte des zones traitées. L'algorithme 8 formalise la procédure de génération des actions optimales à un instant t donné. Cet algorithme permet de traiter prioritairement à chaque unité de temps les zones dont un rebalancement imminent s'avère nécessaire, sans toutefois générer un nombre trop élevé de changements par unité de temps.

Une limitation de cet algorithme dans son état actuel est qu'il ne vérifie pas systématiquement les zones de charge "acceptable" (ni en surcharge, ni en sous-charge) dont la latence moyenne deviendrait trop élevée. La présélection des zones à rebalancer s'effectue uniquement selon le critère de charge (charge élevée, charge faible puis charge élevée). Cependant, dans l'une des trois phases, lorsque l'algorithme fait appel au sous-algorithme *Générer action optimale* pour une zone donnée, ce sous-algorithme vérifie la latence moyenne et propose un rebalancement si cette dernière s'avère trop élevée. Qui plus est, tel que vu dans la section 3.5.1, toute assignation ou réassignation d'une zone à un noeud serveur donné tentera de sélectionner le noeud adéquat qui minimise la latence moyenne. Par conséquent, la vérification de la latence moyenne est tout de même effectuée sous certaines conditions, mais pas systématiquement. Cela constitue une limitation du modèle.

Algorithme 8 (Génération des actions optimales pour un instant `t` donné)

1. `actions` $= \emptyset$ *(liste d'actions optimales générées)*
2. `BlacklistZones` = *ensemble des zones qui ne peuvent être rebalancées (qui ont subi un rebalancement dans les* t_b *dernières secondes)*
3. `ZonesTriees` = *Trier les zones par ordre croissant de ratio de charge* (LR_t)
4. **Pour chaque** *zone surchargée de l'ensemble* `ZonesTriees`*, en partant de la zone la plus chargée*
   (a) **Si** *la zone* $\notin$ `BlacklistZones` *(zone n'ayant pas été rebalancée dans les* t_b *dernières secondes)*
      i. `action` = *Exécuter l'algorithme Générer action optimale*
      ii. *Ajouter* `action` *à l'ensemble des actions optimales* `actions`
   (b) **Si** n_{high} *zones surchargées ont été traitées*
      i. **Sortie:** *Arrêter le traitement des zones surchargées (passer à la prochaine étape)*
5. **Pour chaque** *zone sous-chargée de l'ensemble* `ZonesTriees`*, en partant de la zone la moins chargée*
   (a) **Si** *la zone* $\notin$ `BlacklistZones` *(zone n'ayant pas été rebalancée dans les* t_b *dernières secondes)*
      i. `action` = *Exécuter l'algorithme Générer action optimale*
      ii. *Ajouter* `action` *à l'ensemble des actions optimales* `actions`
   (b) **Si** n_{low} *zones surchargées ont été traitées*
      i. **Sortie:** *Arrêter le traitement des zones sous-chargées (passer à la prochaine étape)*
6. **Si** *le nombre total de zones traitées* $< n_r^{total}$
   (a) **Pour chaque** *zone surchargée non traitée de l'ensemble* `ZonesTriees`*, en partant de la zone la plus chargée*
      i. **Si** *la zone* $\notin$ `BlacklistZones` *(zone n'ayant pas été rebalancée dans les* t_b *dernières secondes)*
         A. `action` = *Exécuter l'algorithme Générer action optimale*
         B. *Ajouter* `action` *à l'ensemble des actions optimales* `actions`
      ii. **Si** *le nombre total de zones traitées* $= n_r^{total}$
         A. **Sortie:** *Arrêter le traitement*
7. **Fin: Retourner** `actions` *(liste d'actions optimales générées)*

FIGURE 3.13 Génération des actions optimales pour un instant `t` donné

Chapitre 4

IMPLÉMENTATION ET ÉVALUATION

Afin de démontrer le fonctionnement adéquat et l'efficacité du modèle proposé au chapitre 3, des simulations de jeux massivement multijoueurs en ligne seront réalisées. Ces simulations introduiront des variations au niveau de différents paramètres afin d'évaluer dans quels contextes l'architecture proposée répond ou ne répond pas aux besoins.

Tel que mentionné dans les chapitres précédents, le domaine des jeux massivement multijoueurs en ligne est un domaine d'études très récent. Pour cette raison, nous n'avons pas été en mesure d'identifier d'outil adéquat pour réaliser les simulations. Certains outils existent pour certaines tâches données telles que la simulation d'environnements pair-à-pair ou de clusters. Aucun n'est cependant orienté spécifiquement vers le domaine des jeux massivement multijoueurs. De plus, puisque nous proposons une architecture nouvelle, il a été décidé de construire une plate-forme de simulation spécifique à cette tâche.

La plate-forme de simulation réalisée sera présentée dans le présent chapitre. Par la suite, les différents paramètres relatifs aux simulations réalisées seront présentés. Suite à l'exécution des simulations prévues, les données pertinentes seront extraites et analysées afin d'évaluer le fonctionnement du modèle. Les résultats seront finalement présentées. Parmi les résultats qui seront présentés, mentionnons notamment une comparaison entre le modèle pair-à-pair proposé et une architecture client-serveur classique (selon les mêmes paramètres), l'évaluation de la consommation de ressources auprès des noeuds serveurs et différentes statistiques ayant trait à l'affectation des joueurs aux zones.

4.1 Simulateur de jeux

Une plate-forme de simulation complète a été réalisée. Cette application est construite selon une architecture modulaire. Initialement, certains objectifs (spécifications informelles) ont été déterminés avant de réaliser la conception. La section 4.1.1 décrit les principaux objectifs sous-jacents à cette conception.

4.1.1 Objectifs et principes

Interface graphique

Il est souhaité que l'application possède une interface graphique complète utilisable par l'utilisateur, et non seulement un mode console. De plus, puisqu'une grande quantité de joueurs sont simulés à travers un univers géographique très large, il est souhaitable que l'application puisse afficher de manière visuelle l'état du monde virtuel à tout instant `t`. Le modèle de rebalancement de charge proposé fonctionne selon une répartition géographique ; il est donc souhaitable de visualiser à tout instant l'état des rebalancements. L'interface graphique doit permettre de visualiser l'état de la simulation à différents facteurs d'agrandissement pour offrir une vue globale et une vue locale sur certaines parties du territoire. Il est également possible de se "déplacer" au sein de la vue.

Grande quantité de joueurs

De par la nature des jeux massivement multijoueurs en ligne, les simulations réalisées prennent en compte plusieurs milliers de joueurs au sein d'une même partie. Les simulations sont réalisées sur un ordinateur de performance normale selon les standards actuels du marché. Le simulateur proposé doit donc être en mesure de prendre en compte une très grande quantité de joueurs simultanément et d'effectuer les calculs dans un temps raisonnable. Une limite minimale de 15000 joueurs s'avère adéquate pour les simulations. Si le simulateur permet de prendre en compte davantage de joueurs, il sera possible d'effectuer des simulations plus poussées.

Simulations de longue durée

Une autre caractéristique intrinsèque aux jeux massivement multijoueurs en ligne est qu'ils s'exécutent en permanence (persistence). L'état du jeu est en perpétuelle évolution. Les joueurs sont libres de quitter et de venir dans le jeu à tout moment, mais le jeu continue toujours à s'exécuter. Par conséquent, il est nécessaire d'exécuter le simulateur durant un temps appréciable afin d'avoir une vue d'ensemble du déroulement. Qui plus est, puisque notre modèle propose un rebalancement dynamique, il est important d'effectuer les simulations sur une période de temps suffisante pour valider le fonctionnement de l'algorithme de rebalancement au fil du temps. L'idée n'est pas ici de mesurer "ponctuellement" la charge imposée, mais plutôt de mesurer si la charge imposée demeure en-délà d'un certain seuil pendant une période de temps donnée. Ces jeux s'exécutent normalement sur une période de plusieurs années (la durée de vie active du produit). Évidemment, des interruptions sont parfois prévues pour procéder à des mises à jour, mais ces interruptions demeurent très

infréquentes. Dans le cadre de ce travail, il n'est pas réaliste d'envisager une simulation qui dure aussi longtemps ; l'analyse portera donc sur une "tranche" de l'exécution du jeu. Puisque la collecte de toutes les traces de simulation consomme beaucoup d'espace disque et de temps de calcul, il a été convenu de limiter la durée d'une simulation à un temps en-jeu de $3,5$ jours (302400 secondes). Cela constitue un bon compromis entre l'utilisation des ressources et la diversité des données collectées (ie. augmenter la durée de simulation ne générerait pas d'intérêt supplémentaire au niveau des données). Tel qu'il sera discuté plus tard, il a malheureusement été nécessaire de raccourcir certaines simulations dont les résultats étaient nettement mois bons puisque cela consommait trop de temps de calcul et d'espace disque et aurait pu excéder la capacité maximale pouvant être traitée par le simulateur.

Sauvegarde complète et relecture

Puisque les simulations prennent beaucoup de temps à s'exécuter, il est requis que le simulateur puisse sauvegarder complètement une simulation réalisée afin d'être capable de la recharger et de la revisionner entièrement. La sauvegarde regroupe l'ensemble des éléments suivants :

- l'ensemble des déplacements des joueurs selon le patron de mobilité choisi ;
- le calcul de coûts pour l'ensemble des zones, à chaque instant t, prenant en compte les changements de zones dus aux opérations de rebalancement ;
- l'ensemble des opérations de rebalancement effectuées ;
- l'ensemble des paramètres propres à la simulation courante

L'ensemble de ces éléments constitue une sauvegarde des données "brutes" de la simulation. Le mécanisme de sauvegarde permet donc de sauvegarder (sérialiser) l'ensemble des données générées par la simulation à travers toute sa durée afin d'être capable de la recharger par la suite et de restituer l'image mémoire du simulateur. Le processus de sauvegarde consomme une très grande quantité de ressources. Il va sans dire qu'une relecture peut se faire à une vitesse supérieure à la vitesse à laquelle le processus de simulation en tant que tel se déroule puisque la plupart des résultats intermédiaires ont déjà été calculés, économisant beaucoup de temps de traitement. Un autre avantage de disposer des données brutes de simulation est que cela laisse la porte ouverte à effectuer des analyses complémentaires pour extraire davantage de résultats.

Le mécanisme de relecture fonctionne à la manière d'un lecteur vidéo. Compte tenu de la nature du projet, il apparaît important de pouvoir visualiser l'état à n'importe quel instant t. Le mécanisme de "lecture" permet d'avancer et de reculer dans le temps à une vitesse donnée (variable), de mettre le déroulement en pause ou de "sauter" à un temps donné. Ce faisant, il est possible de visualiser le déroulement tout au long de la simulation, mais également de

figer le temps pour se concentrer sur l'état à un instant donné.

Il est toutefois important de noter que bien que la vue graphique soit utile pour la compréhension de l'exécution de la simulation, cette vue demeure qualitative. Les analyses et résultats obtenus seront plutôt tirés à partir des jeux de données produits par l'opération d'extraction.

4.1.2 Architecture à haut niveau

L'architecture logicielle est présentée à la figure 4.1. La conception du simulateur est réalisée de façon modulaire afin que chaque module soit facilement modifiable ou remplaçable. Une bonne pratique en génie logiciel est de minimiser les interdépendances entre les modules afin de promouvoir un couplage faible. Cet objectif a été visé au départ, mais malheureusement, il a été difficile de maintenir cette pratique au long du développement, de sorte que plusieurs modules dépendent d'autres modules.

Le simulateur a été réalisé en utilisant le langage de programmation Java 6. L'interface graphique est réalisée en utilisant la bibliothèque Java Swing qui est simple d'utilisation et permet une interface utilisateur multi-plateformes. Le simulateur s'exécute sous toute plateforme possédant une machine virtuelle Java. Il a été testé sous Microsoft Windows (XP, Vista et 7) et Linux (Ubuntu et Fedora). L'environnement de développement est Eclipse 3.5. La librairie `JDom` est utilisé pour les manipulations de fichiers XML. Le format de sortie des données de simulation est CSV.

Le simulateur possède une classe principale appelée `Application` qui expose le patron de conception *singleton*, de façon à permettre l'accès à partir de n'importe quel module. Cette classe contient des références vers l'ensemble des principaux sous-modules :

- Carte du monde (`WorldMap`)
- Bassin de joueurs (`PlayerPool`)
- Vue (interface graphique) (`View`)
- Simulateur (`Simulator`)

Chacun des modules sera décrit de façon plus détaillée dans les sections suivantes.

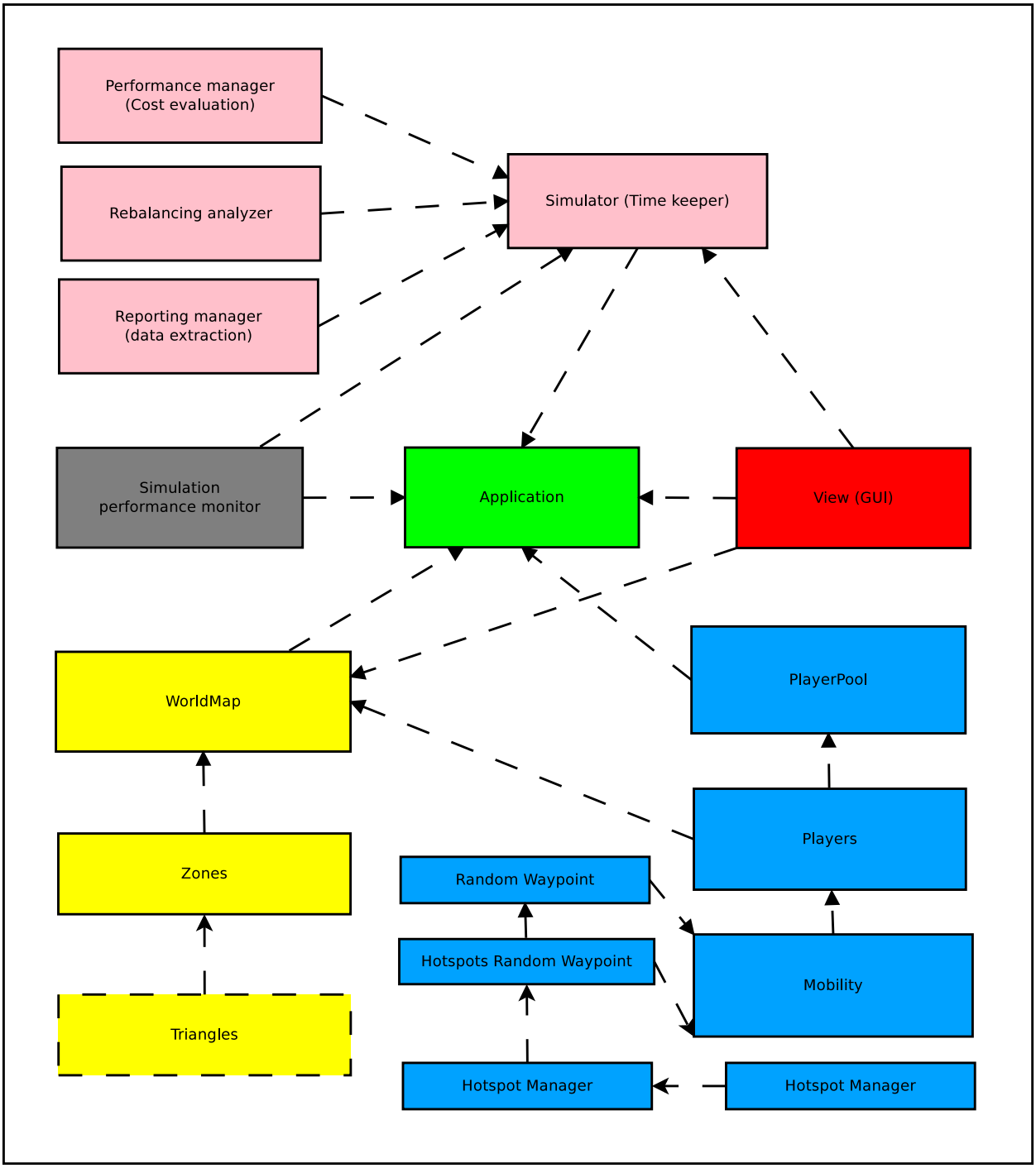


FIGURE 4.1 Architecture à haut niveau de la plate-forme de simulation

4.1.3 Joueurs

Le concept de joueur est modélisé par une classe `Joueur`. À la base, chaque joueur possède une position initiale, une zone d'intérêt, un identificateur unique (`id`) et une capacité de charge maximale (utilisée si ce noeud est appelé à agir en tant que serveur). Chaque joueur est également caractérisé par un modèle de mobilité qui définit le comportement de mouvement du joueur. Au temps 0, on connaît la position du joueur ; cependant, la position du joueur à un temps t est fonction de son modèle de mobilité. Bien que le simulateur permette des modèles de mobilité différents pour chaque joueur, nous imposons que le même modèle soit utilisé pour tous les joueurs pour fins de simplification.

Au sein de notre simulateur, trois modèles de mobilité sont implémentés :

- `NullMobilityModel`, qui représente un modèle trivial où aucun mouvement ne survient (ie. les joueurs demeurent à leur position initiale et ne bougent pas) ;
- `RandomWaypointMobilityModel`, qui représente le modèle *Random Waypoint* décrit au chapitre 3 ;
- `HotspotRandomWaypointMobilityModel`, qui représente le modèle *Hotspot Random Waypoint* décrit au chapitre 3.

Les modèles `RandomWaypointMobilityModel` et `HotspotRandomWaypointMobilityModel` contiennent une liste d'actions (déplacement et pause ou déconnexion) générées aléatoirement. Afin de récupérer l'état d'un joueur au temps t (connectivité, position), cette liste est parcourue pour trouver la bonne action en cours au temps t. Mentionnons que les actions sont générées "à-la-volée", c'est-à-dire que si l'état est demandé pour un temps t donné, l'ensemble des actions nécessaires pour se rendre au temps t demandé seront générées.

Au niveau de l'implémentation, le modèle avec régions d'intérêt (*Hotspots*) reprend (hérite) du modèle *Random Waypoint* en biaisant la procédure de sélection des points de destination pour prendre en charge les régions d'intérêt. La gestion des régions d'intérêt est assurée par la classe `HotspotManager`. Ce dernier gère le cycle de vie des régions d'intérêt (temps de création, temps de destruction, taille, poids, position) au moyen de la classe `HotspotGenerator` selon les paramètres de simulation chosis.

Un point important à noter est que les joueurs ne sont pas liés à la carte du monde choisie. Ils sont plutôt contenus dans une banque de joueur (`PlayerPool`). La banque de joueurs est indépendante de la carte du monde, ce qui signifie qu'une banque de joueurs différente peut être choisie en combinaison avec une carte du monde différente. Le seul requis est que la banque de joueurs comporte un nombre de joueurs suffisants pour la carte du monde choisie. L'idée derrière ce principe est que l'on voudra éventuellement réutiliser les mêmes "profils" de joueurs incluant la capacité de charge maximale simulée au sein de différentes simulations pour une meilleure reproductibilité. La banque de joueurs se sauvegarde et se charge à partir

d'un fichier XML.

4.1.4 Carte du monde

La carte du monde (`WorldMap`) est l'entité qui regroupe l'ensemble des joueurs (via une référence à la banque de joueurs) ainsi que l'ensemble des zones. La carte du monde regroupe également des méthodes pour séparer ou fusionner des zones. Il est important de noter que ces méthodes sont primitives dans le sens où le changement est directement appliqué à la carte. Il est donc déconseillé d'invoquer directement ces méthodes puisque cela court-circuite le processus de fonctionnement du simulateur qui décide lui-même quelles opérations sont pertinentes suite à l'évaluation de performance. Ce volet sera discuté plus en détails à la section 4.1.5.

La carte du monde se sauve et se charge à partir d'un fichier XML. Le flot de travail suppose de charger le fichier de carte du monde voulu avant de débuter une simulation puisque le fait de charger un tel fichier retire les données de la simulation en cours de la mémoire. Le fichier de carte du monde contient l'état des zones de jeu au moment de la sauvegarde. À toutes fins pratiques, ce fichier ne devrait contenir que les zones initiales au début de la simulation, normalement deux zones couvrant la superficie totale de la carte du monde, puisqu'initialement il n'y a aucun joueur présent.

4.1.5 Simulateur

Le module `Simulateur` à proprement parler constitue le coeur de la plate-forme de simulation. C'est le composant qui est responsable de la gestion de tout ce qui est dynamique (variation en fonction du temps) au sein du logiciel. À la base, le module `Simulateur` gère le temps actuel de la simulation (t) ainsi que la vitesse de déroulement de la simulation. Au chapitre 3, le temps était exprimé en "unités de temps" afin de ne pas lier le concept de temps à une valeur absolue quelconque. Dans ce chapitre, le temps t s'exprime en secondes, mais n'est pas nécessairement une valeur discrète (ie. les fractions de seconde sont permises). Le temps actuel est borné par une valeur minimale (habituellement 0 secondes) alors que le temps maximal est borné par la durée à laquelle la simulation actuelle est parvenue. Il est possible de "sauter dans le temps" à n'importe quel instant t situé entre la valeur minimale et maximale.

La vitesse de simulation constitue la vitesse de visée à laquelle le processus de simulation, ou de relecture si la simulation pour cette période de temps a déjà été effectuée, s'exécutera. Une valeur de 1.0 indique que la simulation s'exécutera en temps réel : une seconde de simulation sera exécutée approximativement en une seconde de temps réel. Une valeur supérieure

à 1.0 indique que le temps de simulation s'écoulera plus vite que le temps réel, alors qu'une valeur inférieure à 1.0 indique que le temps de simulation s'écoulera plus lentement que le temps réel. Le simulateur tentera d'aller aussi rapidement que la vitesse demandée. Si cela s'avère impossible, le simulateur ira aussi vite que possible.

La mise à jour de la vue graphique s'effectue continuellement au fil de la simulation si la vitesse de simulation demandée peut être honorée. Si la vitesse demandée ne peut être honorée, la mise à jour de la vue est suspendue afin de consacrer le plus de ressources possibles à la simulation. Cela s'avère utile pour les simulations de longue durée où l'opérateur n'a pas nécessairement besoin d'assister sur toute la durée. Il convient également de mentionner que dans le cas d'une relecture de simulation déjà effectuée, la vitesse possible est beaucoup plus élevée puisque la majeure partie des calculs sont éliminés.

En plus d'être responsable de la gestion du temps, le simulateur permet l'accès aux sous-modules pertinents à la simulation : les modules évaluation de performance, analyseur de rebalancement et générateur de rapports.

Évaluation de performance

Le module évaluation des performance (classe `CostManager` qui est contenue dans le paquetage `simulation.performance`) est responsable d'analyser la performance à l'instant t actuel. L'analyse de performance s'effectue sur l'ensemble des zones actuelles de la carte du monde. Pour chacune des zones, une structure de coût (`Cost`) est définie, de manière analogue à une structure de gain dans les réseaux de téléphonie cellulaire. Le calcul des coûts utilise les formules définies à la section 4.2. Plus spécifiquement, les différentes valeurs pertinentes sont collectées à partir de l'état actuel de mobilité des joueurs et des zones. Ces valeurs sont ensuite multipliées par leur facteur de pondération respectif. Les différents facteurs sont encapsulés au sein d'une classe `CostWeights` et constituent des paramètres de simulations qu'il est possible de modifier. Le coût total et le coût total amorti sont finalement calculés. Tel que vu précédemment, le ratio de charge s'obtient simplement en effectuant le rapport entre le coût total amorti et la capacité de charge du joueur hôte. Une fois le ratio de charge calculé pour l'ensemble des zones, le module analyseur de rebalancement, qui sera décrit à la sous-section 4.1.5, peut identifier les zones nécessitant une opération de rebalancement et générer les opérations pertinentes.

Il convient de noter que le module évaluation de performance ne conserve pas uniquement les coûts des zones pour l'instant t actuel, mais bien l'ensemble des coûts pour toutes les secondes écoulées depuis le début de la simulation (temps 0), pour chacune des zones présentes à cet instant. Retenir l'ensemble de ces informations nécessite beaucoup de mémoire. La rétention complète de toutes les données d'évaluation de coûts pour une simu-

lation “complète” (de durée maximale, soit 302400 secondes) excède la capacité en mémoire RAM d'un PC standard. Pour cette raison, un fichier binaire en flux (*streaming*) est utilisé comme tampon (extension `.cache`). Le module évaluation de performance sauvegarde automatiquement les structures de coûts (objets `Cost`) de manière séquentielle dans ce fichier. Le format est optimisé pour réduire le nombre d'informations à écrire, dans le but d'économiser l'espace disque. Lorsque le logiciel a besoin d'accéder aux structures de coûts déjà calculées (par exemple, lors d'une relecture ou lors d'un saut dans le temps), le pointeur de lecture du fichier tampon est positionné à la position demandée et les structures de coûts pertinentes sont extraites et reconverties en objets `Cost`. À la fin de l'exécution d'une simulation typique comme celles qui seront présentées ultérieurement, le fichier tampon prend une taille qui varie approximativement entre 5 GB et 13 GB.

En plus d'évaluer les coûts pour chacune des zones, le module évaluation de performance effectue également une estimation de coût pour le serveur central, si le fonctionnement est en mode pair-à-pair selon le modèle proposé et également si le fonctionnement est en mode client-serveur. La formules pertinentes pour effectuer ces deux calculs ont été décrites à la section 3.4.7. Ces calculs n'ont pas d'incidence sur les opérations de rebalancement, mais ils servent à produire des statistiques sur la comparaison de consommation de ressources du serveur central entre le modèle traditionnel et le modèle pair-à-pair présenté.

Analyseur de rebalancement

Le module analyseur de rebalancement (classe `RebalancingAnalyzer`) est responsable d'analyser la situation actuelle au temps t et de déterminer quelles opérations de rebalancement sont nécessaires. Ce module intervient immédiatement après le travail du module évaluation de performance. Plus spécifiquement, ce module analyse le ratio de charge actuel de chacune des zones et détermine si une opération de rebalancement doit être effectuée. À cette fin, les paramètres et les algorithmes décrits dans le modèle théorique à la section 3.5 sont utilisés.

Du point de vue de l'implémentation, trois classes représentant les trois opérations de rebalancement possibles sont définies : `ReassignRebalancingAction`, `SplitRebalancingAction` et `MergeRebalancingAction`. Une classe `RebalancingManager` (gestionnaire de rebalancement) est également définie. Cette classe regroupe la liste complète de toutes les actions pour toute la durée de la simulation, à chaque unité de temps t. Lorsque la simulation s'effectue au temps t, le gestionnaire de rebalancement vérifie si l'analyse a déjà été effectuée au temps t (c'est le cas lors d'une relecture, par exemple). Si cette dernière n'a pas été effectuée, le gestionnaire de rebalancement demande à l'analyseur de rebalancement de générer une liste d'*opérations de rebalancement*. Ce dernier récupère les valeurs calculées par le mo-

dule évaluation de performance et trie les zones selon leur ratio de charge. Par la suite, les algorithme discutés précédemment sont exécutés pour produire la liste d'opérations de rebalancement, jusqu'à un maximum donné. Tel que décrit à la section 3.5, les zones les plus chargées sont traitées en premier. Les zones les moins chargées sont ensuite traitées. Finalement, si le nombre maximal d'opérations admissibles par unité de temps n'a pas été atteint, l'algorithme poursuivra le traitement des zones les plus chargées.

Il convient de noter que le module analyseur de rebalancement *planifie* les opérations de rebalancement pertinentes, mais ne les exécute pas immédiatement. Pour chaque opération de rebalancement planifiée, une instance de la classe correspondante est créée et placée dans la liste du gestionnaire de rebalancement par ordre séquentiel. Les opérations seront exécutées au début de la prochaine itération de simulation. En fait, à début de chaque itération au temps t, le gestionaire de rebalancement applique séquentiellement l'ensemble des opérations planifiées jusqu'au temps t qui n'ont pas encore exécutées. Ce mécanisme minimise le temps requis lorsque l'on saute à une valeur arbitraire t lors de la relecture d'une simulation.

Il convient de noter que les classes modélisant les opérations de rebalancement implémentent un dérivé du patron *commande*. Plus précisément, ces classes permettent l'*application* de l'opération (`execute()`) mais également l'*annulation* de l'opération (`undo()`). Ce sont ces méthodes qui ont la responsabilité d'appliquer les changements à la carte du monde (`WorldMap`). La procédure d'application effectue les changements inhérents à l'opération de rebalancement à la carte du monde tandis que l'opération d'annulation effectue les changements requis pour *annuler* l'opération de rebalancement. C'est en raison de cette propriété transactionnelle que l'on peut sauter à n'importe quel instant t dans la simulation lors d'une relecture et également faire défiler la simulation en sens normal et inverse. Notons que chacune des classes d'opérations de rebalancement encapsule les données nécessaires à l'application de l'opération. Ces données, déterminées par le module analyseur de rebalancement, sont passées en paramètres au constructeur. Lorsque l'opération a été appliquée, les données permettant l'annulation de l'opération sont automatiquement déterminées par la classe d'opération.

La figure 4.2 illustre le principe d'echaînement des opérations de rebalancement. La ligne bleue indique l'application successive des opérations de rebalancement pour parcourir la simulation du début à la fin (sens normal) alors que la ligne rouge indique l'application successive des opérations de rebalancement pour partir de la fin et retourner au temps 0 de la simulation (sens inverse).

Générateur de rapports

Le module générateur de rapports (`ReportManager` du paquetage `simulation.reporting` dans le code) est un module optionnel qui peut être utilisé en cours de simulation, ou lorsque la

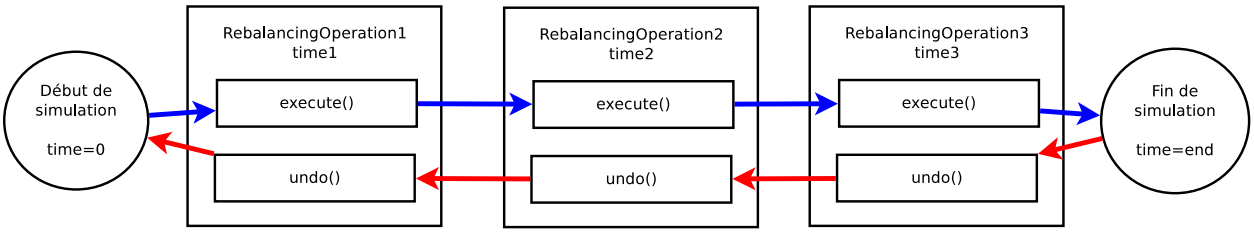


FIGURE 4.2 Illustration du principe d'enchaînement des opérations de rebalancement

simulation complète est terminée. Ce module sert à extraire les données pertinentes à l'étude de la simulation. La simulation en tant que tel regroupe l'ensemble de données pour la durée de la simulation complète, à chaque unité de temps t, pour chaque joueur et pour chaque zone. Il a été décrit précédemment que le volume total de données produites prenait plusieurs giga-octets. Il ne serait pas pratique d'utiliser directement ces données brutes pour effectuer les analyses. Le module générateur de rapports a donc pour objectif de parcourir l'état de la simulation (actuel ou global) et d'extraire les données pertinentes. Le module générateur de rapports est constitué de plusieurs rappors individuels qui sont générés sur demande de l'utilisateur. Chaque rapport est une classe qui dérive de la classe `AbstractReport` et regroupe un ensemble donné de paires clé-valeur (`key-value`). La clé représente le nom d'un paramètre évalué (par exemple, *nombre de joueurs connectés*) et la valeur représente la valeur évaluée (par exemple, *150*).

Deux types de rapports sont définis : les rapports de simulation et les rapports d'analyse. Les rapports de simulation sont des rapports très simples contenant peu de paires clé-valeur qui peuvent être affichés directement à l'écran et mis à jour continuellement selon le déroulement de la simulation. Un exemple d'affichage sera montré à la section 4.1.6. Ces rapports servent à avoir une vue ponctuelle de l'état de la simulation. L'extraction des données pour ces rapports est presque instantanée puisque le rapport est mis à jour par l'interface graphique à chaque itération de simulation. Le tableau 4.1 illustre les principaux rapports de simulation. Notons que les noms des rapports sont en anglais par souci de cohérence par rapport au simulateur.

Les rapports d'analyse sont des rapports beaucoup plus complexes qui sont calculés sur une base ponctuelle uniquement. Ils contiennent beaucoup de données et nécessitent beaucoup de temps à générer. Ces rapports ne sont pas affichables à l'écran. Pour chaque rapport d'analyse, il y a un bouton permettant de le générer et de le sauvegarder dans un fichier `csv`. Les rapports d'analyse parcourent typiquement l'ensemble de la simulation effectuée, extraient les données pertinentes et effectuent un pré-traitement pour éliminer les données

Tableau 4.1 Principaux rapports de simulation

Rapport	Description
General info	Affiche le nombre de joueurs actuellement connectés et déconnectés ainsi que le nombre actuel de zones triangulaires
Player & zones	Affiche quelques extrémums (zones les plus peuplées et le plus de zones affectées à un joueur donné). Ce rapport est désuet.
Player info	Affiche des informations sur le joueur actuellement sélectionné (numéro de joueur, capacité maximale de charge, nombre de zones de domiciliation, latence joueur-hôte).
Player actions	Affiche les actions de mobilité subséquentes (à venir) pour le joueur sélectionné. Pour fins de déverminage.
Zone cost	Affiche les informations servant au calcul du coût de la zone sélectionnée. Pour fins de déverminage, mais très utile.
Performance	Affiche des mesures de temps d'exécution (en temps machine) pour les différents modules de l'application. Sert uniquement au déverminage, pour diagnostiquer les troubles de performance.
Rebal. actions	Affiche les 250 dernières actions de rebalancement effectuées et à venir. Pour fins de déverminage.

redondantes. Le mécanisme de base de fonctionnement en paires clé-valeur est repris par souci de réutilisation de l'architecture logicielle, mais il est légèrement altéré afin que plusieurs "valeurs" puissent être assignées à une même clé. Cela simule un "tableau" (colonnes et lignes). Le format de données `csv` est approprié puisqu'il permet à un chiffrier tel que Microsoft Excel ou OpenOffice Calc d'ouvrir directement le fichier de rapport. Évidemment, c'est le chiffrier qui sert à effectuer les analyses et les synthèses et à produire les tableaux et les graphiques. Cet enchaînement d'opérations a été utilisé ici pour produire les données de simulation, les tableaux et les graphiques pour la rédaction de ce mémoire. Le tableau 4.2 illustre les principaux rapports d'analyse. Notons que les noms des rapports sont en anglais par souci de cohérence par rapport au simulateur.

4.1.6 Interface graphique

Tel que discuté, l'interface graphique joue un rôle prépondérant au sein du simulateur puisqu'elle permet de visualiser le déroulement de la simulation. L'aire de visualisation occupe la partie centrale de l'écran. Les éléments affichées dans l'aire de visualisation sont paramétrable à partir du menu *View*. Les fonctionnalités offertes par l'aire de visualisation sont les suivantes :

- affichage des zones triangulaires selon le mode de coloration choisi ;
- affichage des joueurs et de leur zone d'intérêt ;

Tableau 4.2 Principaux rapports d'analyse

Rapport	Description
[XT] : CS/P2P Comparison	Pour chaque minute de simulation, extrait le coût moyen imposé au serveur central par seconde de simulation si le mode de fonctionnement est pair-à-pair et client-serveur. Sert à évaluer la performance du modèle proposé par rapport au modèle classique.
[XT] : Per-second	Pour chaque seconde de simulation, extrait et pré-traite différentes valeurs telles que le nombre de joueurs connectés, le nombre de zones, le nombre de zones par classe de ratio de charge, le nombre de zones par classe de latence moyenne et le nombre de joueurs par classe de latence moyenne. Très utile pour l'évaluation de fonctionnement du modèle sous plusieurs conditions selon les simulations réalisées.
[XT] : Per-player	Inutilisé pour l'instant car non-implémenté.

- affichage d'informations textuelles sur les zones ;
- affichage d'une ligne reliant chaque zone avec son noeud responsable ;
- affichage du graphe d'adjacence des zones (reliant chaque zone avec l'ensemble de ses zones voisines) ;
- déplacement de la portion de la vue affichée actuellement (clôture) à l'aide du bouton droit de la souris ;
- mise en sélection (surbrillance) de joueurs et de zones (utilisé par certains rapports qui affichent l'état des zones et des joueurs sélectionnés).

Un panneau de contrôle occupe la partie droite de l'écran. Ce panneau permet de manipuler l'état actuel de la vue et d'afficher les informations produites par les différents rapports décrits à la section 4.1.5. Les différentes commandes proposées par le panneau de simulations sont les suivantes :

- manipuler le niveau de zoom pour visualiser la simulation à petite ou grande échelle (vue globale imprécise ou vue locale plus détaillée) ;
- se positionner à un temps t différent dans la simulation. Notons qu'en raison des contraintes énoncées à la section 4.1.5, il est possible de se positionner uniquement à un instant t avant la fin de la simulation.
- régler la vitesse de déroulement ou de relecture de la simulation ;
- visualiser la liste des rapports disponibles et choisir quels rapports doivent être affichés (rapports de simulation) et sauvegardés (rapports de simulation et d'analyse).

Une barre de menus vient compléter l'interface graphique. Les différents menus sont les suivants :

- *File* : permet de charger et sauvegarder la carte du monde, le bassin de joueur et le

fichier de simulation.

- *View* : permet de sélectionner les éléments qui sont affichés dans l'aire de visualisation, sélectionner le mode de coloration des zones (selon leur identificateur, leur ratio de charge ou leur nombre de joueurs), sélectionner une zone via son identificateur et visualiser une zone via son identificateur.
- *State* : permet de définir l'action courante associée à chaque bouton de la souris (gauche, droit, milieu). L'action sélectionnée est exécutée lorsque le bouton correspondant de la souris est appuyé sur l'aire de visualisation.
- *Mobility* : permet d'appliquer un patron de mobilité donné à l'ensemble des joueurs.
- *Misc* : contient des fonctionnalités servant uniquement au déverminage.

Notons que certaines commandes qui prennent du temps à s'exécuter émettent du texte dans la console de sortie Java. Il peut être judicieux de jeter un coup d'oeil à cette console pour suivre le déroulement et obtenir certaines informations complémentaires.

Les figures 4.3, 4.4 et 4.5 représentent trois captures-écran du simulateur. La première capture-écran illustre une vue d'ensemble avec une coloration des zones selon la couleur associée à leur noeud responsable. La seconde capture-écran illustre la même vue d'ensemble avec une coloration basée sur la charge actuelle des zones. La troisième capture-écran illustre une vue rapprochée pour voir précisément l'état de quelques zones.

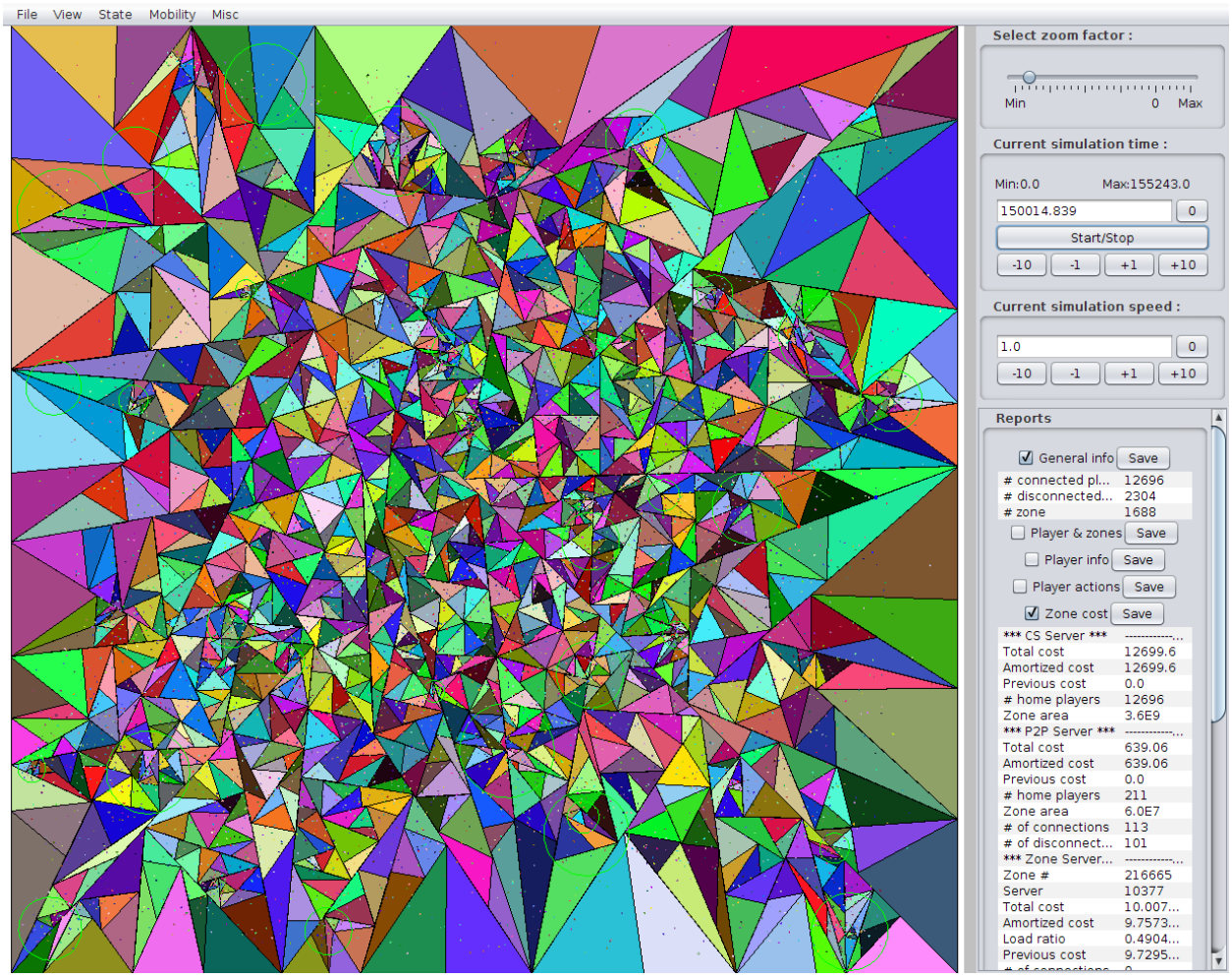


FIGURE 4.3 Capture-écran du simulateur (1) (vue éloignée)

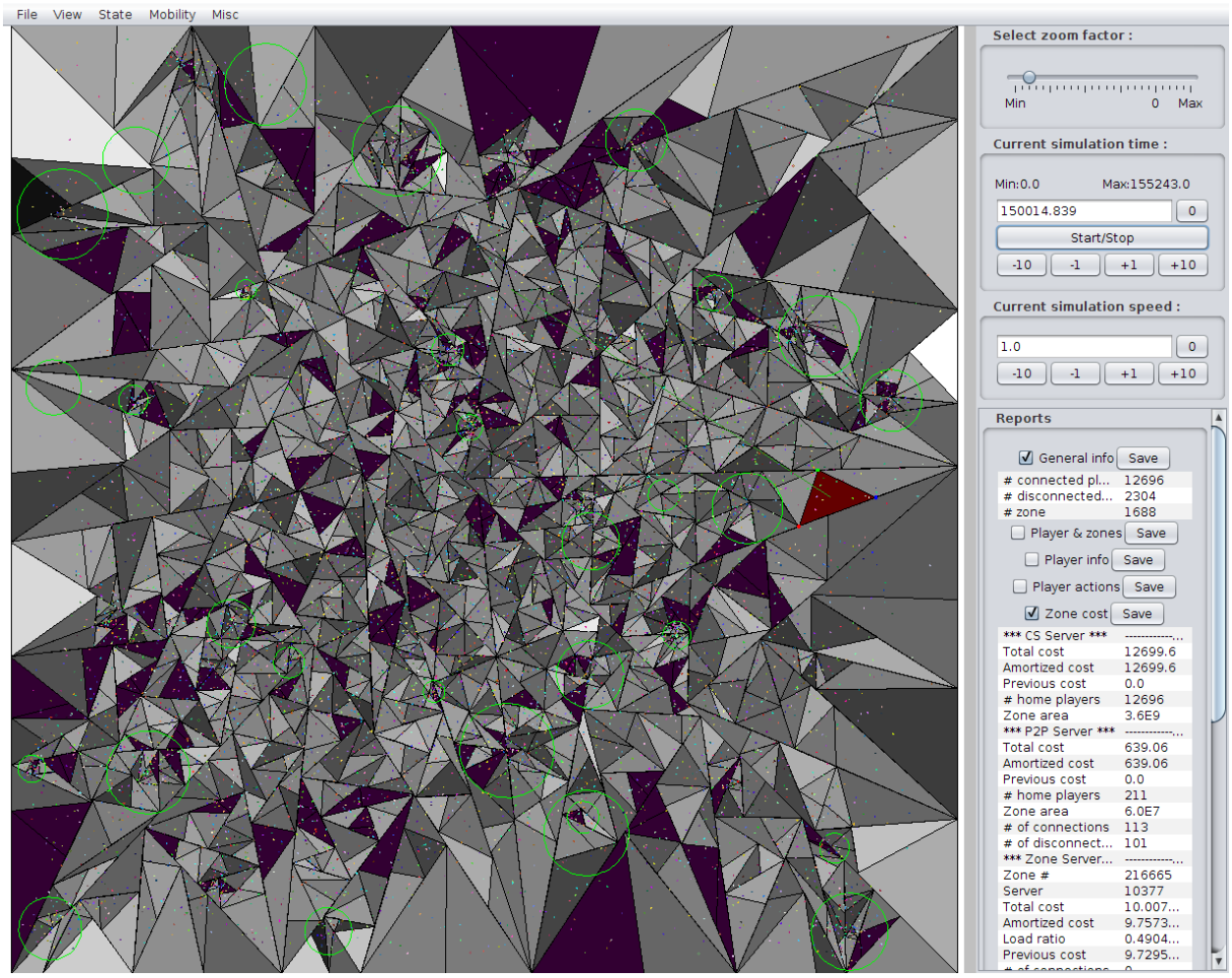


FIGURE 4.4 Capture-écran du simulateur (2) (vue éloignée)

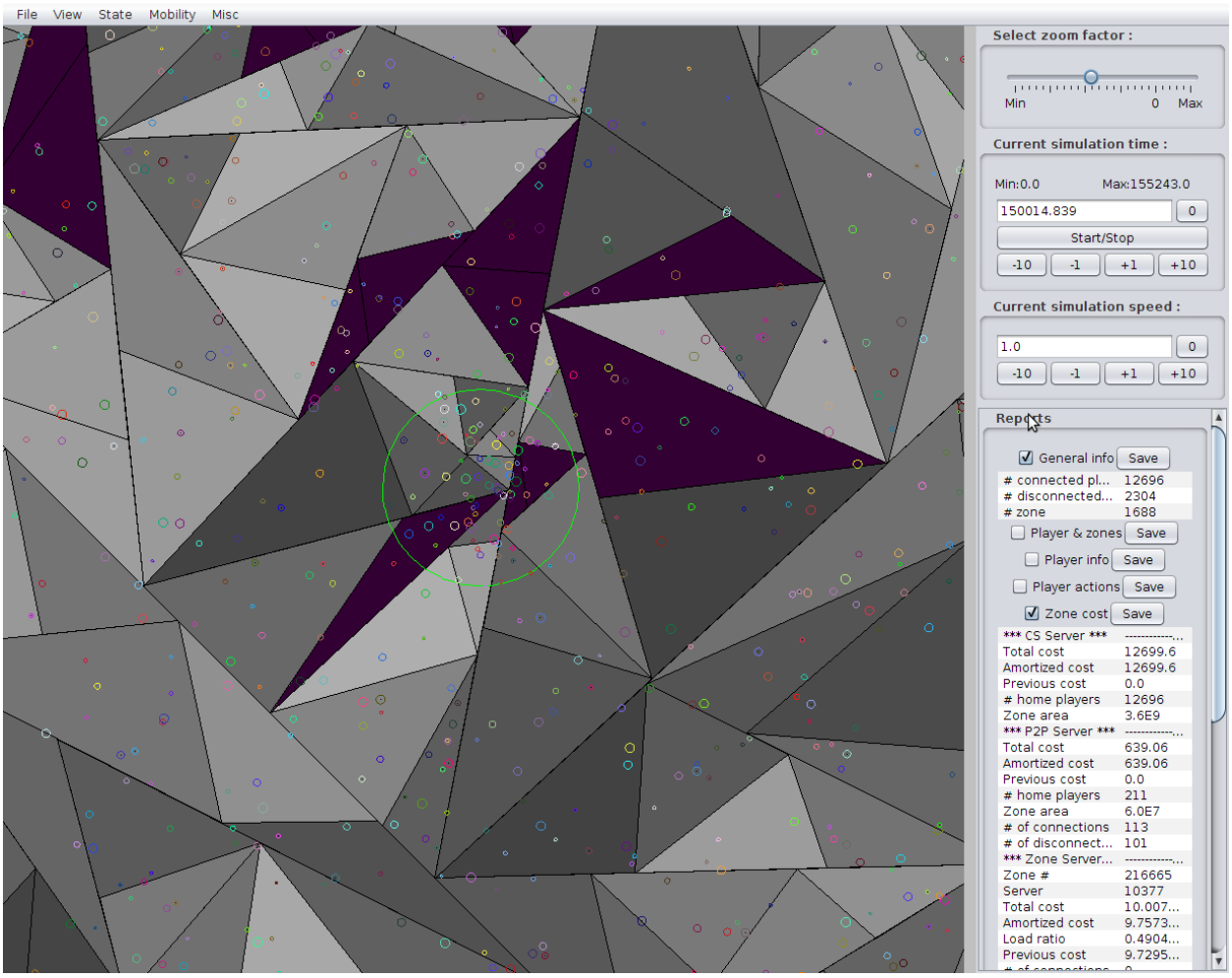


FIGURE 4.5 Capture-écran du simulateur (3) (vue rapprochée)

4.2 Paramètres des simulations

Au total, six simulations ont été réalisés. Chaque simulation fait varier certains paramètres pertinents pour vérifier le fonctionnement du modèle. Nous avons visé un temps total de simulation de $3,5$ jours, ce qui correspond à 84 heures ou 302400 secondes ; c'est-à-dire que le simulateur devait simuler le déroulement d'un jeu massivement multijoueurs pour chaque seconde t de l'ensemble de la durée définie. En ce qui concerne les simulations qui se sont exécutées correctement, il n'y a eu aucun problème à laisser tourner le simulateur pour la durée prévue. Cependant, en ce qui concerne les simulations dont les paramètres choisis n'ont pas permis de donner des résultats satisfaisants, nous n'avons pas pu les laisser s'exécuter complètement pour toute la durée prévue. Il a été nécessaire de les interrompre à environ 70%-80% de leur durée. Ces limitations seront discutées davantage à la présentation des résultats (section 4.3).

Les tableaux 4.3 et 4.4 détaillent l'ensemble des paramètres de simulation considérés. Pour fins de compréhension, un bref rappel des paramètres est présenté. Nous pouvons remarquer que le simulateur possède une très grande quantité de paramètres. Cependant, ces paramètres n'ont pas tous la même importance. Au fur et à mesure de l'implémentation du modèle, des valeurs plausibles et fonctionnelles ont été affectées aux différents paramètres. Vu la nature du travail de maîtrise, il était impossible d'étudier la variation de tous les paramètres. Certains paramètres-clés importants ont été identifiés. Ce sont ces paramètres qui varieront pour les différentes simulations. Ces paramètres apparaîssent en caractères gras.

Pour l'ensemble des simulations, une carte de forme carrée de superficie 40 000 unités carrés et un nombre de 15 000 joueurs inscrits (bassin de joueurs) ont été considérés. Ce ratio (nombre de joueurs par unité de superficie) a été estimé empiriquement à partir d'observations de jeux massivement multijoueurs en ligne actuellement sur le marché. Notons ici que le nombre de joueurs actuellement dans le jeu n'est pas de 15 000 tout au long du déroulement de la partie. Tel que décrit dans le modèle de mobilité à la section 3.2, un patron de connectivé a été intégré, ce qui fait en sorte que les joueurs peuvent entrer et quitter le jeu à tout moment. D'une part, nous avons voulu modéliser le comportement typique des joueurs qui ont plus tendance à jouer à certaines heures clés (soirée et début de nuit) par rapport à certaines heures creuses (matin et début d'après-midi). D'autre part, nous avons voulu tester le modèle sous des conditions plus rigoureuses pour évaluer le rendement. À cette fin, nous avons établi un jeu de paramètres de connectivité prenant en compte empiriquement le patron de connectivité typique des joueurs de tels jeux. Cependant, les "variations" au niveau la répartition des joueurs connectés en fonction de l'heure de la journée ont été amplifiées. Le nombre de joueurs maximal est d'environ 14650 et le nombre de joueurs minimal est d'environ 1300.

Cela constitue un grand écart (un facteur supérieur à 10), afin de vérifier si le modèle est en mesure de s'adapter correctement. Le patron de connectivité fonctionne sur la base des heures d'une journée de temps simulé. Pour cette raison, la répartition des joueurs connectés en fonction du temps est très similaire d'une journée simulée à l'autre.

La section *mobilité* du tableau 4.3 détaille les paramètres de mobilité. L'ensemble de ces paramètres ont été définis empiriquement afin qu'ils soient plausibles et fonctionnels et demeurent fixes pour toutes les simulations. v_{min}, v_{max}, p_{min} et p_{max} représentent respectivement la vitesse minimale et maximale de déplacement et le temps de pause minimal et maximal pour chaque action de mobilité générée. μ_d et σ_d représentent respectivement la moyenne et l'écart-type de la durée des périodes de déconnexion générées. P_d représente la probabilité qu'une action de mobilité générée soit une action de déconnexion et W_d^0 à W_d^{23} représentent les facteurs modulant la probabilité de générer une action de déconnexion selon de l'heure du jour du temps simulé.

Il convient de préciser qu'en début de simulation, tous les joueurs sont déconnectés (la première action de mobilité est toujours une action de déconnexion). Dès la fin de cette première action, les joueurs se connectent progressivement, suivant le déroulement normal du modèle de mobilité utilisé. Cela sert à simuler la phase d'initialisation du jeu dans laquelle une partie est amorcée sur le serveur central et les joueurs s'y joignent progressivement. De plus, après trois jours (72 heures) de simulation, le serveur entre en mode "fermeture" (tous les joueurs doivent quitter la partie). À cette fin, l'ensemble des actions de mobilité générées après ce délai sont des actions de déconnexion. Cette procédure induit une libération progressive du serveur central, servant à simuler une phase de "maintenance" du jeu dans laquelle tous les joueurs doivent quitter, sans nécessairement forcer une déconnexion dure simultanée. La simulation dure 84 heures ; par conséquent, durant les 12 dernières heures, l'ensemble des actions générées sont des actions de déconnexion. À la fin des simulations, il ne reste que quelques joueurs connectés (pas plus d'une dizaine).

La section *régions d'intérêt* du tableau 4.4 détaille les paramètres de génération afférents aux régions d'intérêt pour les différentes simulations. $p_{stayInsideHotspot}$ représente la probabilité qu'une action de mobilité générée soit à l'intérieur de la même région d'intérêt (s'applique uniquement si le joueur est déjà situé dans une région d'intérêt). $W_{outsideHotspot}$ représente le "poids" associé à la génération d'une action de mobilité hors de toute région d'intérêt lors du calcul de la prochaine position de joueur. n_{min} et n_{max} représentent respectivement le nombre minimal et maximal de régions d'intérêt qui seront générées à chaque jour simulé. r_{min} et r_{max} représentent respectivement le rayon minimal et maximal des régions d'intérêt générées. d_{min} et d_{max} représentent respectivement la durée minimale et maximale d'existence des régions d'intérêt générées. Finalement, W_{min} et W_{max} représentent respectivement le poids

minimal et maximal associé aux différentes régions d'intérêt générées. Tel que décrit dans le modèle, rappelons qu'il existe des régions d'intérêt temporaires et permanentes. Les régions temporaires sont en vigueur pour la durée choisie, alors que les régions permanentes sont en vigueur pour toute la durée du jeu. Les paramètres n_{min} et n_{max} sont donc "doublement" considérés : pour générer les régions d'intérêt pour une nouvelle journée donnée et pour générer les régions d'intérêt permanentes. À tout instant t, il faut donc s'attendre à avoir un nombre moyen de régions d'intérêt en vigueur qui est le double de ce que les paramètres définissent.

Dans le cadre de ce travail, pour les simulations 1, 2, 3 et 5, nous avons affecté aux paramètres $W_{outsideHotspot}$, n_{min} et n_{max} respectivement les valeurs 6, 10 et 30. Ces valeurs sont considérées comme "standard". Pour la simulation 4, nous avons réduit le nombre de régions d'intérêt générées d'environ 70% pour évaluer l'impact d'une plus grande concentration de joueurs dans peu d'endroits ($W_{outsideHotspot} = 2$, $n_{min} = 3$, $n_{max} = 9$). Pour la simulation 6, nous avons supprimé l'ensemble des régions d'intérêt. Cela équivaut au modèle *random waypoint* classique puisque la notion de régions d'intérêt est complètement abstraite ($W_{outsideHotspot} = 6$, $n_{min} = 0$, $n_{max} = 0$).

La section *capacité de charge* détaille les paramètres qui ont été considérés pour modéliser la capacité de charge maximale des noeuds-joueurs, tel que décrit à la section 3.2.3 du modèle. Pour les simulations 1, 4, 5 et 6, une capacité de charge considérée "standard" (ou moyenne) a été utilisée (moyenne μ_c de 12 et écart type σ_c de 5). Pour la simulation 2, nous avons décidé de simuler une capacité de charge doublée ($\mu_c = 24$ et $\sigma_c = 10$) pour évaluer le fonctionnement avec des noeuds possédant une capacité doublée. Pour la simulation 3, nous avons décidé de simuler une capacité de charge des noeuds réduite de moitiée ($\mu_c = 6$ et $\sigma_c = 2,5$). Les latences entre chaque paire de noeuds sont modélisées pour s'approcher de la réalité, selon une distribution obtenue grâce à une étude exhaustive réalisée sur le sujet (Gummadi *et al.* (2002)).

La section *pondération des coûts* détaille les facteurs pondératifs pour les différents paramètres servant à calculer la charge totale d'une zone donnée. Ces paramètres ont été décrits de façon plus détaillée à la section . Ces valeurs ont été obtenues empiriquement et ne varient pas. W_h, W_v et W_r représentent respectivement les coûts de maintenance d'un joeur domicile, visiteur et relais pour un noeud serveur donné. W_c et W_d représentent respectivement le coût de connexion et de déconnexion d'un noeud joueur à un noeud serveur donné. W_a représente le coût de gestion associé à la maintenance d'une unité carrée de superficie.

La section *rebalancement* détaille les paramètres servant à générer les opérations de rebalancement. Ces paramètres, déterminés empiriquement, ne changent pas pour les six simulations. Seul le paramètre t_b change pour la première simulation uniquement. LR_{high} et

LR_{low} représentent respectivement les seuils à partir desquels une zone est considérée en sous-charge et en surcharge. L^{avg}_{high} représente le seuil à partir duquel l'on considère qu'une zone donnée a une latence moyenne trop élevée. LT^{R}_{max} représente la charge maximale admissible pour une opération de fusion. n_σ représente le seuil en nombre d'écarts type à partir duquel une opération de séparation sera directement considérée plutôt qu'une opération de réassignation. n_r^{high} et n_r^{low} représentent respectivement le nombre maximal d'opérations de rebalancement qui seront effectuées pour les zones surchargées et sous-chargées. t_b représente le temps de "liste noire" (blacklist) en secondes pour une zone récemment rebalancée. Pour la première simulation, un temps de 30 secondes a été considéré. Toutefois, en analysant les résultats, nous avons réalisé que le délai était beaucoup trop élevé et faisait en sorte que certaines zones devenaient surchargées pendant un temps considérable. Pour les simulations subséquentes, nous avons donc décidé de réduire ce délai à 10 secondes. $angle_{min}$ représente la valeur minimale admise pour les trois angles de toute nouvelle zone créée.

TABLEAU 4.3 Paramètres de simulation

Simulation	1	2	3	4	5	6
Temps visé	302400	302400	302400	302400	302400	302400
Temps réel	302400	302400	246117	209102	302400	302400
Joueurs	15000	15000	15000	15000	15000	15000
Superficie	40000^2	40000^2	40000^2	40000^2	40000^2	40000^2
Mobilité						
v_{min}	1	1	1	1	1	1
v_{max}	2	2	2	2	2	2
p_{min}	50	50	50	50	50	50
p_{max}	60	60	60	60	60	60
μ_d	21600	21600	21600	21600	21600	21600
σ_d	14400	14400	14400	14400	14400	14400
P_d	$0,01$	$0,01$	$0,01$	$0,01$	$0,01$	$0,01$
W_d^0	$0,503$	$0,503$	$0,503$	$0,503$	$0,503$	$0,503$
W_d^1	$0,542$	$0,542$	$0,542$	$0,542$	$0,542$	$0,542$
W_d^2	$0,647$	$0,647$	$0,647$	$0,647$	$0,647$	$0,647$
W_d^3	$1,187$	$1,187$	$1,187$	$1,187$	$1,187$	$1,187$
W_d^4	$1,993$	$1,993$	$1,993$	$1,993$	$1,993$	$1,993$
W_d^5	$2,931$	$2,931$	$2,931$	$2,931$	$2,931$	$2,931$
W_d^6	$3,56$	$3,56$	$3,56$	$3,56$	$3,56$	$3,56$
W_d^7	$3,56$	$3,56$	$3,56$	$3,56$	$3,56$	$3,56$
W_d^8	$3,322$	$3,322$	$3,322$	$3,322$	$3,322$	$3,322$
W_d^9	$3,115$	$3,115$	$3,115$	$3,115$	$3,115$	$3,115$
W_d^{10}	$2,769$	$2,769$	$2,769$	$2,769$	$2,769$	$2,769$
W_d^{11}	$2,492$	$2,492$	$2,492$	$2,492$	$2,492$	$2,492$
W_d^{12}	$2,492$	$2,492$	$2,492$	$2,492$	$2,492$	$2,492$
W_d^{13}	$1,718$	$1,718$	$1,718$	$1,718$	$1,718$	$1,718$
W_d^{14}	$1,424$	$1,424$	$1,424$	$1,424$	$1,424$	$1,424$
W_d^{15}	$1,51$	$1,51$	$1,51$	$1,51$	$1,51$	$1,51$
W_d^{16}	$1,311$	$1,311$	$1,311$	$1,311$	$1,311$	$1,311$
W_d^{17}	$1,107$	$1,107$	$1,107$	$1,107$	$1,107$	$1,107$
W_d^{18}	$0,906$	$0,906$	$0,906$	$0,906$	$0,906$	$0,906$
W_d^{19}	$0,664$	$0,664$	$0,664$	$0,664$	$0,664$	$0,664$
W_d^{20}	$0,548$	$0,548$	$0,548$	$0,548$	$0,548$	$0,548$
W_d^{21}	$0,479$	$0,479$	$0,479$	$0,479$	$0,479$	$0,479$
W_d^{22}	$0,445$	$0,445$	$0,445$	$0,445$	$0,445$	$0,445$
W_d^{23}	$0,453$	$0,453$	$0,453$	$0,453$	$0,453$	$0,453$

TABLEAU 4.4 Paramètres de simulation (suite)

Simulation	1	2	3	4	5	6
Temps visé	302400	302400	302400	302400	302400	302400
Temps réel	302400	302400	246117	209102	302400	302400
Joueurs	15000	15000	15000	15000	15000	15000
Superficie	40000^2	40000^2	40000^2	40000^2	40000^2	40000^2
Régions d'intérêt						
$p_{stayInsideHotspot}$	$0,7$	$0,7$	$0,7$	$0,7$	$0,7$	$0,7$
$W_{outsideHotspot}$	**6**	**6**	**6**	**2**	**6**	**6**
n_{min}	**10**	**10**	**10**	**3**	**10**	**0**
n_{max}	**30**	**30**	**30**	**9**	**30**	**0**
r_{min}	600	600	600	600	600	600
r_{max}	3000	3000	3000	3000	3000	3000
d_{min}	3600	3600	3600	3600	3600	3600
d_{max}	72000	72000	72000	72000	72000	72000
W_{min}	$0,5$	$0,5$	$0,5$	$0,5$	$0,5$	$0,5$
W_{max}	2	2	2	2	2	2
Capacité de charge						
μ_c	**12**	**24**	**6**	**12**	**12**	**12**
σ_c	**5**	**10**	**2,5**	**5**	**5**	**5**
Latency	KING	KING	KING	KING	KING	KING
Pondérations des coûts						
W_h	1	1	1	1	1	1
W_v	1	1	1	1	1	1
W_r	1	1	1	1	1	1
W_c	2	2	2	2	2	2
W_d	2	2	2	2	2	2
W_a	10^{-9}	10^{-9}	10^{-9}	10^{-9}	10^{-9}	10^{-9}
Rebalancement						
LR_{high}	$0,3$	$0,3$	$0,3$	$0,3$	$0,3$	$0,3$
LR_{low}	$0,8$	$0,8$	$0,8$	$0,8$	$0,8$	$0,8$
L_{high}^{avg}	200	200	200	200	200	200
LT_{max}^{R}	$0,5$	$0,5$	$0,5$	$0,5$	$0,5$	$0,5$
n_σ	1	1	1	1	1	1
n_r^{high}	5	5	5	5	5	5
n_r^{low}	5	5	5	5	5	5
t_b	**30**	**10**	**10**	**10**	**10**	**10**
$angle_{min}$	30	30	30	30	30	30

4.3 Résultats des simulations

Les six simulations ont été exécutées avec les paramètres décrits dans les tableau 4.3 et 4.4. Par la suite, la procédure de génération des rapports `CS/P2P Comparison` et `Per-second` a été invoquée pour produire les fichiers de données. Les analyses de données ont ensuite été réalisées pour produire les graphiques et tableaux des résultats. La prochaine section (4.3.1) compare la charge totale imposée au serveur central entre le modèle client-serveur et pair-à-pair pour chacune des six simulations. Les sections suivantes analysent respectivement l'impact des opérations de rebalancement (4.3.2), la charge imposée aux noeuds serveurs (4.3.3) et la latence (4.3.4).

4.3.1 Comparaison avec le modèle client-serveur

Les équations mathématiques permettant de calculer les coûts imputés au serveur central par unité de temps ont été décrites à la section 3.4.7. Ces formules ont été appliquées aux six simulations réalisées afin de calculer les coûts pour le serveur central, selon le modèle pair-à-pair et selon le modèle classique. Les graphiques 4.6, 4.7, 4.8, 4.9, 4.10 et 4.11 présentent les coûts selon ces deux modèle tout au long de la durée du jeu simulé.

Mathématiquement, rappelons que le coût imputé au serveur central selon le modèle client-serveur est calculé à partir du nombre de joueurs présents et de l'aire du territoire global (équation 3.13). Le nombre de joueurs, la taille du territoire et les patrons de connectivité sont les même dans toutes les simulations effectuées. Conséquemment, tel qu'attendu, les coûts du serveur central pour le modèle classique sont presques identifiques pour les six simulations effectuées.

Il y a variation au niveau des coûts associés au serveur central selon le modèle pair-à-pair (équation 3.12). Mathématiquement, la première partie de ce coût est calculée à partir du nombre de joueurs et de la superficie du territoire global, mais atténué par le facteur B_i (intervalle temporel de propagation). Ce coût demeure stable pour les mêmes conditions énoncées pour le calcul des coûts pour le modèle classique. La seconde partie du coût est constituée du nombre de *connexions* et *déconnexions* (coûts de migration) puisque l'on suppose que le serveur central est avisé lorsqu'un tel événement de connexion ou déconnexion est généré. Ce sont ces facteurs qui font varier les coûts associés au serveur central sous le modèle pair-à-pair.

Les coûts selon le modèle pair-à-pair sont presque identiques pour les simulations 1 et 5, qui sont des simulations selon des paramètres en conditions normales (rappelons que seul l'intervalle de liste noire, t_b, change, ce qui a peu d'impact). Les coûts sont également très similaires quoique légèrement inférieurs pour la simulation 6. Cela s'explique par le fait qu'il

n'y ait aucune région d'intérêt dans cette simulation, ce qui favorise des zones de superficie mieux répartie, pouvant ainsi contribuer à réduire les migrations de joueurs. La différence est toutefois très peu significative. Les coûts pour la simulation 2 sont significativement inférieurs à ceux des simulations 1 et 5. Rappelons que les noeuds de la région 2 ont une capacité de charge doublée. Cela permet aux noeuds de gérer des zones de taille supérieure, induisant moins de zones pour un même nombre de joueurs, réduisant ainsi le nombre de migrations. Les coûts associés à la simulation 3 sont plus élevés, mais l'écart semble toutefois raisonnable. Cette simulation représente le cas opposé à la simulation 2 : les noeuds joueurs ont une puissance réduite de moitié par rapport à la normale, ce qui induit le besoin d'avoir un plus grand nombre de zones. Il y a davantage de migrations. Toutefois, les vrais problèmes engendrés par la simulation 3 n'apparaîssent pas dans cette analyse. Cela signifie que le serveur central du jeu ne sera pas beaucoup affecté par les conditions restrictives imposées. Les problèmes engendrés par ces conditions seront détaillés dans les analyses subséquentes. Finalement, la simulation 4 s'avère très problématique puisque les coûts du serveur central pair-à-pair sont exagérément élevés. Cela est dû au fait que la simulation 4 possède très peu de régions d'intérêt qui sont fortement surpeuplées, induisant une multitude de zones de taille très petite à l'intérieur de ces régions d'intérêt. Les coûts de migration qui en résultent sont énormes.

Le graphique 4.12 illustre le ratio de charge serveur économisée en utilisant le modèle pair-à-pair tout au long du déroulement des simulations. Nous remarquons que pour toutes les simulations sauf la 4, ce ratio est toujours inférieur à 90% et avoisine souvent 95% et plus pour certaines simulations. Pour la simulation 4, nous constatons beaucoup de variations et de chutes de performance.

En rapport à ces observations, nous remarquons que le modèle pair-à-pair permet de réduire significativement le charge imposée au serveur central dans les situations où la taille des zones demeure raisonnable, mais n'est pas efficace dans les situations où il y a beaucoup de zones de taille très réduite.

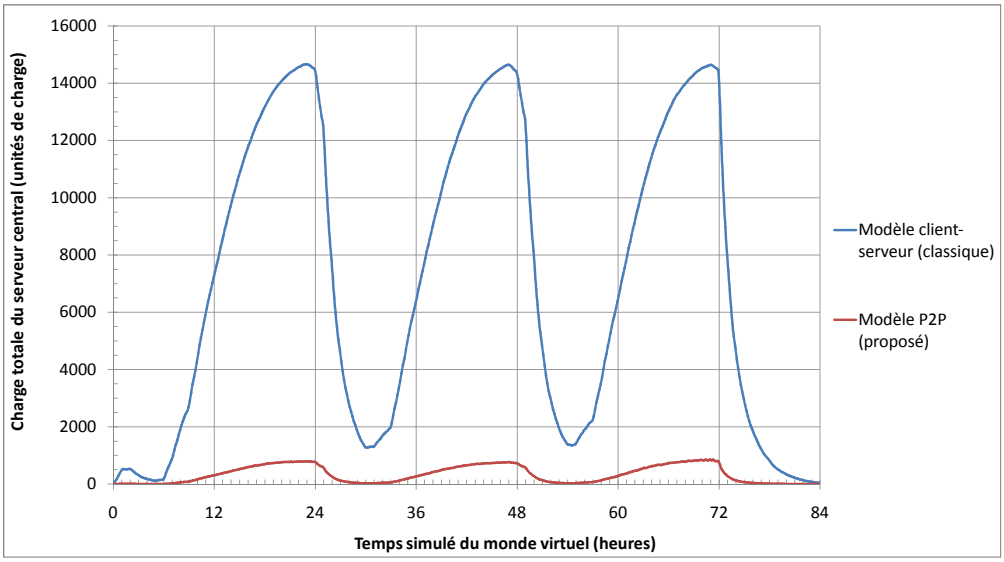


FIGURE 4.6 Charge totale imposée au serveur central en fonction du temps (simulation 1)

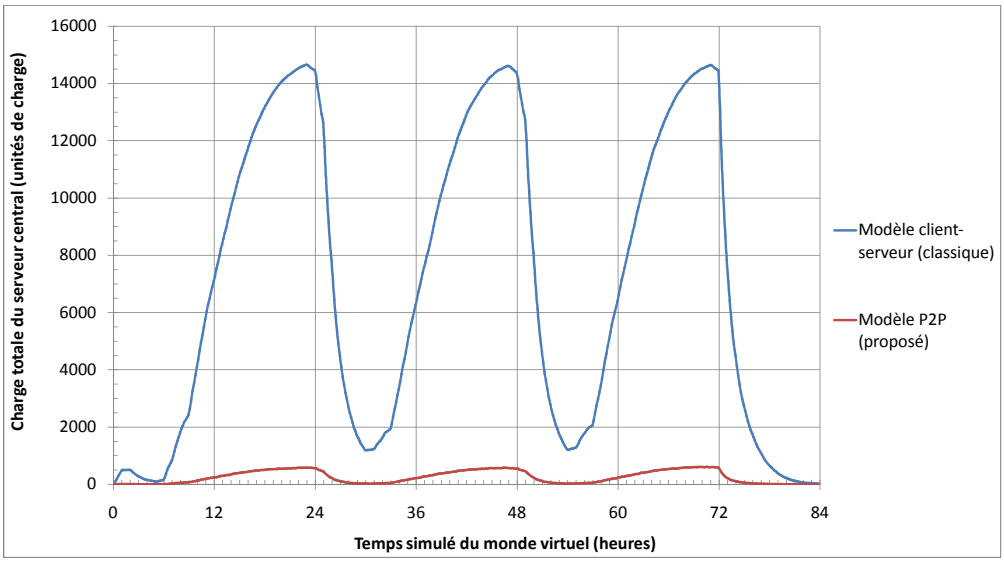


FIGURE 4.7 Charge totale imposée au serveur central en fonction du temps (simulation 2)

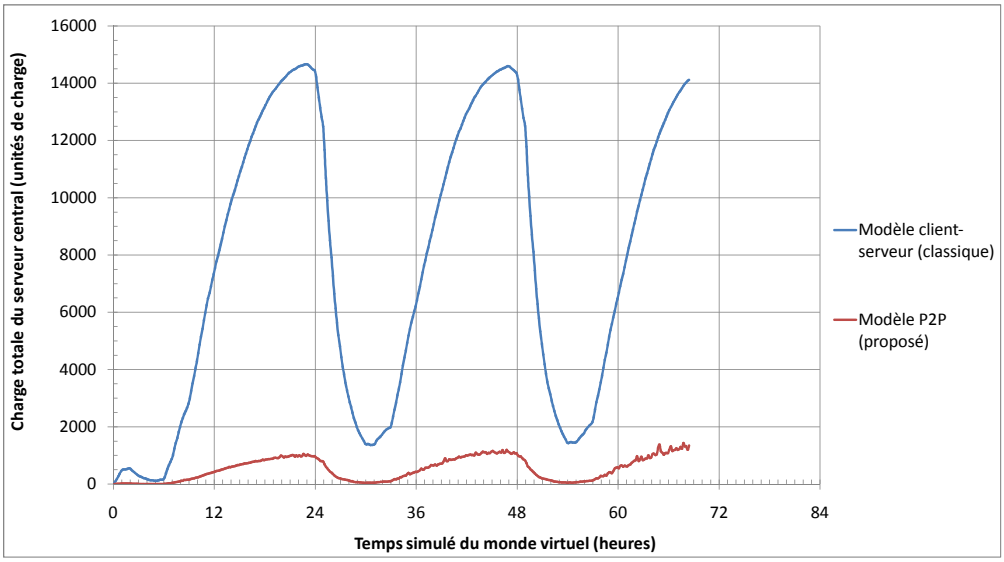


FIGURE 4.8 Charge totale imposée au serveur central en fonction du temps (simulation 3)

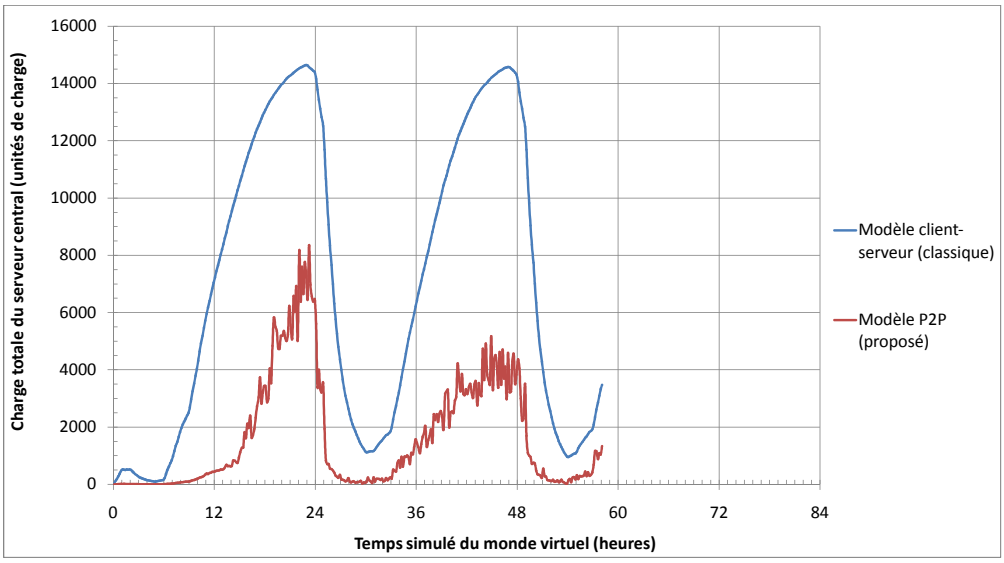


FIGURE 4.9 Charge totale imposée au serveur central en fonction du temps (simulation 4)

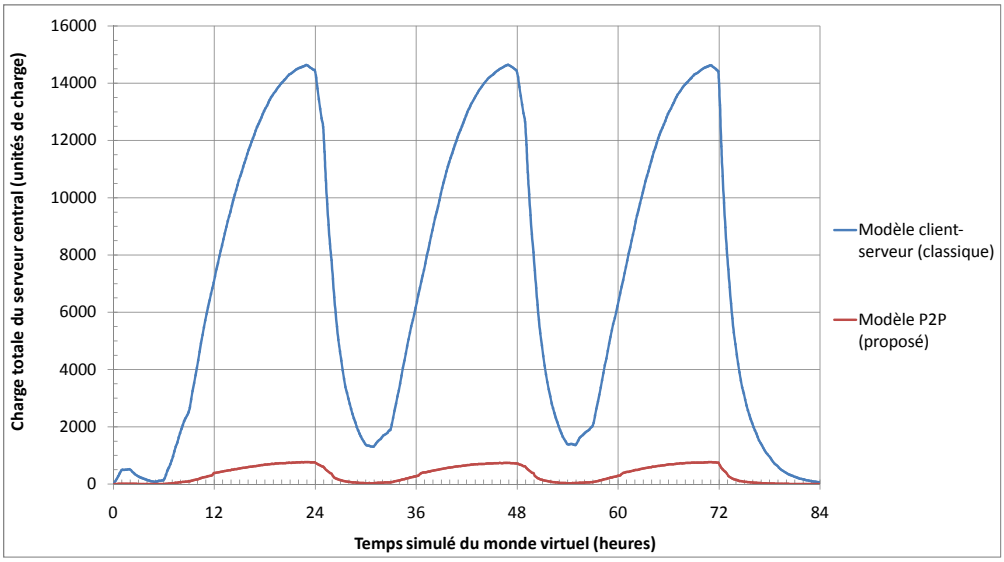


FIGURE 4.10 Charge totale imposée au serveur central en fonction du temps (simulation 5)

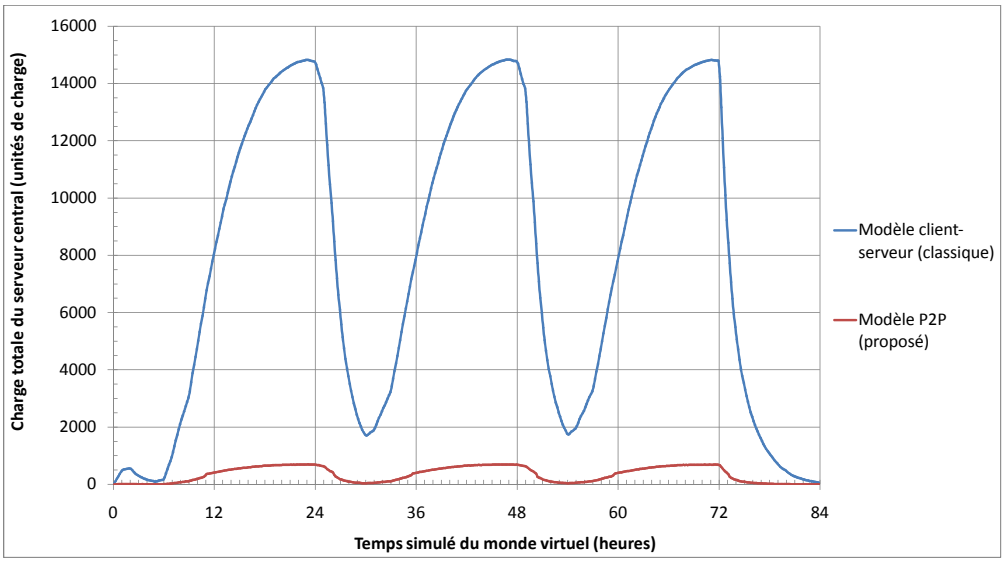


FIGURE 4.11 Charge totale imposée au serveur central en fonction du temps (simulation 6)

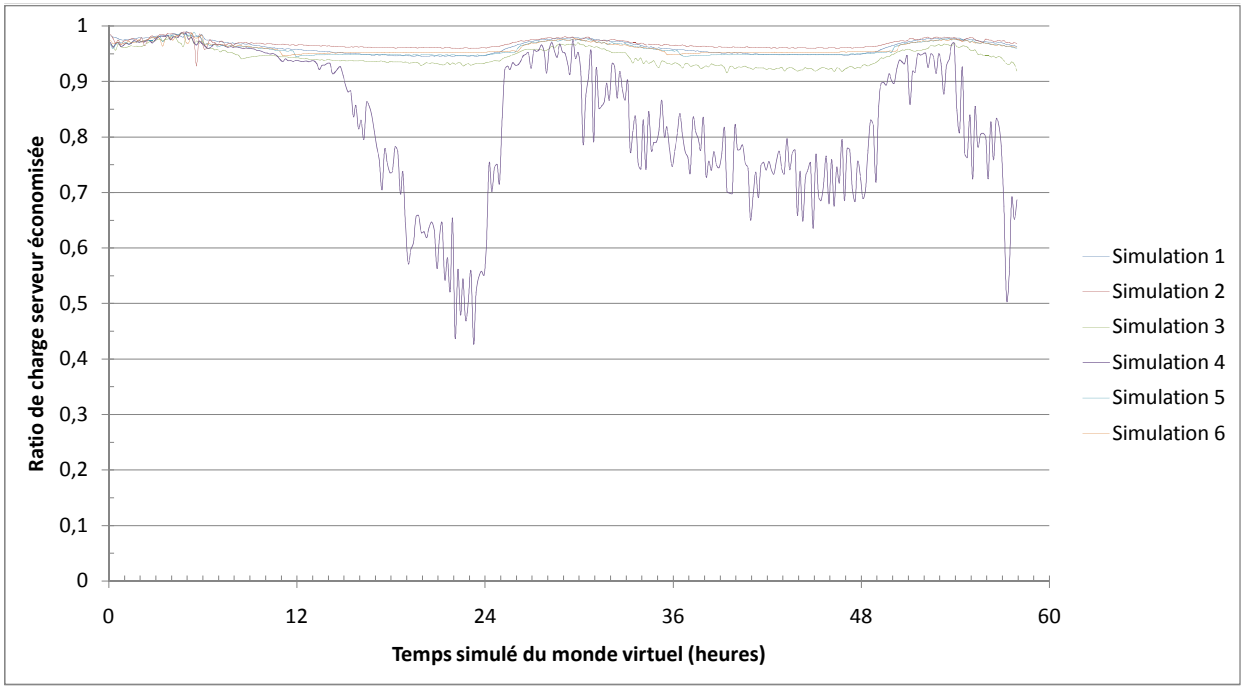


FIGURE 4.12 Ratio d'économie de charge au serveur central en utilisant le modèle P2P

4.3.2 Analyse des rebalancements

Cette section détaille des statistiques sur les opérations de rebalancement qui ont été effectuées. Plus précisément, pour chaque simulation, le premier graphique illustre le nombre de joueurs et de serveurs de zones en fonction du temps de simulation. Le second graphique illustre, pour chaque minute de simulation et groupé en fonction du nombre de joueurs connectés :

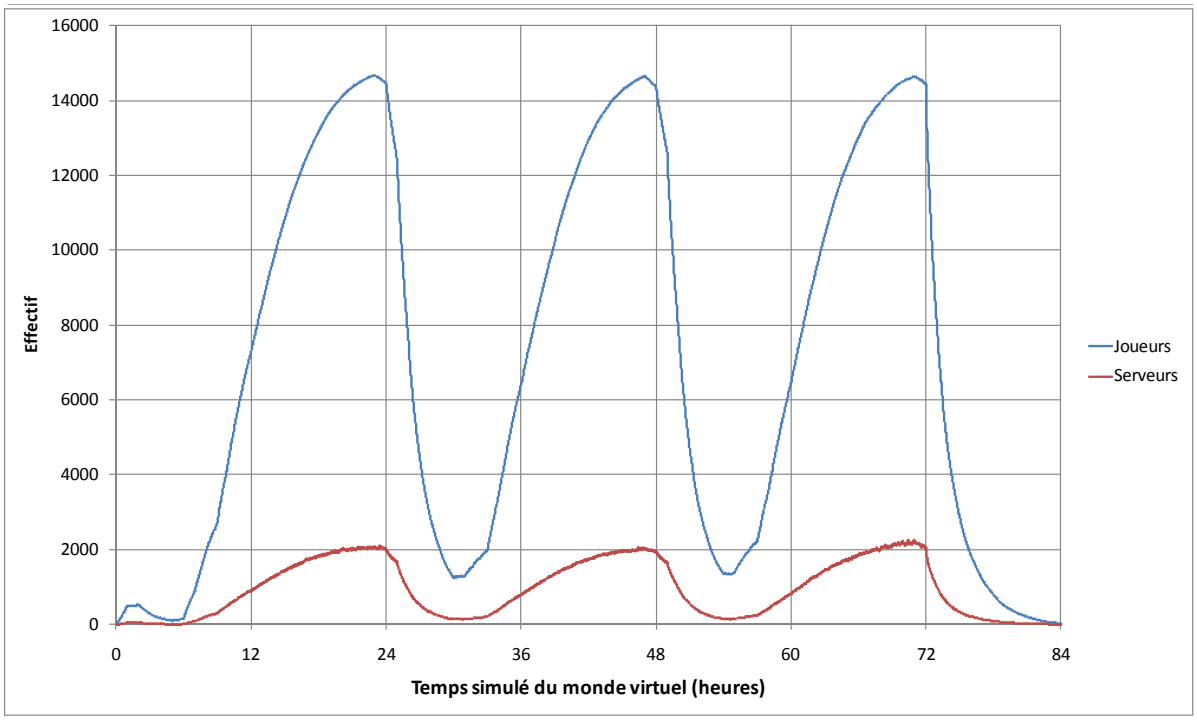
- le pourcentage de noeuds serveurs (ou zones) affectés par une opération de rebalancement ;
- le pourcentage de joueurs affectés par une opération de rebalancement ;
- le pourcentage de superficie affectée par une opération de rebalancement (par rapport à la superficie du territoire global).

Lors du traitement des données, les valeurs sont calculées par seconde et ensuite agrégées sur la base d'une minute. Pour chaque minute, une même zone peut avoir été réaffectée plus d'une fois et un même ensemble de joueurs peut être affecté par plus d'une opération. Prenant en compte ces conditions, il est possible que le pourcentage dépasse 100%, puisque la valeur est calculée sur l'agrégat des données par seconde et non sur la base de l'unicité des entités. Cela signifie également que ce pourcentage n'est pas très précis, mais contribue tout de même à donner une bonne estimation de l'impact des opérations de rebalancement.

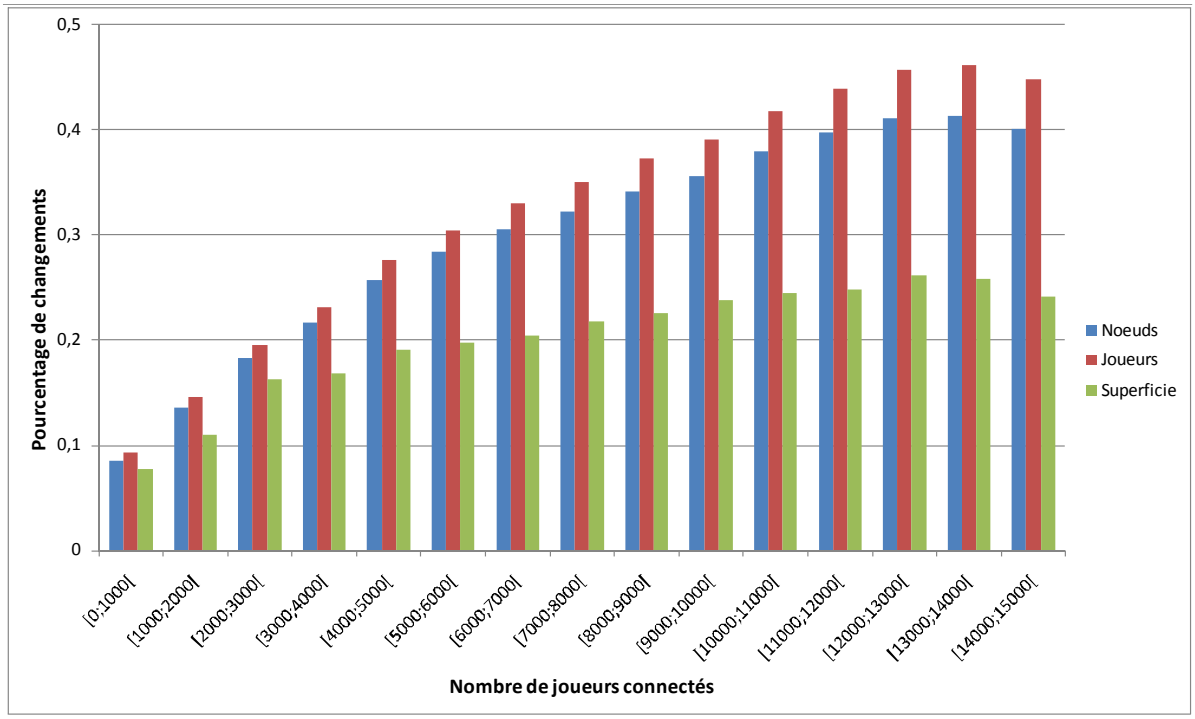
En ce qui concerne le premier graphique, nous remarquons que c'est très similaire pour les simulations 1, 5 et 6 puisque les capacités de charge sont les mêmes. Même si la simulation 6 a pour différence qu'il n'y a pas de région d'intérêt, il semble y avoir tout de même le même nombre de zones par rapport au nombre de joueurs tout au long de la simulation, quoique que ces zones sont réparties plus uniformément. La simulation 3 possède environ $2,5$ fois plus de zones par rapport au nombre de joueurs, ce qui s'explique par une diminution de moitié de la capacité de charge des noeuds. De plus, à plusieurs moments, la différence entre le nombre de zones et le nombre de joueurs est faible, ce qui indique que le système possède souvent beaucoup de zones pour peu de joueurs, une situation qui n'est pas adéquate puisque cela augmente considérablement les coûts de gestion des serveurs de zones, tel que discuté à la section 4.3.1. La simulation 4 amplifie davantage le phénomène en augmentant de façon drastique le nombre de zones par rapport au nombre de joueurs. À certains moments, il y a plus de de zones que de joueurs, une situation qui ne serait pas possible dans la réalité puisque cela viole l'hypothèse selon laquelle une seule zone est assignée à chaque noeud serveur. De plus, conceptuellement, cette situation est nettement moins performante que le modèle client-serveur classique. Ce phénomène s'explique encore une fois par le nombre réduit de régions d'intérêt, favorisant une multitude de régions surpeuplées qui sont redécoupées continuellement. En regardant les résultats de la simulation 2, nous remarquons une baisse

marquée de la consommation de charge selon le modèle pair-à-pair. La charge serveur semble être réduite de plus de la moitié, ce qui démontre que disposer de noeuds de capacité maximale doublée a un impact direct sur la charge imputée au serveur central.

En ce qui a trait au second graphique, nous remarquons à première vue que les résultats de la simulation 1 sont significativement différents des autres simulations. Dans cette simulation, le pourcentage de noeuds, joueurs et de superficie affectés par un rebalancement croît selon le nombre de joueurs connectés. De plus, les valeurs (pourcentages) sont moins élevées que pour les autres simulations. Cela s'explique encore ici par le temps de liste noire (t_b) augmenté, faisant en sorte qu'un minimum de 30 secondes doit s'écouler avant qu'une même zone ne soit rebalancée, réduisant les possibilités de rebalancement. Tel que discuté, ce n'est toutefois pas souhaitable puisque certaines zones pourraient devenir en surcharge sans possibilité de rebalancement jusqu'à l'écoulement du délai. Dans les autres simulations, la situation est plutôt similaire puisque le nombre de rebalancements par seconde est borné jusqu'à un maximum (paramètres n_r^{high} et n_r^{low} décrits à la section 3.5.4). Lorsque ces bornes sont atteintes pour une unité de temps donné, aucun autre rebalancement n'est permis pour cette même unité de temps. De plus, plus il y a de joueurs connectés, plus les zones sont de taille réduite. Par conséquent, un rebalancement d'une zone donnée affectera moins de joueurs et de superficie, ce qui explique pourquoi ces valeurs décroissent lorsque le nombre de joueurs connecté est important. Encore ici, la puissance accrue des noeuds pour la simulation 2 réduit de façon significative le volume d'opérations de rebalancement nécessaires.

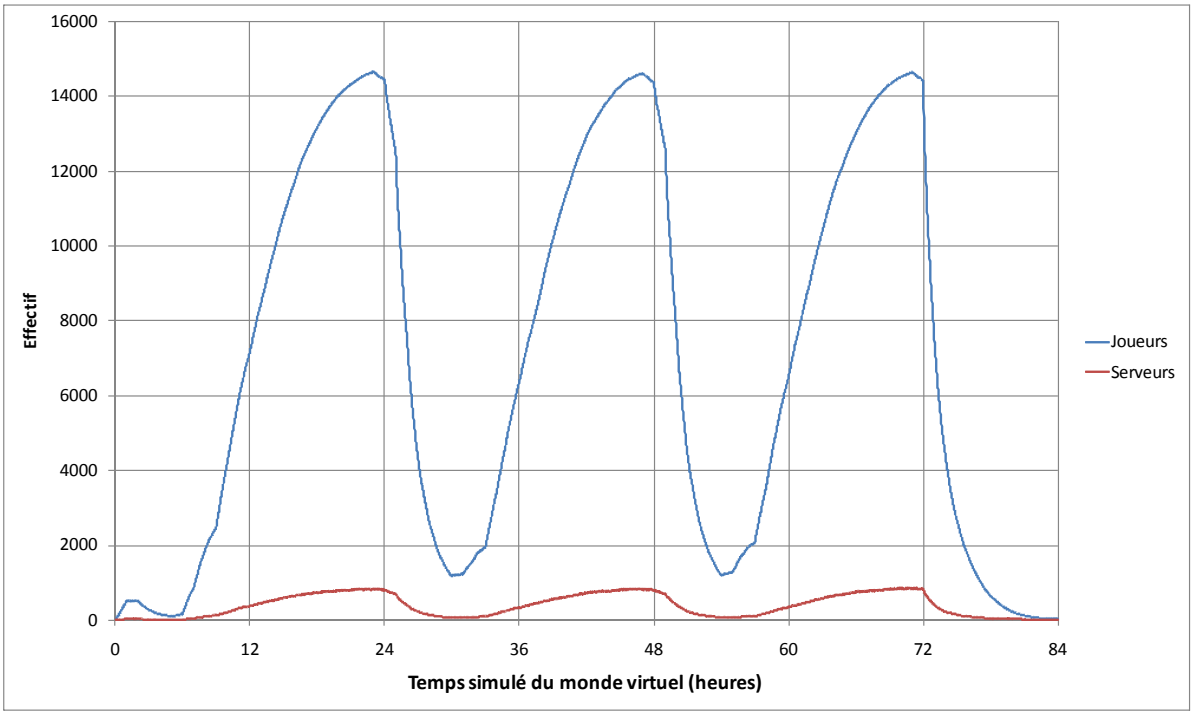


(a) Nombre de joueurs et de serveurs (zones) en fonction du temps

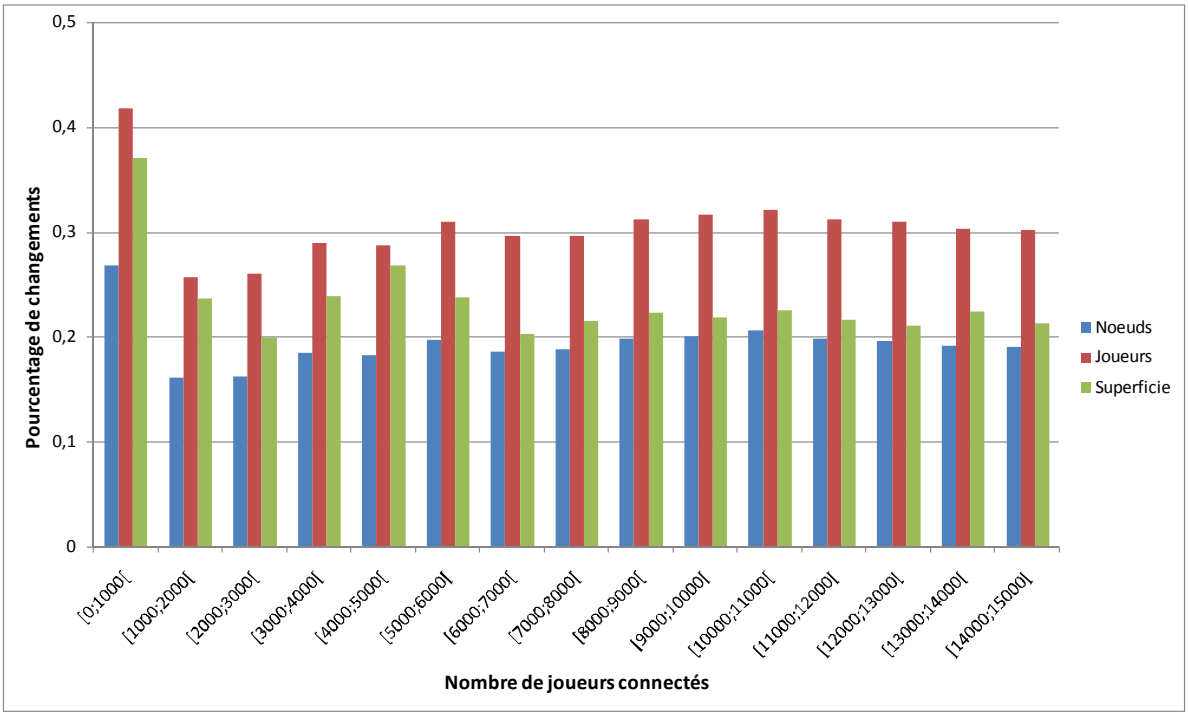


(b) Pourcentage moyen de noeuds serveurs, de joueurs, et de superficie du territoire qui ont été réaffectés ou modifiés par minute de jeu en fonction du nombre de joueurs

FIGURE 4.13 Analyse des rebalancements pour la simulation 1

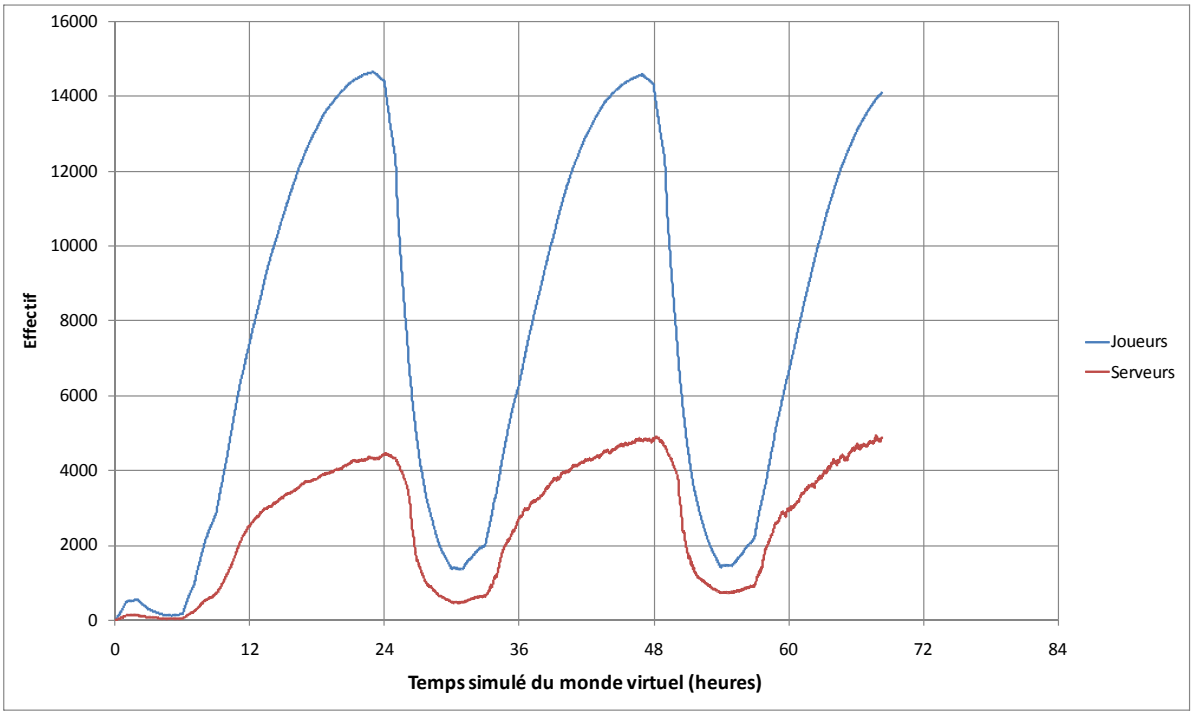


(a) Nombre de joueurs et de serveurs (zones) en fonction du temps

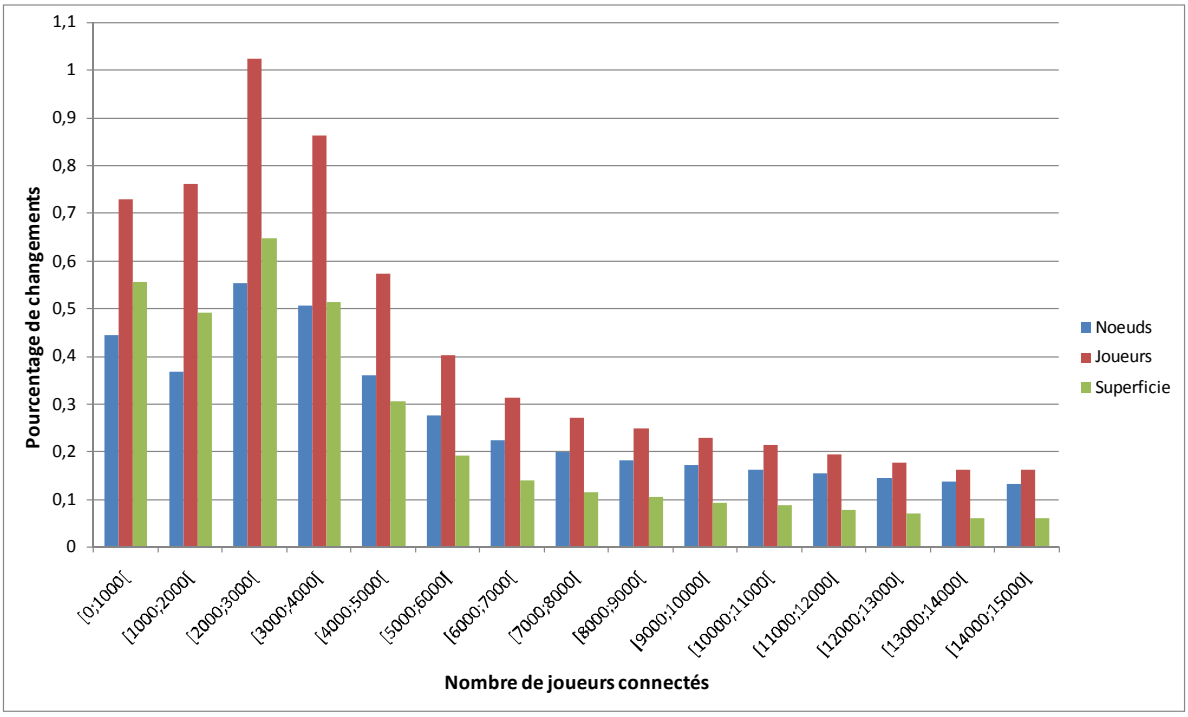


(b) Pourcentage moyen de noeuds serveurs, de joueurs, et de superficie du territoire qui ont été réaffectés ou modifiés par minute de jeu en fonction du nombre de joueurs

FIGURE 4.14 Analyse des rebalancements pour la simulation 2

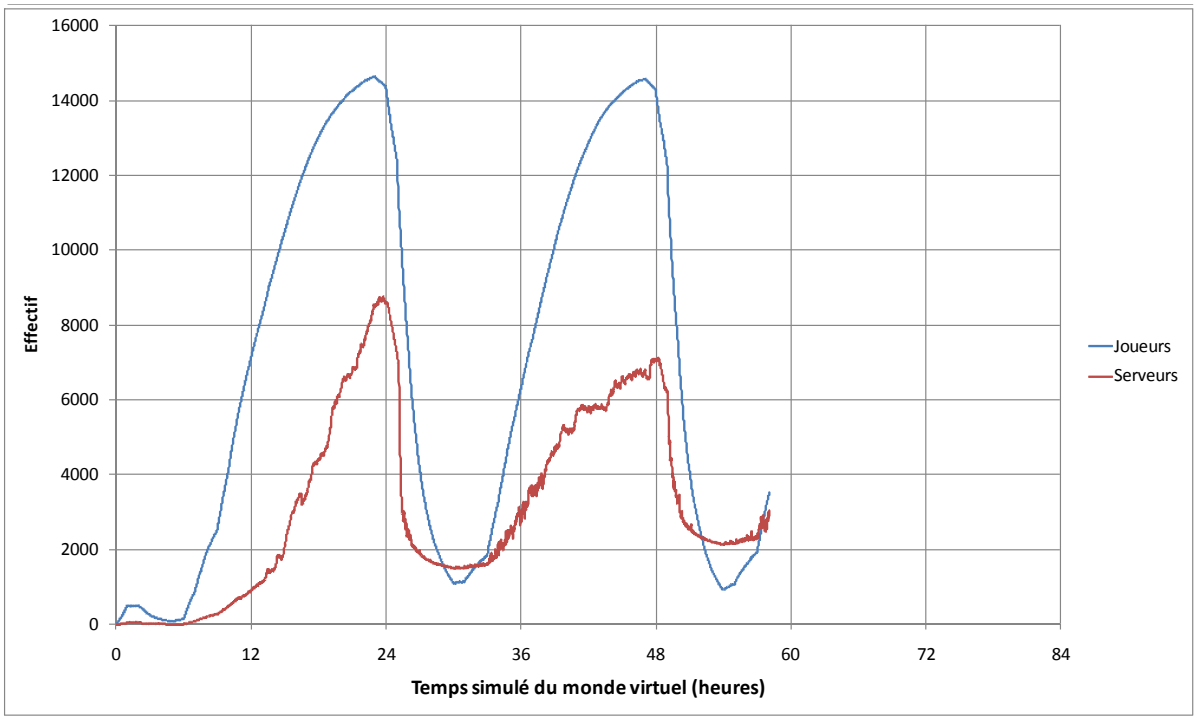


(a) Nombre de joueurs et de serveurs (zones) en fonction du temps

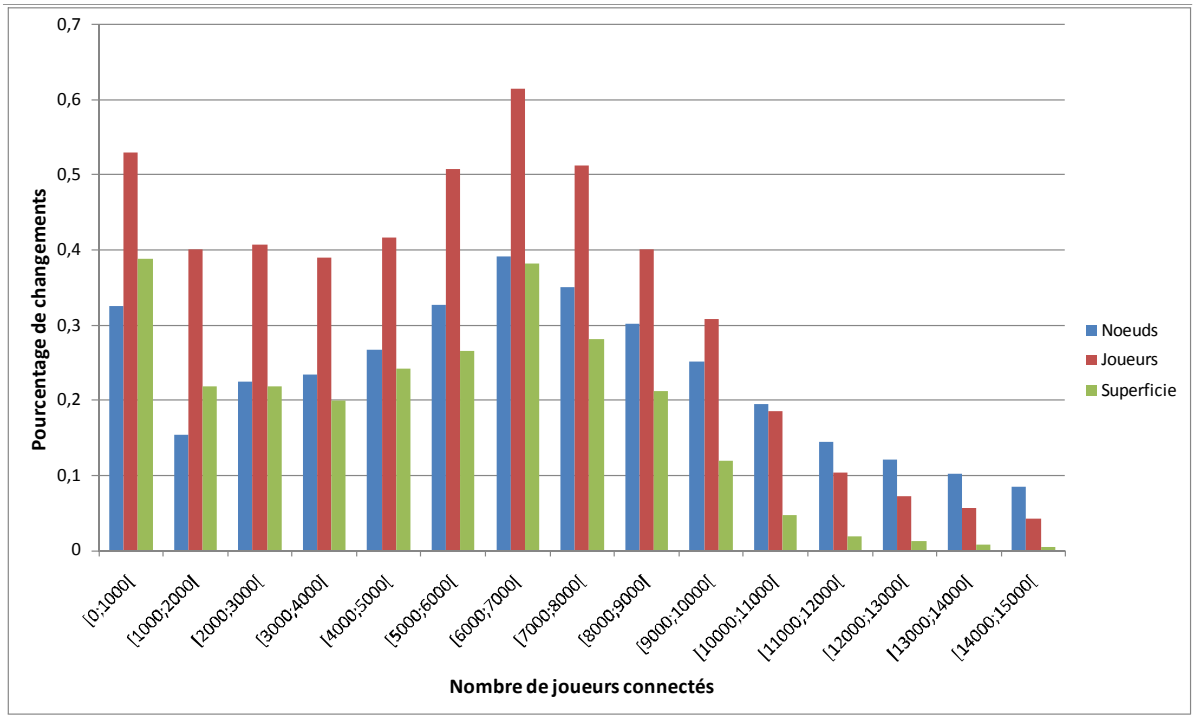


(b) Pourcentage moyen de noeuds serveurs, de joueurs, et de superficie du territoire qui ont été réaffectés ou modifiés par minute de jeu en fonction du nombre de joueurs

FIGURE 4.15 Analyse des rebalancements pour la simulation 3

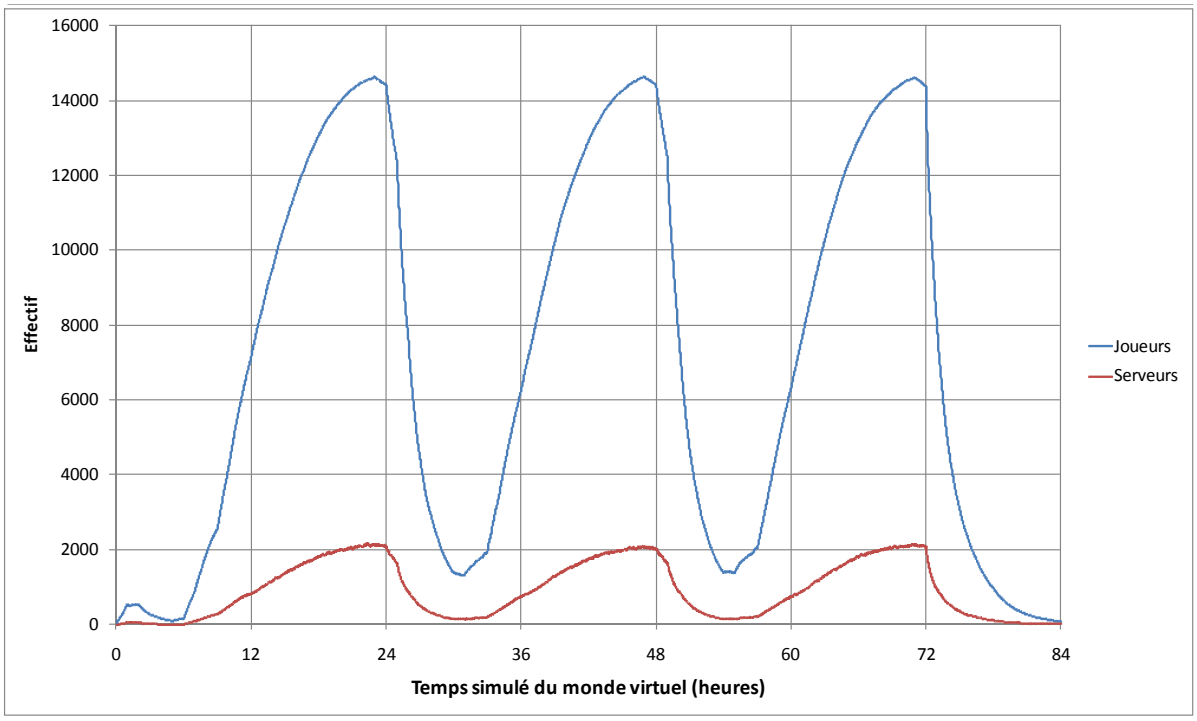


(a) Nombre de joueurs et de serveurs (zones) en fonction du temps

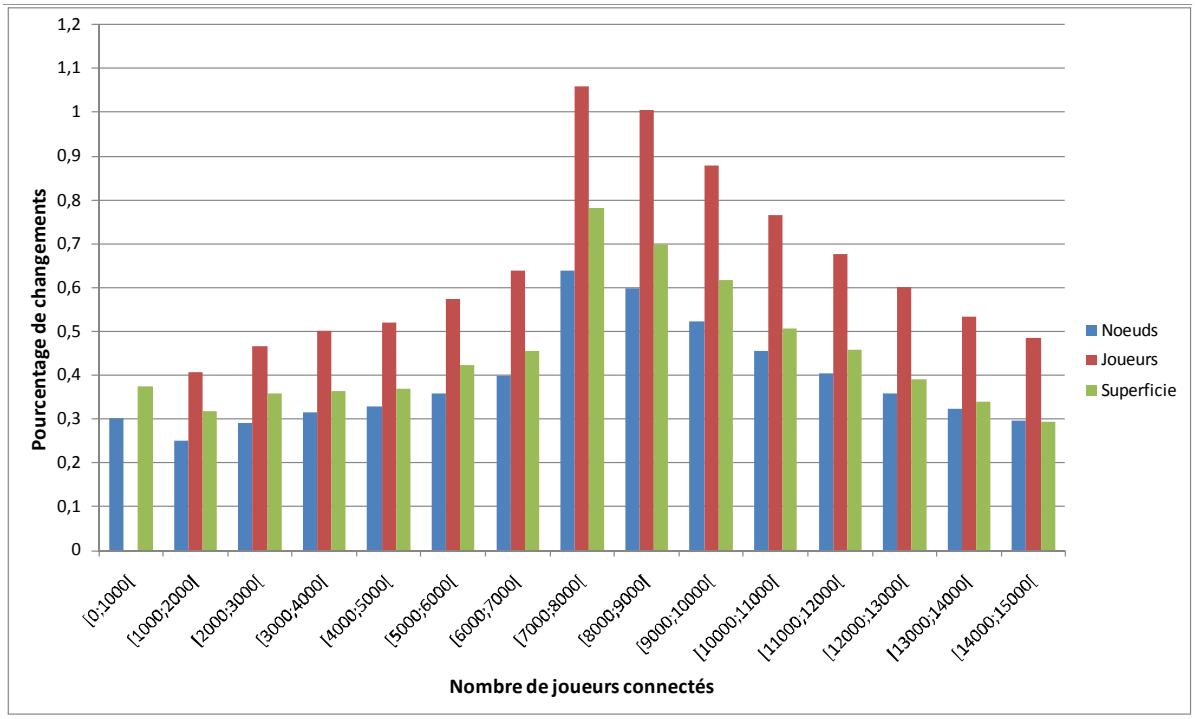


(b) Pourcentage moyen de noeuds serveurs, de joueurs, et de superficie du territoire qui ont été réaffectés ou modifiés par minute de jeu en fonction du nombre de joueurs

FIGURE 4.16 Analyse des rebalancements pour la simulation 4

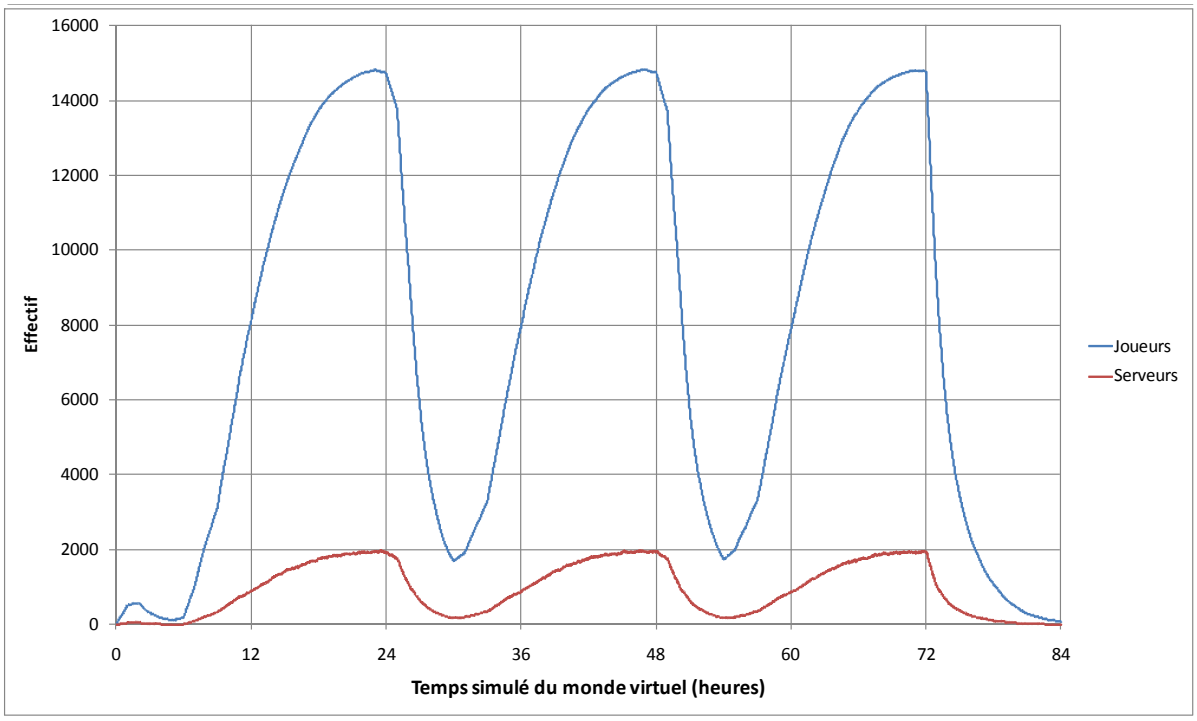


(a) Nombre de joueurs et de serveurs (zones) en fonction du temps

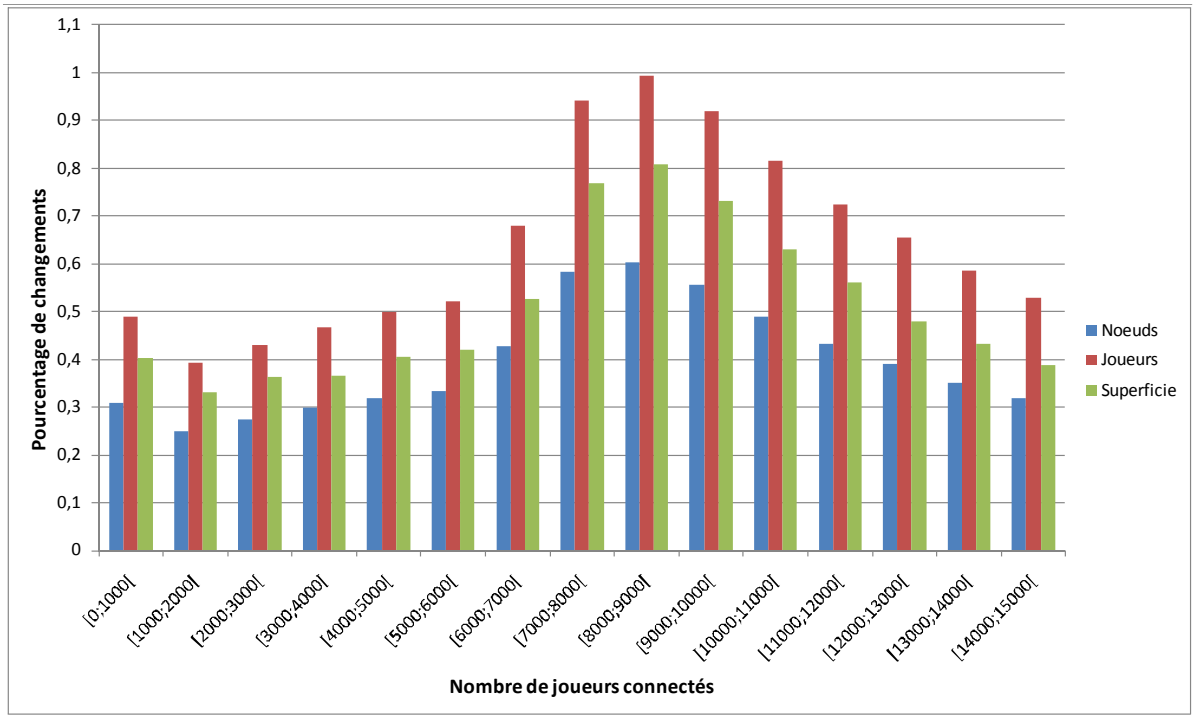


(b) Pourcentage moyen de noeuds serveurs, de joueurs, et de superficie du territoire qui ont été réaffectés ou modifiés par minute de jeu en fonction du nombre de joueurs

FIGURE 4.17 Analyse des rebalancements pour la simulation 5



(a) Nombre de joueurs et de serveurs (zones) en fonction du temps



(b) Pourcentage moyen de noeuds serveurs, de joueurs, et de superficie du territoire qui ont été réaffectés ou modifiés par minute de jeu en fonction du nombre de joueurs

FIGURE 4.18 Analyse des rebalancements pour la simulation 6

4.3.3 Analyse de la charge

Cette analyse porte sur le ratio de charge des différentes zones du jeu. Puisque l'univers virtuel comporte beaucoup de zones, il n'est évidemment pas possible d'analyser le ratio de charge pour chaque zone sur une base individuelle. Nous avons donc décidé de regrouper les zones selon différentes classes de ratio de charge : 0% à 20% ([0-20[), 20% à 40% ([20-40[), 40% à 60% ([40-60[), 60% à 80% ([60-80[), 80% à 100% ([80-100[) et 100% et plus ([100-∞[). Tel qu'énoncé à la section 3.4.5, un ratio de charge supérieur à 100% est problématique puisque l'on peut potentiellement excéder la limite de consommation de ressources fixée initialement. Selon les politiques établies pour le jeu, dépasser la limite peut être critique si la limite de consommation de ressources fixée équivaut à la capacité réelle disponible sur la machine. Si la limite fixée est en-déça de la capacité réelle de la machine, dépasser peut être moins critique. Cependant, nous considérons que dépasser la limite fixée est une sitation très défavorable qui doit se produire le moins souvent possible. À l'inverse, tout ratio de charge inférieur à 100% ne pose aucun problème et est parfaitement acceptable.

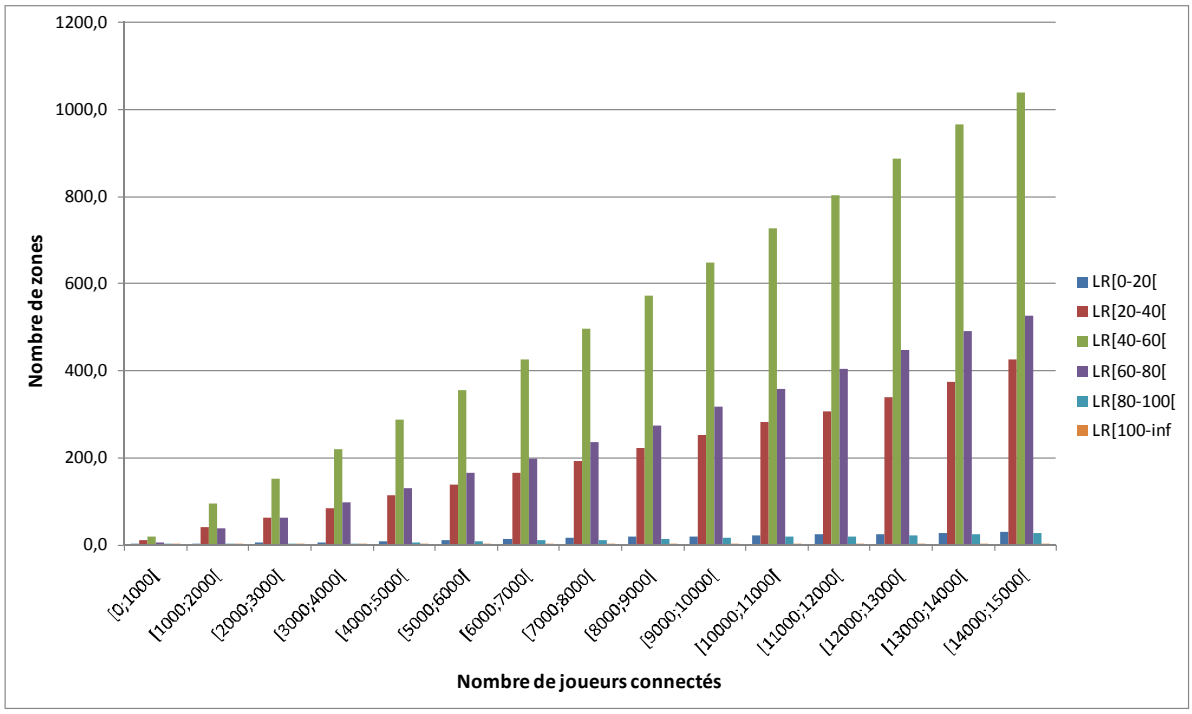
Les analyses de charge pour les six simulations sont présentées aux figures 4.19, 4.20, 4.21, 4.22, 4.23 et 4.24. Les analyses sont présentées à la fois sous la forme d'un graphique et à la fois sous la forme d'un tableau. Les données ont été regroupées selon le nombre de joueurs. Ainsi, pour chaque classe de 1000 joueurs, la vue graphique illustre le nombre de zones pour chaque classe de ratio de charge. Le tableau illustre ces mêmes données, mais en pourcentages (le pourcentage étant calculé à partir du nombre de zones pour chaque catégorie sur le nombre de zones total moyen pour la classe de 1000 joueurs).

Les simulations 2, 5 et 6 illustrent des conditions parfaites au niveau de la répartition de la charge ("load balancing"), selon les critères d'analyse définis. En effet, il y a toujours aucune zone en surcharge, peu importe le nombre de joueurs. Cela indique que l'algorithme de rebalancement permet de maintenir continuellement la charge en dessous d'un seuil donné, peu importe les variations en cours de jeu, pour les paramètres propres à ces simulations. Rappelons que pour ces simulations, les noeuds ont un ratio de charge dit "standard" (ou "moyen"). Le temps de liste noire (t_b) réduit par rapport à la simulation 1 a probablement également un effet positif puisque les zones devenant en surcharge peuvent être rebalancées plus rapidement. La simulation 6 n'a aucune région d'intérêt alors que la simulation 5 en a un nombre "standard", mais cela ne semble pas avoir causé obstacle au bon fonctionnement de l'algorithme. La simulation 2 a également un nombre "standard" de régions d'intérêts et possède également des noeuds de capacité maximale doublée. Mentionnons également pour ces simulations que le pourcentage de zones possédant un ratio entre 80% et 100% (toutes classes de joueurs confondues) est assez faible, voire même presque négligeable pour la simulation 2. Même si avoir des zones dans ce ratio de charge est parfaitement acceptable,

cela indique que peu de zones sont sujettes à passer la limite, ce qui renforce la stabilité. Du point de vue de la charge imposée aux noeuds, les paramètres assignés aux simulations 5 et 6 assurent un fonctionnement très adéquat et les paramètres assignés à la simulation 2 assurent un fonctionnement quasi-parfait.

Les résultats de la simulation 1 sont assez prometteurs, mais il y a tout de même un très faible pourcentage de zones avec un ratio de charge supérieur à 100%. Sans être critique, cette situation est néammoins à éviter dans la mesure du possible. Les paramètres de la simulation 1 sont les mêmes que ceux de la simulation 5, à l'exception du temps de liste noire qui est de 30 secondes au lieu de 10 secondes. Nous tirons la conclusion que ce long délai empêche peut-être certaines zones surchargées d'être rebalancées dans les temps voulus.

La simulation 3 (capacité de charge maximale des noeuds réduite de moitié) est problématique puisqu'un nombre important de zones se retrouvent en surcharge pour la majeure partie de la simulation. Le problème s'amplifie de façon draconienne lorsqu'il y a augmentation du nombre de joueurs connectés, rendant le fonctionnement adéquat du modèle impossible dans ces conditions. La simulation 4, qui possède un nombre réduit de régions d'intérêt très densément peuplées, est également problématique quoique dans une moindre mesure que la simulation 3. Cependant, les perturbations sont suffisantes pour nuire au bon déroulement du jeu dans ces conditions.

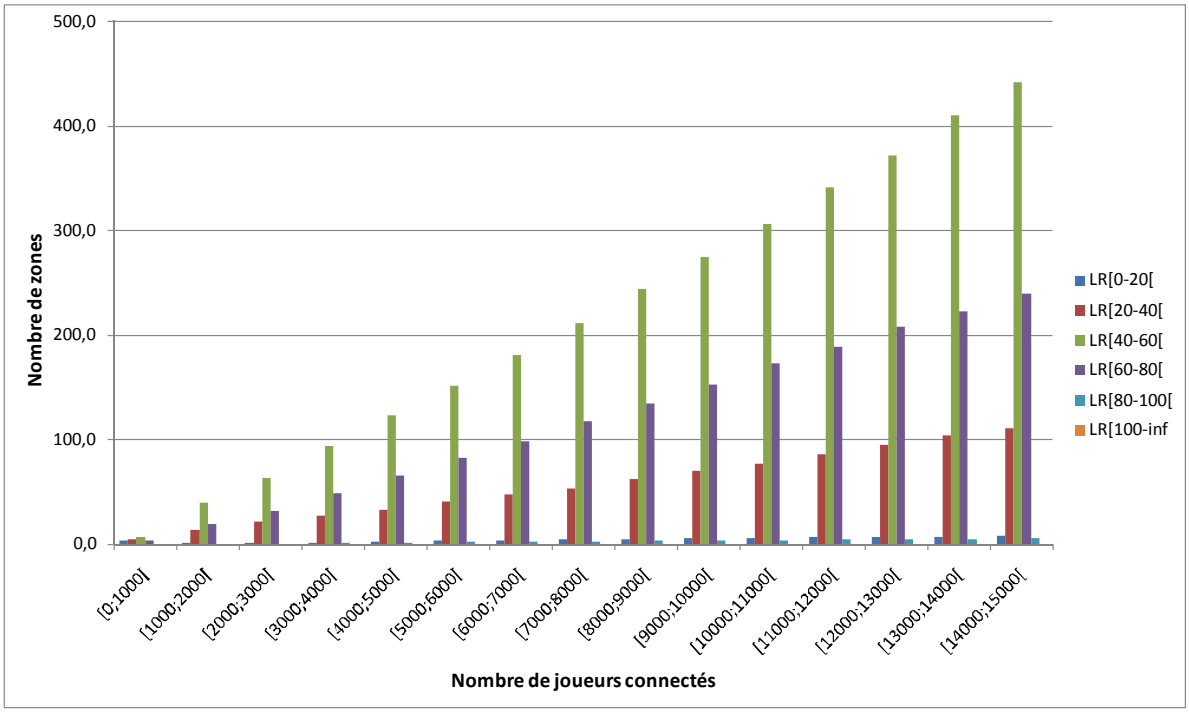


(a) Nombre moyen de zones par classe de ratio de charge en fonction du nombre de joueurs

	[0-20[	[20-40[	[40-60[	[60-80[	[80-100[	[100-∞[
[0; 1000[	10,3%	25,8%	46,5%	16,9%	0,4%	0,1%
[1000; 2000[	1,9%	22,8%	53,5%	21,0%	0,6%	0,1%
[2000; 3000[	1,5%	21,8%	53,5%	22,2%	0,8%	0,1%
[3000; 4000[	1,6%	20,6%	53,1%	23,5%	0,9%	0,1%
[4000; 5000[	1,6%	21,1%	52,4%	23,6%	1,0%	0,2%
[5000; 6000[	1,4%	20,4%	52,4%	24,3%	1,1%	0,2%
[6000; 7000[	1,6%	20,2%	52,3%	24,4%	1,2%	0,2%
[7000; 8000[	1,6%	20,2%	51,9%	24,7%	1,3%	0,2%
[8000; 9000[	1,6%	20,2%	51,6%	24,8%	1,3%	0,2%
[9000; 10000[	1,5%	20,0%	51,6%	25,2%	1,3%	0,2%
[10000; 11000[	1,5%	20,0%	51,5%	25,4%	1,3%	0,2%
[11000; 12000[	1,5%	19,6%	51,4%	25,8%	1,3%	0,1%
[12000; 13000[	1,5%	19,7%	51,4%	25,9%	1,3%	0,1%
[13000; 14000[	1,4%	19,9%	51,1%	26,0%	1,3%	0,1%
[14000; 15000[	1,4%	20,8%	50,6%	25,6%	1,4%	0,1%

(b) Nombre moyen de zones par classe de ratio de charge en fonction du nombre de joueurs (pourcentage)

FIGURE 4.19 Analyse de la charge pour la simulation 1

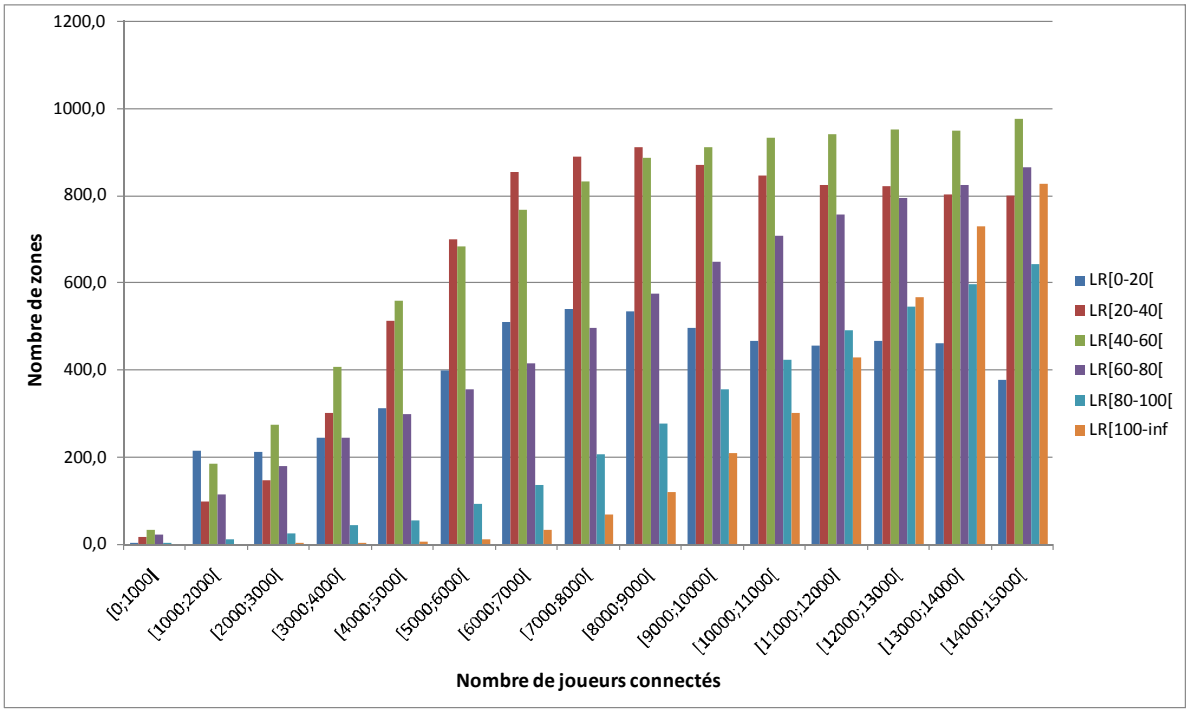


(a) Nombre moyen de zones par classe de ratio de charge en fonction du nombre de joueurs

	[0-20[	[20-40[	[40-60[	[60-80[	[80-100[	[100-∞[
[0; 1000[	21,9%	22,3%	36,8%	18,2%	0,8%	0,0%
[1000; 2000[	2,0%	18,7%	52,9%	25,7%	0,6%	0,0%
[2000; 3000[	1,4%	17,9%	53,3%	26,8%	0,5%	0,0%
[3000; 4000[	0,9%	15,9%	54,5%	28,0%	0,6%	0,0%
[4000; 5000[	0,9%	14,7%	54,6%	29,0%	0,7%	0,0%
[5000; 6000[	1,0%	14,6%	54,2%	29,3%	0,7%	0,0%
[6000; 7000[	1,0%	14,4%	54,3%	29,5%	0,6%	0,0%
[7000; 8000[	1,1%	13,8%	54,3%	30,0%	0,7%	0,0%
[8000; 9000[	1,0%	13,9%	54,3%	29,9%	0,7%	0,0%
[9000; 10000[	1,0%	13,8%	54,2%	30,2%	0,7%	0,0%
[10000; 11000[	1,1%	13,6%	54,0%	30,5%	0,7%	0,0%
[11000; 12000[	1,0%	13,7%	54,3%	30,1%	0,7%	0,0%
[12000; 13000[	1,0%	13,7%	54,1%	30,3%	0,7%	0,0%
[13000; 14000[	0,9%	13,8%	54,7%	29,7%	0,7%	0,0%
[14000; 15000[	1,0%	13,7%	54,7%	29,8%	0,7%	0,0%

(b) Nombre moyen de zones par classe de ratio de charge en fonction du nombre de joueurs (pourcentage)

FIGURE 4.20 Analyse de la charge pour la simulation 2

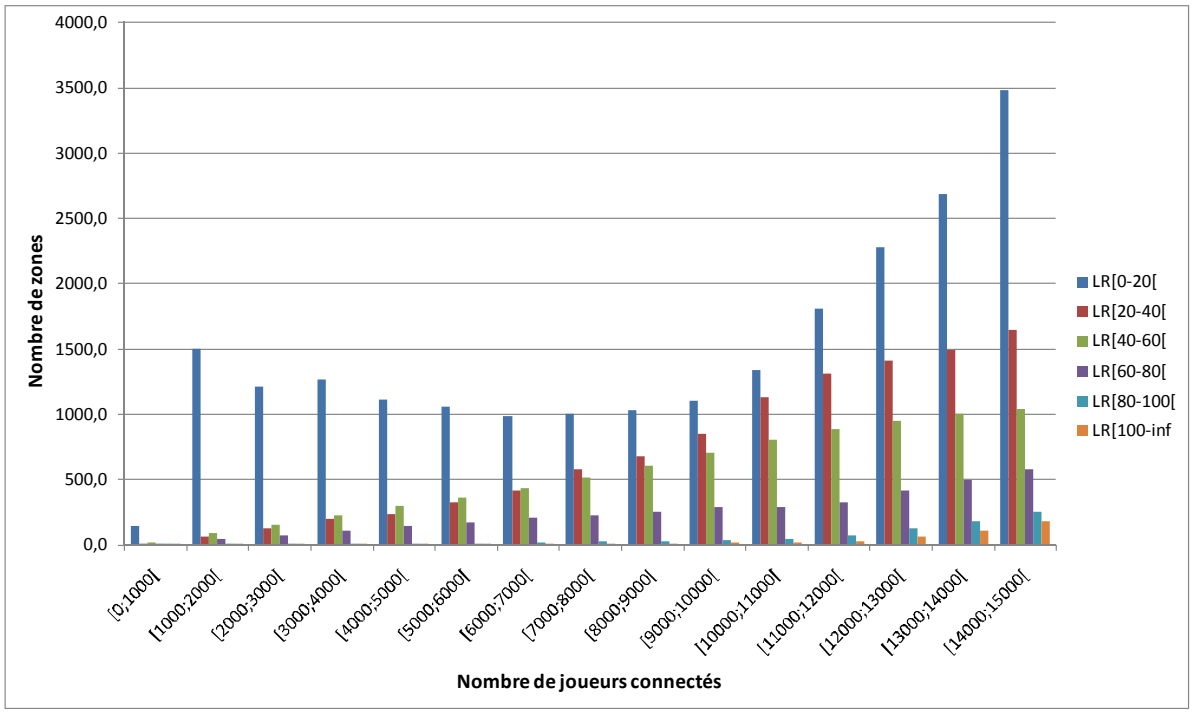


(a) Nombre moyen de zones par classe de ratio de charge en fonction du nombre de joueurs

	[0-20[	[20-40[	[40-60[	[60-80[	[80-100[	[100-∞[
[0; 1000[	4,4%	20,6%	43,4%	28,8%	2,2%	0,1%
[1000; 2000[	34,5%	15,7%	29,7%	18,3%	1,4%	0,1%
[2000; 3000[	25,3%	17,4%	32,7%	21,4%	2,7%	0,2%
[3000; 4000[	19,7%	24,1%	32,7%	19,5%	3,4%	0,3%
[4000; 5000[	17,9%	29,4%	32,1%	17,1%	3,1%	0,3%
[5000; 6000[	17,7%	31,2%	30,4%	15,9%	4,0%	0,5%
[6000; 7000[	18,8%	31,4%	28,2%	15,2%	4,9%	1,2%
[7000; 8000[	17,8%	29,3%	27,4%	16,3%	6,8%	2,3%
[8000; 9000[	16,2%	27,6%	26,8%	17,4%	8,3%	3,6%
[9000; 10000[	14,2%	24,9%	26,1%	18,5%	10,2%	6,0%
[10000; 11000[	12,7%	22,9%	25,3%	19,2%	11,5%	8,2%
[11000; 12000[	11,7%	21,1%	24,1%	19,4%	12,5%	11,0%
[12000; 13000[	11,2%	19,8%	22,9%	19,2%	13,1%	13,7%
[13000; 14000[	10,6%	18,4%	21,7%	18,9%	13,7%	16,7%
[14000; 15000[	8,4%	17,8%	21,7%	19,3%	14,3%	18,4%

(b) Nombre moyen de zones par classe de ratio de charge en fonction du nombre de joueurs (pourcentage)

FIGURE 4.21 Analyse de la charge pour la simulation 3

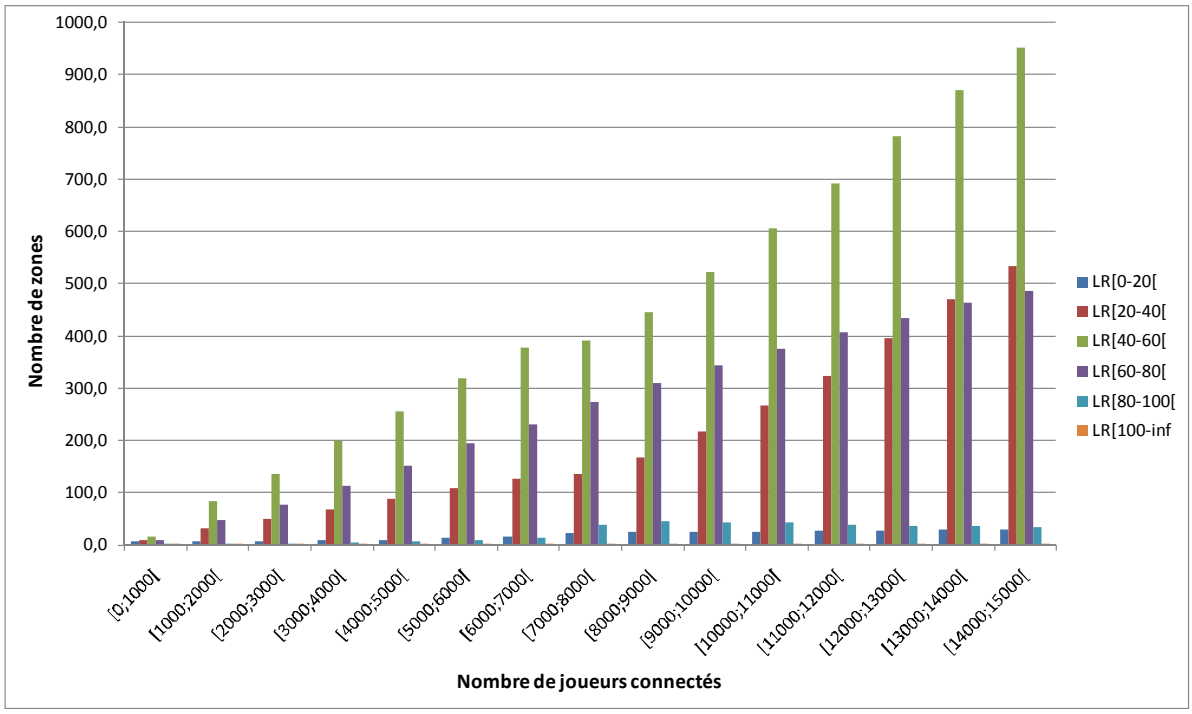


(a) Nombre moyen de zones par classe de ratio de charge en fonction du nombre de joueurs

	[0-20[	[20-40[	[40-60[	[60-80[	[80-100[	[100-∞[
[0; 1000[	78,7%	4,8%	10,1%	6,0%	0,3%	0,1%
[1000; 2000[	88,5%	3,5%	5,1%	2,6%	0,1%	0,1%
[2000; 3000[	76,4%	8,3%	9,9%	4,8%	0,3%	0,3%
[3000; 4000[	69,6%	11,2%	12,6%	5,9%	0,3%	0,3%
[4000; 5000[	61,7%	13,1%	16,6%	7,9%	0,4%	0,3%
[5000; 6000[	54,7%	16,8%	18,7%	8,8%	0,6%	0,3%
[6000; 7000[	47,6%	20,2%	20,7%	9,8%	1,1%	0,4%
[7000; 8000[	42,4%	24,4%	21,9%	9,5%	1,2%	0,4%
[8000; 9000[	39,5%	26,1%	23,1%	9,6%	1,1%	0,4%
[9000; 10000[	36,9%	28,3%	23,4%	9,5%	1,2%	0,5%
[10000; 11000[	36,7%	31,1%	22,2%	8,1%	1,2%	0,6%
[11000; 12000[	40,7%	29,5%	20,0%	7,4%	1,5%	0,7%
[12000; 13000[	43,4%	26,8%	18,1%	7,9%	2,4%	1,3%
[13000; 14000[	44,9%	25,0%	16,8%	8,3%	3,0%	1,8%
[14000; 15000[	48,5%	22,9%	14,5%	8,0%	3,5%	2,6%

(b) Nombre moyen de zones par classe de ratio de charge en fonction du nombre de joueurs (pourcentage)

FIGURE 4.22 Analyse de la charge pour la simulation 4

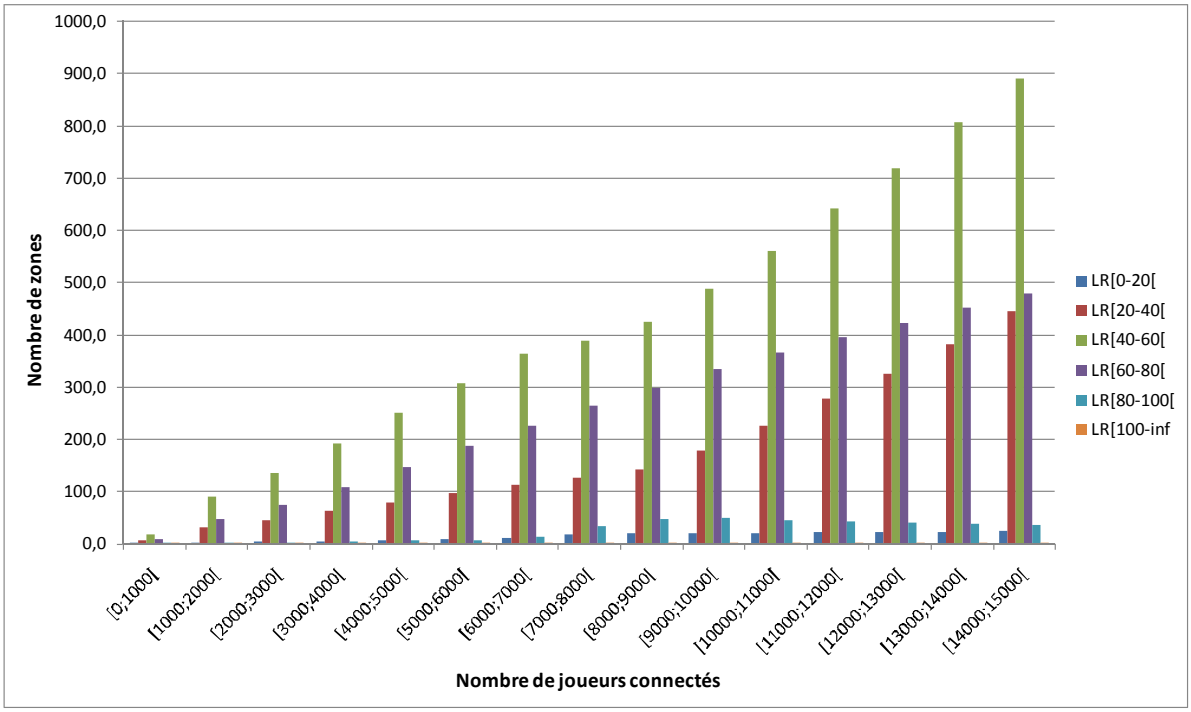


(a) Nombre moyen de zones par classe de ratio de charge en fonction du nombre de joueurs

	[0-20[	[20-40[	[40-60[	[60-80[	[80-100[	[100-∞[
[0; 1000[	16,0%	20,0%	40,3%	22,7%	0,9%	0,0%
[1000; 2000[	4,0%	18,7%	48,8%	27,2%	1,0%	0,0%
[2000; 3000[	2,6%	18,1%	49,8%	28,2%	1,1%	0,0%
[3000; 4000[	2,3%	17,2%	50,4%	28,6%	1,2%	0,0%
[4000; 5000[	2,0%	17,2%	49,7%	29,5%	1,3%	0,0%
[5000; 6000[	2,0%	17,0%	49,4%	30,0%	1,4%	0,0%
[6000; 7000[	2,0%	16,5%	49,3%	30,2%	1,7%	0,0%
[7000; 8000[	2,5%	15,7%	45,3%	31,5%	4,4%	0,0%
[8000; 9000[	2,4%	16,8%	44,6%	31,1%	4,5%	0,0%
[9000; 10000[	2,2%	18,7%	45,1%	29,8%	3,8%	0,0%
[10000; 11000[	1,9%	20,2%	45,8%	28,4%	3,2%	0,0%
[11000; 12000[	1,8%	21,7%	46,3%	27,2%	2,6%	0,0%
[12000; 13000[	1,6%	23,5%	46,5%	25,9%	2,2%	0,0%
[13000; 14000[	1,5%	25,1%	46,5%	24,7%	1,9%	0,0%
[14000; 15000[	1,4%	26,2%	46,6%	23,9%	1,7%	0,0%

(b) Nombre moyen de zones par classe de ratio de charge en fonction du nombre de joueurs (pourcentage)

FIGURE 4.23 Analyse de la charge pour la simulation 5



(a) Nombre moyen de zones par classe de ratio de charge en fonction du nombre de joueurs

	[0-20[	[20-40[	[40-60[	[60-80[	[80-100[	[100-∞[
[0; 1000[	2,4%	21,9%	50,9%	23,7%	0,8%	0,0%
[1000; 2000[	1,5%	18,2%	52,0%	27,2%	0,9%	0,0%
[2000; 3000[	1,4%	17,0%	51,7%	28,6%	1,1%	0,0%
[3000; 4000[	1,3%	16,8%	51,6%	28,9%	1,1%	0,0%
[4000; 5000[	1,4%	16,3%	51,0%	29,9%	1,2%	0,0%
[5000; 6000[	1,4%	15,9%	50,5%	30,6%	1,3%	0,0%
[6000; 7000[	1,6%	15,6%	49,8%	30,8%	1,8%	0,0%
[7000; 8000[	2,1%	15,1%	46,6%	31,7%	4,0%	0,0%
[8000; 9000[	2,1%	15,2%	45,3%	31,8%	5,0%	0,0%
[9000; 10000[	2,0%	16,5%	45,3%	31,0%	4,7%	0,0%
[10000; 11000[	1,7%	18,5%	45,8%	29,8%	3,8%	0,0%
[11000; 12000[	1,6%	20,1%	46,4%	28,5%	3,1%	0,0%
[12000; 13000[	1,5%	21,2%	46,8%	27,5%	2,7%	0,0%
[13000; 14000[	1,3%	22,3%	47,2%	26,5%	2,3%	0,0%
[14000; 15000[	1,3%	23,7%	47,3%	25,5%	2,0%	0,0%

(b) Nombre moyen de zones par classe de ratio de charge en fonction du nombre de joueurs (pourcentage)

FIGURE 4.24 Analyse de la charge pour la simulation 6

4.3.4 Analyse de la latence

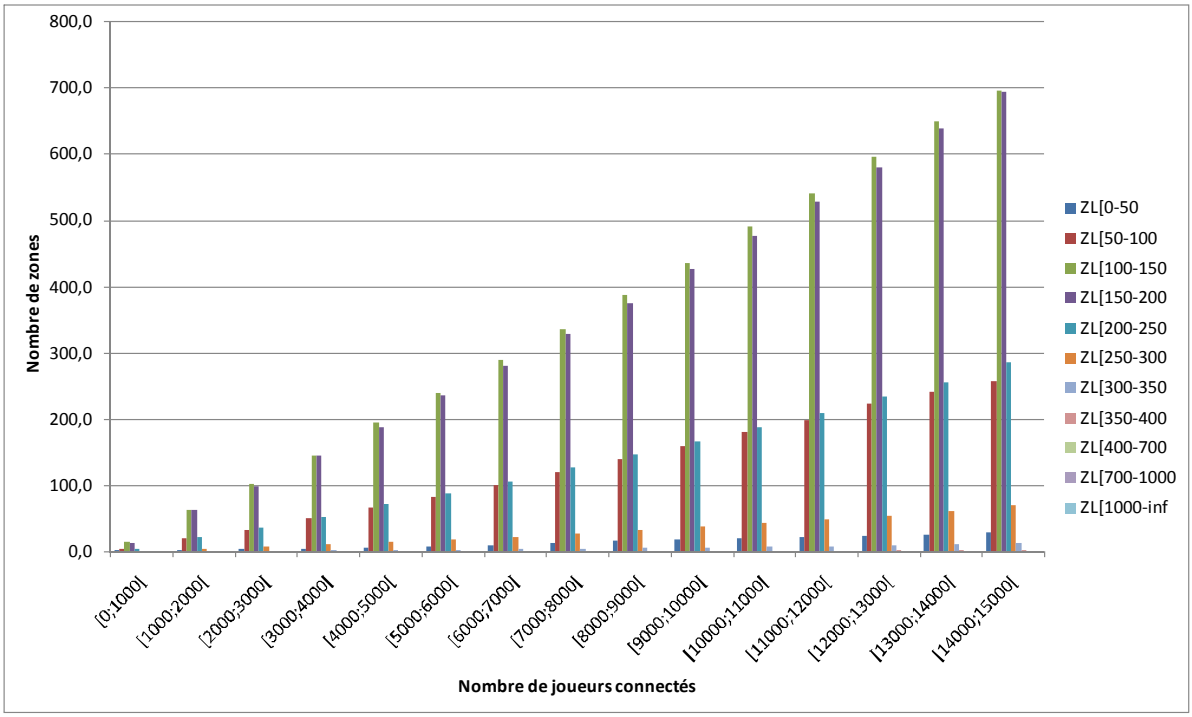
L'analyse de la latence vise à jeter un coup d'oeil aux différentes valeurs de latence entre les noeuds responsables des différentes zones et les joueurs sous leur responsabilité. À cette fin, la notion de latence moyenne (section 3.4.6) est utilisée. Pour chaque zone, il s'agit de calculer la latence moyenne entre le joueur responsable de la zone et l'ensemble des joueurs présents dans cette zone. Plus la latence tend à diminuer, meilleure sera la qualité du jeu. Typiquement, une valeur de latence de moins de 700 millisecondes est acceptable, et une valeur de latence de moins de 400 millisecondes est bonne. Inversement, une valeur de latence supérieure à 700 millisecondes cause certains problèmes de jouabilité et une valeur de latence supérieure à 1000 millisecondes rend le jeu très difficile à jouer. Le concept de latence est utilisé lors des opérations de rebalancement afin de sélectionner, parmi les noeuds aptes à devenir responsables de la zone, le noeud qui minimisera la latence moyenne.

La présentation des figures est analogue à la présentation des figures pour l'analyse de charge. Les analyses de latence pour les six simulations sont présentées aux figures 4.25, 4.26, 4.27, 4.28, 4.29 et 4.30. Les analyses sont présentées à la fois sous la forme d'un graphique et à la fois sous la forme d'un tableau. Les données ont été regroupées selon le nombre de joueurs. Ainsi, pour chaque classe de 1000 joueurs, la vue graphique illustre le nombre de zones pour chaque classe de latence. Le tableau illustre ces mêmes données, mais en pourcentages (le pourcentage étant calculé à partir du nombre de zones pour chaque catégorie sur le nombre de zones total moyen pour la classe de 1000 joueurs).

L'ensemble des latences moyennes des simulations 1, 2 et 6 sont inférieures à 400 millisecondes et ce, peu importe le nombre de joueurs connectés. Rappelons que les simulations 1 et 6 ont des noeuds de capacité maximale "standard" et que la simulation 2 possède des noeuds de capacité doublée. Les simulations 1 et 2 contiennent des régions d'intérêt, mais pas la simulation 6. La simulation 5, qui possède les mêmes paramètres que la simulation mais avec un temps de liste noire t_b plus court, donne également de très bons résultats, avec seulement 0,1% des zones possédant une latence entre 400 et 700 millisecondes pour un nombre de joueurs supérieur à 13000 uniquemnt. Ce pourcentage est toutefois très faible ; par conséquent, il serait prématuré de s'y fier pour conclure que le temps t_b réduit a un impact sur la latence. Qui plus est, les valeurs de latence moyennes pour les autres classes de joueurs sont très similaires à celles de la simulation 1 et 6. Les simulations 3 (capacité maximale des noeuds réduite de moitié) et 4 (très peu de régions d'intérêts, mais très densément peuplées) produisent des valeurs de latence moyennes légèrement moins bonnes. En effet, il y a un pourcentage un peu plus important de zones dans la plage 400-700 millisecondes et également quelques zones dans la plage 700-1000 millisecondes (quoique très faible). Ces deux simulations montrent que la latence moyenne est légèrement moins bonne pour une petite

partie des zones, sans toutefois être très grave.

Enfin, mentionnons que pour toutes les simulations, presque toutes les valeurs de latence sont inférieures à 250 millisecondes, ce qui est très performant. Mentionnons également qu'aucune zone ne possède de latence supérieure à 1000 millisecondes, toutes simulations confondues. La latence moyenne ne semble donc que faiblement dépendante des valeurs de charge moyenne des différentes zones et des conditions de simulations exposées dans les six simulations. En d'autres termes, le modèle pair-à-pair proposé semble être en mesure de maintenir des latences moyennes très bonnes à la fois sous des conditions de déroulement strictes et moins strictes.

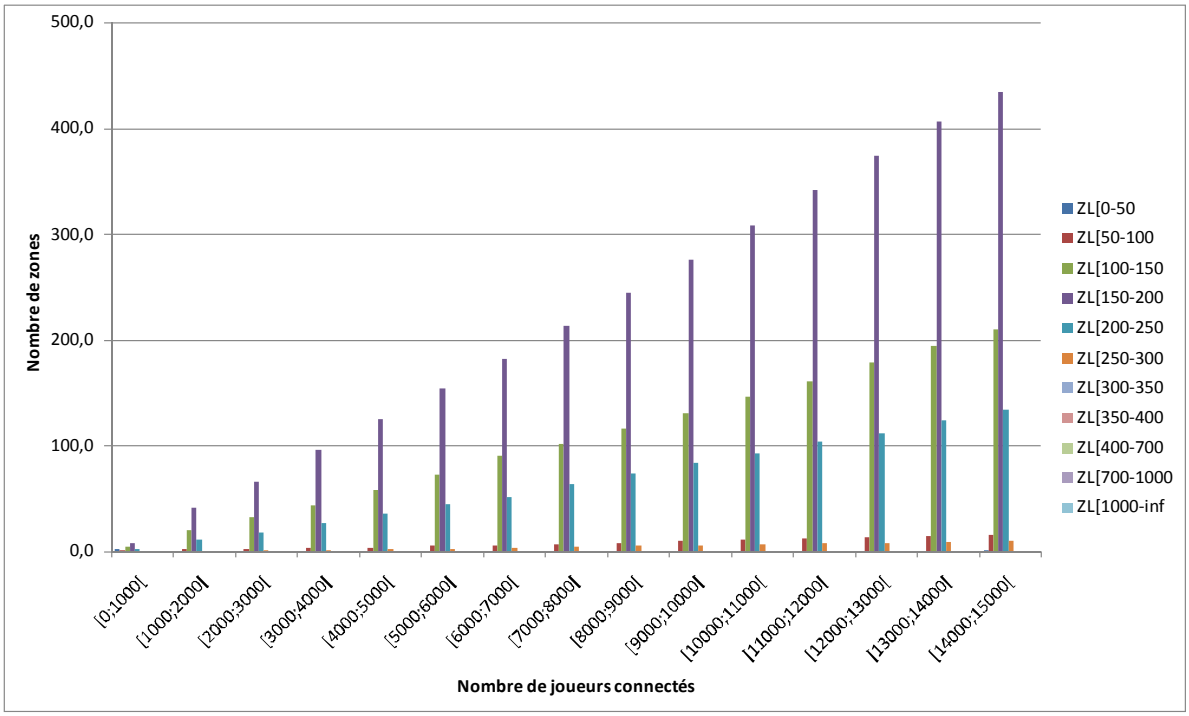


(a) Nombre moyen de zones par classe de latence en fonction du nombre de joueurs

Latence :	0-50	50-100	100-150	150-200	200-250	250-300	300-350	350-400	400-700	700-1000	1000-inf
[0; 1000[	7,2%	10,6%	35,5%	32,9%	10,9%	2,2%	0,5%	0,1%	0,0%	0,0%	0,0%
[1000; 2000[	1,5%	11,7%	35,9%	35,8%	12,4%	2,2%	0,3%	0,1%	0,0%	0,0%	0,0%
[2000; 3000[	1,3%	11,6%	36,1%	35,1%	12,7%	2,7%	0,4%	0,1%	0,0%	0,0%	0,0%
[3000; 4000[	1,3%	12,2%	35,0%	35,3%	12,9%	2,7%	0,4%	0,1%	0,0%	0,0%	0,0%
[4000; 5000[	1,3%	12,3%	35,6%	34,5%	13,0%	2,7%	0,4%	0,1%	0,0%	0,0%	0,0%
[5000; 6000[	1,2%	12,2%	35,3%	34,9%	13,0%	2,7%	0,5%	0,1%	0,0%	0,0%	0,0%
[6000; 7000[	1,3%	12,4%	35,4%	34,4%	12,9%	2,8%	0,5%	0,1%	0,0%	0,0%	0,0%
[7000; 8000[	1,3%	12,5%	35,0%	34,2%	13,3%	2,9%	0,5%	0,1%	0,0%	0,0%	0,0%
[8000; 9000[	1,5%	12,6%	35,0%	33,8%	13,3%	2,9%	0,5%	0,1%	0,0%	0,0%	0,0%
[9000; 10000[	1,5%	12,7%	34,7%	34,0%	13,3%	3,1%	0,6%	0,1%	0,0%	0,0%	0,0%
[10000; 11000[	1,4%	12,8%	34,7%	33,8%	13,3%	3,1%	0,6%	0,1%	0,0%	0,0%	0,0%
[11000; 12000[	1,4%	12,8%	34,6%	33,8%	13,4%	3,1%	0,6%	0,1%	0,0%	0,0%	0,0%
[12000; 13000[	1,4%	12,9%	34,5%	33,6%	13,6%	3,1%	0,6%	0,1%	0,0%	0,0%	0,0%
[13000; 14000[	1,4%	12,7%	34,3%	33,8%	13,5%	3,3%	0,6%	0,1%	0,0%	0,0%	0,0%
[14000; 15000[	1,4%	12,5%	33,9%	33,8%	13,9%	3,4%	0,7%	0,1%	0,0%	0,0%	0,0%

(b) Nombre moyen de zones par classe de latence en fonction du nombre de joueurs (pourcentage)

FIGURE 4.25 Analyse de la latence pour la simulation 1

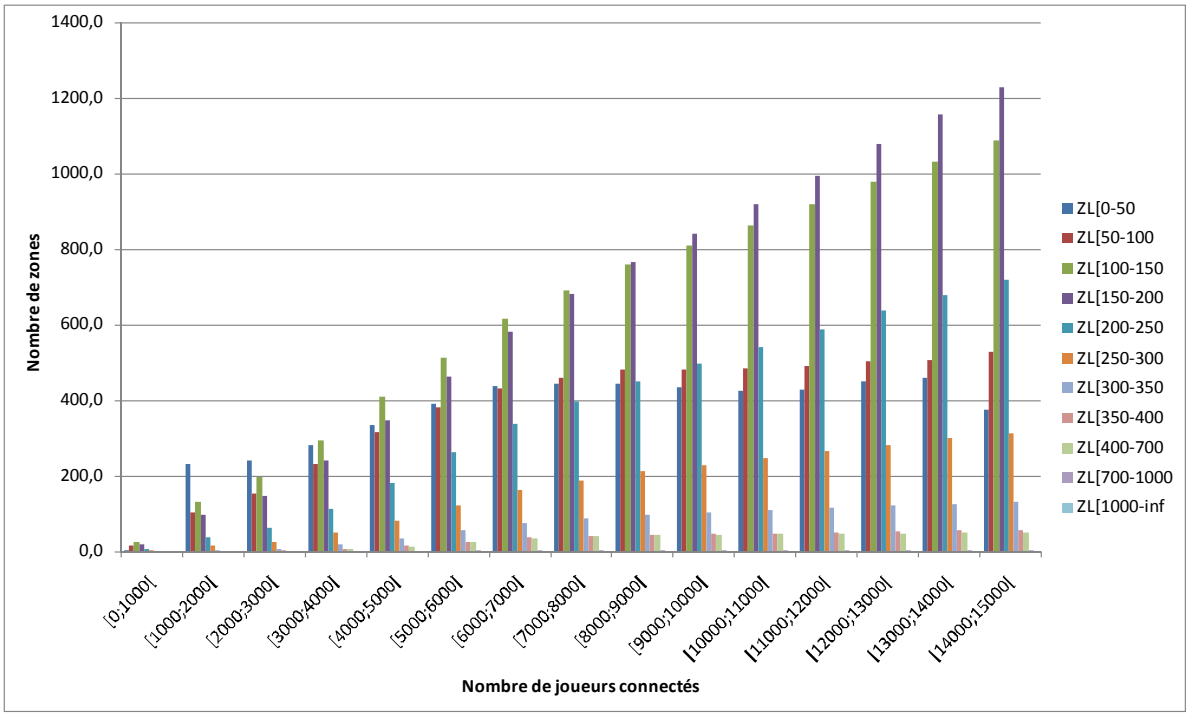


(a) Nombre moyen de zones par classe de latence en fonction du nombre de joueurs

Latence :	0-50	50-100	100-150	150-200	200-250	250-300	300-350	350-400	400-700	700-1000	1000-inf
[0; 1000[	14,1%	5,3%	25,6%	44,2%	9,6%	1,0%	0,0%	0,0%	0,0%	0,0%	0,0%
[1000; 2000[	0,3%	2,2%	26,5%	55,9%	14,3%	0,8%	0,0%	0,0%	0,0%	0,0%	0,0%
[2000; 3000[	0,1%	2,2%	26,9%	54,7%	15,0%	0,9%	0,0%	0,0%	0,0%	0,0%	0,0%
[3000; 4000[	0,0%	1,7%	25,6%	55,9%	15,7%	1,0%	0,0%	0,0%	0,0%	0,0%	0,0%
[4000; 5000[	0,0%	1,6%	25,7%	55,6%	15,8%	1,0%	0,0%	0,0%	0,0%	0,0%	0,0%
[5000; 6000[	0,0%	1,8%	25,9%	55,0%	16,0%	1,1%	0,0%	0,0%	0,0%	0,0%	0,0%
[6000; 7000[	0,0%	1,7%	27,1%	54,4%	15,5%	1,1%	0,0%	0,0%	0,0%	0,0%	0,0%
[7000; 8000[	0,1%	1,7%	25,9%	54,7%	16,3%	1,1%	0,1%	0,0%	0,0%	0,0%	0,0%
[8000; 9000[	0,1%	1,8%	25,9%	54,5%	16,4%	1,2%	0,1%	0,0%	0,0%	0,0%	0,0%
[9000; 10000[	0,1%	1,9%	25,7%	54,4%	16,5%	1,2%	0,1%	0,0%	0,0%	0,0%	0,0%
[10000; 11000[	0,1%	1,9%	25,8%	54,4%	16,3%	1,3%	0,1%	0,0%	0,0%	0,0%	0,0%
[11000; 12000[	0,1%	2,0%	25,5%	54,4%	16,5%	1,3%	0,1%	0,0%	0,0%	0,0%	0,0%
[12000; 13000[	0,1%	1,9%	26,0%	54,4%	16,3%	1,2%	0,1%	0,0%	0,0%	0,0%	0,0%
[13000; 14000[	0,1%	1,9%	25,8%	54,1%	16,6%	1,3%	0,1%	0,0%	0,0%	0,0%	0,0%
[14000; 15000[	0,1%	1,9%	26,0%	53,8%	16,6%	1,3%	0,1%	0,0%	0,0%	0,0%	0,0%

(b) Nombre moyen de zones par classe de latence en fonction du nombre de joueurs (pourcentage)

FIGURE 4.26 Analyse de la latence pour la simulation 2

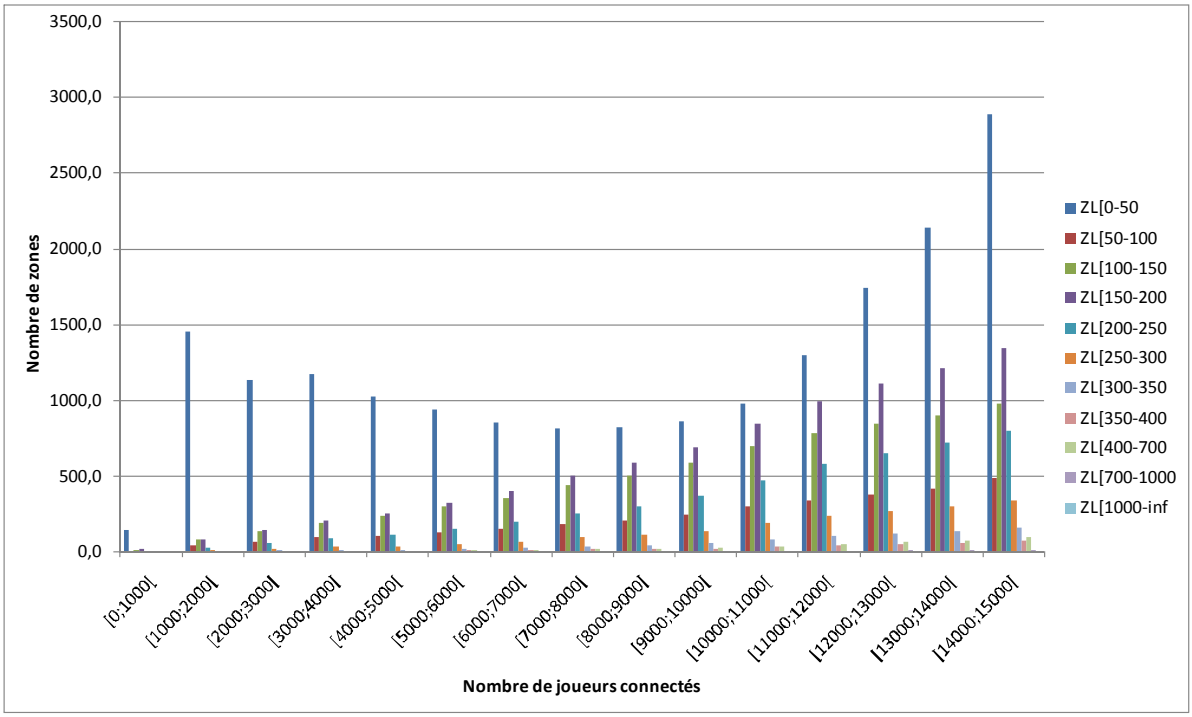


(a) Nombre moyen de zones par classe de latence en fonction du nombre de joueurs

Latence :	0-50	50-100	100-150	150-200	200-250	250-300	300-350	350-400	400-700	700-1000	1000-inf
[0; 1000[	5,5%	20,9%	33,1%	24,4%	9,6%	4,1%	1,5%	0,4%	0,1%	0,0%	0,0%
[1000; 2000[	37,2%	16,5%	21,2%	15,3%	6,2%	2,3%	0,7%	0,2%	0,1%	0,0%	0,0%
[2000; 3000[	28,5%	18,1%	23,5%	17,5%	7,5%	2,9%	1,0%	0,3%	0,2%	0,0%	0,0%
[3000; 4000[	22,5%	18,6%	23,6%	19,1%	9,1%	3,9%	1,6%	0,6%	0,5%	0,0%	0,0%
[4000; 5000[	19,3%	18,2%	23,4%	20,0%	10,4%	4,7%	2,1%	0,9%	0,8%	0,0%	0,0%
[5000; 6000[	17,5%	17,0%	22,8%	20,6%	11,6%	5,4%	2,6%	1,2%	1,1%	0,1%	0,0%
[6000; 7000[	16,1%	15,9%	22,6%	21,4%	12,4%	5,9%	2,8%	1,3%	1,3%	0,1%	0,0%
[7000; 8000[	14,6%	15,1%	22,7%	22,4%	13,1%	6,2%	2,9%	1,4%	1,3%	0,1%	0,0%
[8000; 9000[	13,4%	14,6%	23,0%	23,2%	13,6%	6,4%	3,0%	1,4%	1,3%	0,1%	0,0%
[9000; 10000[	12,5%	13,7%	23,2%	24,1%	14,2%	6,5%	3,0%	1,3%	1,3%	0,1%	0,0%
[10000; 11000[	11,5%	13,1%	23,4%	24,9%	14,6%	6,7%	3,0%	1,3%	1,2%	0,1%	0,0%
[11000; 12000[	11,0%	12,5%	23,5%	25,5%	15,1%	6,8%	3,0%	1,3%	1,2%	0,1%	0,0%
[12000; 13000[	10,8%	12,1%	23,5%	25,9%	15,3%	6,8%	2,9%	1,3%	1,2%	0,1%	0,0%
[13000; 14000[	10,5%	11,6%	23,6%	26,4%	15,5%	6,9%	2,9%	1,3%	1,2%	0,1%	0,0%
[14000; 15000[	8,3%	11,7%	24,2%	27,3%	16,0%	7,0%	2,9%	1,3%	1,1%	0,1%	0,0%

(b) Nombre moyen de zones par classe de latence en fonction du nombre de joueurs (pourcentage)

FIGURE 4.27 Analyse de la latence pour la simulation 3

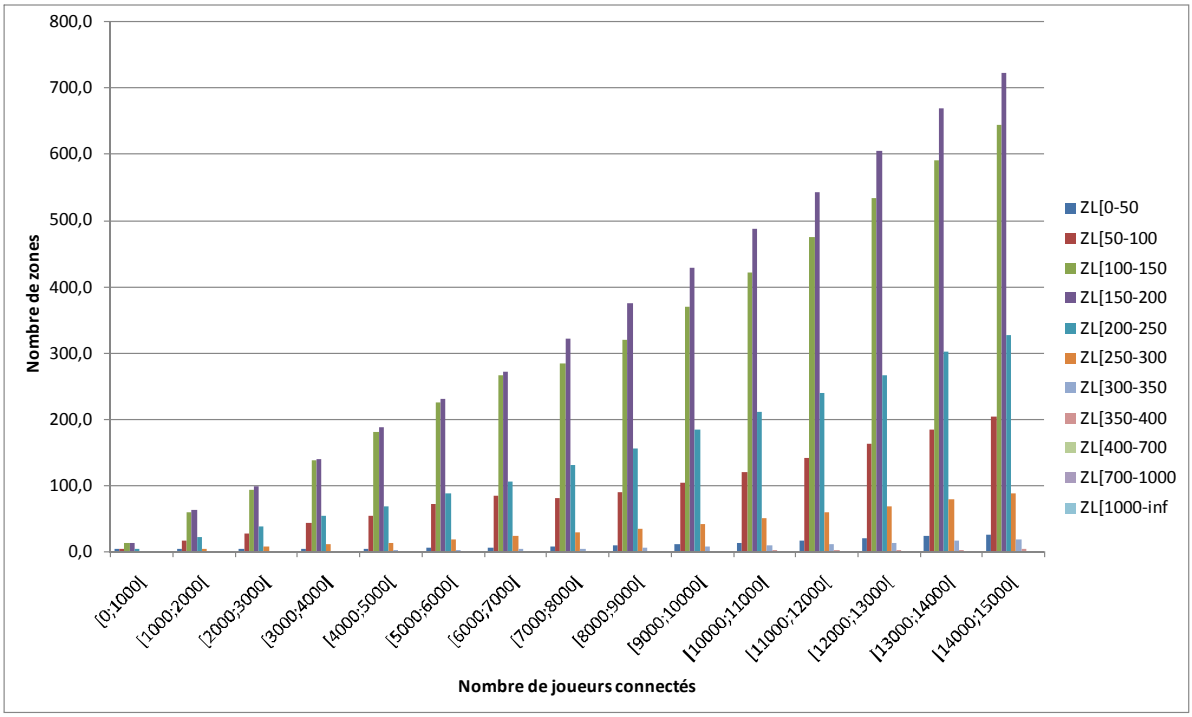


(a) Nombre moyen de zones par classe de latence en fonction du nombre de joueurs

Latence :	0-50	50-100	100-150	150-200	200-250	250-300	300-350	350-400	400-700	700-1000	1000-inf
[0; 1000[	77,1%	2,7%	8,0%	8,5%	2,9%	0,6%	0,1%	0,0%	0,0%	0,0%	0,0%
[1000; 2000[	85,5%	2,5%	4,7%	4,8%	1,7%	0,5%	0,2%	0,1%	0,1%	0,0%	0,0%
[2000; 3000[	71,7%	4,3%	8,5%	9,0%	3,9%	1,3%	0,6%	0,3%	0,4%	0,1%	0,0%
[3000; 4000[	64,5%	5,2%	10,6%	11,4%	4,9%	1,7%	0,7%	0,3%	0,4%	0,1%	0,0%
[4000; 5000[	56,9%	6,0%	13,1%	14,3%	6,1%	1,9%	0,8%	0,3%	0,4%	0,1%	0,0%
[5000; 6000[	48,4%	6,7%	15,3%	16,9%	7,8%	2,7%	1,0%	0,4%	0,5%	0,1%	0,0%
[6000; 7000[	41,1%	7,2%	16,9%	19,4%	9,6%	3,4%	1,3%	0,5%	0,6%	0,1%	0,0%
[7000; 8000[	34,2%	7,9%	18,5%	21,3%	10,8%	4,0%	1,6%	0,7%	0,7%	0,1%	0,0%
[8000; 9000[	31,6%	7,9%	19,2%	22,4%	11,5%	4,3%	1,6%	0,7%	0,7%	0,1%	0,0%
[9000; 10000[	28,7%	8,2%	19,5%	23,0%	12,3%	4,7%	1,8%	0,8%	0,8%	0,1%	0,0%
[10000; 11000[	26,9%	8,2%	19,2%	23,1%	13,0%	5,3%	2,2%	0,9%	0,9%	0,1%	0,0%
[11000; 12000[	29,2%	7,6%	17,8%	22,4%	13,0%	5,5%	2,3%	1,0%	1,1%	0,1%	0,0%
[12000; 13000[	33,2%	7,1%	16,1%	21,2%	12,4%	5,2%	2,3%	1,0%	1,2%	0,2%	0,0%
[13000; 14000[	35,9%	6,9%	15,0%	20,3%	12,0%	5,0%	2,3%	1,0%	1,2%	0,2%	0,0%
[14000; 15000[	40,2%	6,8%	13,6%	18,7%	11,1%	4,7%	2,2%	1,1%	1,4%	0,2%	0,0%

(b) Nombre moyen de zones par classe de latence en fonction du nombre de joueurs (pourcentage)

FIGURE 4.28 Analyse de la latence pour la simulation 4

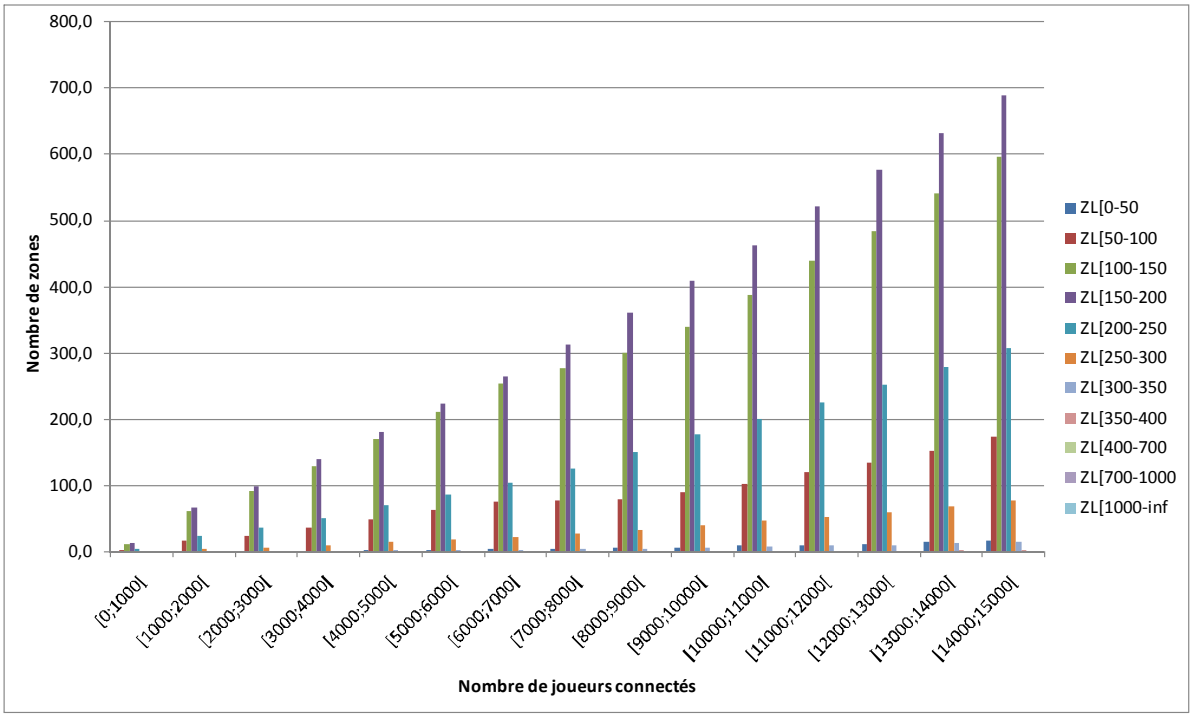


(a) Nombre moyen de zones par classe de latence en fonction du nombre de joueurs

Latence :	0-50	50-100	100-150	150-200	200-250	250-300	300-350	350-400	400-700	700-1000	1000-inf
[0; 1000[	12,6%	9,7%	31,2%	32,0%	11,5%	2,3%	0,5%	0,0%	0,0%	0,0%	0,0%
[1000; 2000[	2,3%	10,0%	34,4%	36,3%	13,3%	2,8%	0,5%	0,1%	0,0%	0,0%	0,0%
[2000; 3000[	1,4%	10,5%	34,7%	36,2%	13,9%	2,7%	0,4%	0,1%	0,0%	0,0%	0,0%
[3000; 4000[	1,2%	11,1%	35,0%	35,7%	13,6%	2,8%	0,3%	0,1%	0,0%	0,0%	0,0%
[4000; 5000[	0,9%	10,8%	35,1%	36,5%	13,2%	2,8%	0,4%	0,0%	0,0%	0,0%	0,0%
[5000; 6000[	1,0%	11,1%	34,9%	35,8%	13,7%	2,8%	0,5%	0,1%	0,0%	0,0%	0,0%
[6000; 7000[	0,9%	11,1%	34,7%	35,4%	13,9%	3,1%	0,5%	0,1%	0,0%	0,0%	0,0%
[7000; 8000[	0,9%	9,5%	32,8%	37,3%	15,1%	3,3%	0,6%	0,1%	0,0%	0,0%	0,0%
[8000; 9000[	0,9%	9,0%	32,2%	37,5%	15,7%	3,5%	0,6%	0,1%	0,0%	0,0%	0,0%
[9000; 10000[	1,0%	9,1%	32,0%	37,1%	15,9%	3,7%	0,6%	0,1%	0,0%	0,0%	0,0%
[10000; 11000[	1,0%	9,1%	31,8%	36,8%	16,0%	3,9%	0,8%	0,1%	0,0%	0,0%	0,0%
[11000; 12000[	1,1%	9,4%	31,8%	36,4%	16,0%	4,0%	0,8%	0,2%	0,0%	0,0%	0,0%
[12000; 13000[	1,2%	9,7%	31,8%	36,0%	15,9%	4,1%	0,8%	0,2%	0,0%	0,0%	0,0%
[13000; 14000[	1,3%	9,8%	31,5%	35,7%	16,1%	4,2%	0,9%	0,2%	0,1%	0,0%	0,0%
[14000; 15000[	1,3%	10,0%	31,6%	35,4%	16,0%	4,3%	0,9%	0,2%	0,1%	0,0%	0,0%

(b) Nombre moyen de zones par classe de latence en fonction du nombre de joueurs (pourcentage)

FIGURE 4.29 Analyse de la latence pour la simulation 5



(a) Nombre moyen de zones par classe de latence en fonction du nombre de joueurs

Latence :	0-50	50-100	100-150	150-200	200-250	250-300	300-350	350-400	400-700	700-1000	1000-inf
[0; 1000[	0,5%	9,5%	34,6%	37,2%	14,7%	2,9%	0,5%	0,0%	0,0%	0,0%	0,0%
[1000; 2000[	0,5%	9,5%	35,0%	37,5%	14,1%	2,6%	0,5%	0,1%	0,0%	0,0%	0,0%
[2000; 3000[	0,5%	9,5%	34,9%	37,6%	14,2%	2,6%	0,4%	0,0%	0,0%	0,0%	0,0%
[3000; 4000[	0,5%	9,9%	34,9%	37,6%	13,8%	2,7%	0,4%	0,1%	0,0%	0,0%	0,0%
[4000; 5000[	0,5%	10,1%	34,5%	36,8%	14,4%	2,9%	0,5%	0,0%	0,0%	0,0%	0,0%
[5000; 6000[	0,5%	10,4%	34,5%	36,6%	14,2%	3,0%	0,5%	0,1%	0,0%	0,0%	0,0%
[6000; 7000[	0,6%	10,3%	34,7%	36,2%	14,4%	3,0%	0,4%	0,1%	0,0%	0,0%	0,0%
[7000; 8000[	0,6%	9,4%	33,1%	37,5%	15,1%	3,3%	0,5%	0,1%	0,0%	0,0%	0,0%
[8000; 9000[	0,6%	8,4%	31,9%	38,3%	16,1%	3,6%	0,6%	0,1%	0,0%	0,0%	0,0%
[9000; 10000[	0,7%	8,4%	31,5%	38,1%	16,5%	3,7%	0,6%	0,1%	0,0%	0,0%	0,0%
[10000; 11000[	0,7%	8,4%	31,6%	37,8%	16,4%	3,9%	0,7%	0,1%	0,0%	0,0%	0,0%
[11000; 12000[	0,8%	8,7%	31,7%	37,6%	16,2%	3,8%	0,7%	0,1%	0,0%	0,0%	0,0%
[12000; 13000[	0,8%	8,8%	31,5%	37,5%	16,4%	3,9%	0,7%	0,1%	0,0%	0,0%	0,0%
[13000; 14000[	0,9%	8,9%	31,7%	36,9%	16,3%	4,0%	0,8%	0,1%	0,0%	0,0%	0,0%
[14000; 15000[	0,9%	9,2%	31,6%	36,6%	16,3%	4,1%	0,8%	0,2%	0,0%	0,0%	0,0%

(b) Nombre moyen de zones par classe de latence en fonction du nombre de joueurs (pourcentage)

FIGURE 4.30 Analyse de la latence pour la simulation 6

Chapitre 5

CONCLUSION

Ce chapitre effectuera, dans un premier temps, une synthèse des principaux objectifs visés ainsi que des principaux éléments réalisés dans le cadre de ce travail. Par la suite, les principales limitations inhérentes au modèle proposé seront explicitées. Ces limitations devraient être retravaillées ultérieurement, dans le cadre de travaux futurs. Enfin, quelques pistes d'amélioration seront mentionnées.

5.1 Synthèse des travaux

L'objectif du projet de recherche était de proposer une architecture novatrice de gestion en temps réel de jeux massivement multijoueurs en ligne. À l'heure actuelle, toutes les implémentations commerciales de jeux massivement multijoueurs en ligne fonctionnent sur la base du modèle client-serveur classique. Cette approche possède certains avantages, mais s'avère très onéreuse pour l'opérateur de tels jeux qui doit maintenir un ou un ensemble de serveurs dédiés à la gestion du jeu. Puisque la capacité d'une machine même très puissante est éventuellement limitée, dans le cas de jeux très populaires qui comportement des dizaines de milliers et parfois même jusqu'à des millions de joueurs, il n'est pas possible pour un seul serveur de prendre en charge tous les joueurs. L'intérêt du modèle pair-à-pair découle de cette problématique. Le modèle pair-à-pair vise à redistribuer la charge de traitement requise pour opérer le jeu, que ce soit en partie ou en totalité, aux joueurs eux-mêmes. Un joueur qui joue peut donc être appelé à agir en tant que serveur pour un groupe d'autres joueurs.

Certaines approches ont été mentionnées dans la littérature, mais elles ne remplissent pas l'ensemble des besoins inhérent à de tels jeux. Parmi ces besoins, mentionnons la prise en charge d'un nombre très élevé très variable de joueurs à tout instant du déroulement du jeu, la prise en compte des capacités des noeuds et de la latence. Nous avons décidé d'aller plus loin en proposant un modèle pair-à-pair hybride (avec présence d'un serveur central qui agit en tant que coordonnateur) pouvant continuellement s'ajuster aux besoins de en ressources. Notre modèle divise le jeu en un ensemble de zones de forme triangulaire et assigne la responsabilité de la gestion de chaque zone à un noeud joueur participant au jeu. Le serveur central analyse l'état en continu et, par un ensemble d'opérations de rebalancement, modifie

à la volée la répartition actuelle de la surcouche pair-à-pair afin de palier aux variations de capacité requise au niveau des différentes zones. Le défi est toutefois de s'assurer que le nombre d'opérations de rebalancement effectuées demeure assez minime pour éviter des cycles de changements continuels.

Un simulateur implémentant totalement le modèle proposé a été réalisé afin de valider le fonctionnement et l'efficacité du modèle. Ce simulateur a servi à produire plusieurs simulations et en tirer des jeux de données qui ont pu être analysés pour vérifier le fonctionnement sous différentes conditions. Les résultats obtenus par les différentes simulations ont montré que le modèle fonctionne très bien sous des conditions normales. Si une capacité raisonnable maximale est assignée aux noeuds serveurs, le modèle et ses algorithmes arrivent à effectuer la gestion et la régulation du jeu en s'adaptant automatiquement selon le nombre de joueurs connectés. Bien que ces conclusions semblent prometteuses si l'on devait réaliser une véritable implémentation de ce modèle, il faut tout de même mentionner un petit bémol concernant les endroits où il y a un fort volume de joueurs. Cette limitation sera détaillée davantage à la section suivante (5.2).

5.2 Limitations de la solution proposée

Il y a malheureusement certaines limitations au niveau du modèle proposé et de son implémentation. En ce qui concerne le calcul des latences, nous n'avons pas considéré que la latence pouvait varier dans le temps ; nous avons assigné une valeur dynamique entre chaque paire de noeuds. Le même cas s'applique à la capacité de charge, en partie liée à la notion de bande passante, qui peut varier lors de l'utilisation. Les techniques de détection de latence et de capacité de charge n'ont pas été explicitées.

Dans le même ordre d'idées, les différentes pondérations utilisées pour le calcul des coûts ne sont que des valeurs arbitraires relatives et ne sont pas liées au coût réel en termes de ressources au sein d'un véritable environnement. En d'autres mots, pour être capable de bien estimer les coûts, il faudrait construire un vrai prototype du modèle avec un protocole sous-jacent. De plus, les différents paramètres entrant en jeu dans le calcul des coûts ne sont peut-être pas exactement représentatifs de l'ensemble des intéractions réelles entre les entités.

Nous avons également négligé le phénomène de relève lors d'un changement de noeud responsable de zone. Dans le cas de notre modèle, le changement est instantané, alors que normalement, il y présence d'un délai associé au changement. Ce délai pourrait nuire à la qualité de jeu et des mécanismes devraient être proposés pour en tenir compte. De plus, nous ne tenons pas compte des déconnexions dures (un noeud responsable d'une zone quitte de façon involontaire). Par contre, en ce qui a trait à la relève inter-zone (un joueur passe d'une

zone à l'autre), ce concept est modélisé par le fait que dès que la zone d'intérêt du joueur chevauche celle d'une zone avoisinante, une connexion est établie.

Concernant le simulateur lui-même, une limitation concerne les joueurs-serveurs qui se déconnectent. Normalement, un noeud responsable d'une zone donnée devrait céder le contrôle de sa zone à un autre noeud. Dans l'implémentation actuelle, le noeud conserve le contrôle.

Au niveau de la génération des rapports, les données sont très fiables, mais il y a tout de même une petite marge d'erreur. Puisque la simulation s'effectue à chaque seconde (sans que ce soit précisé s'il s'agit du début ou de la fin de la seconde), il arrive parfois que le moment dans la seconde où le rapport est généré et le moment où la simulation a été réalisée dans ladite seconde ne soit pas tout à fait identique, ce qui cause une très légère perte de précision. Nous pourrions également ajouter quelques bugs qui n'ont pas été corrigés qui doivent sous certaines conditions être "contournés" par l'utilisateur du simulateur. Un autre point est que le simulateur produit des fichiers de très grande taille et consomme beaucoup de mémoire. Sans être critique en soi, il y aurait sûrement moyen de réduire l'utilisation des ressources.

En ce qui concerne les simulations de plus longue durée ou avec significativement plus de joueurs, il y a malheureusement certaines contraintes de capacité mémoire qui en limitent actuellement le déroulement. En début de projet, nous avions pensé effectuer des simulations avec beaucoup plus de joueurs : 50000 et même jusqu'à 100000. Nous avons malheureusement dû limiter le nombre à 15000, dans un premier temps en raison du temps de simulation qui aurait été beaucoup trop long : les six simulations réalisées et analysées ont pris chacune 24 à 72 heures dépendant des paramètres choisis (les simulations les mieux réussies ont pris moins de temps alors que les simulations les moins bien réussies en ont pris plus). La procédure d'extraction des données (la génération des rapports au sein du simulateur) a nécessité elle aussi beaucoup de temps (environ 12 heures par simulation). Dans un second temps, la consommation de ressources a également été un facteur limitatif. Pour ces deux raisons, augmenter le nombre de joueurs aurait été problématique avec le simulateur actuel. Optimiser le simulateur aurait été possible, mais le temps pris pouvait difficilement être estimé et ne cadrait pas avec l'échéancier du projet.

Mentionnons également une limitation qui nous semblait importante concernant les zones avec une très forte densité de joueurs. Dans notre modèle, nous avons considéré des régions d'intérêt avec une concentration plus élevée de joueurs. Si la concentration de joueurs demeure en-délà d'un certain seuil en rapport à la capacité maximale des noeuds, l'algorithme de rebalancement permet de gérer la situation. Toutefois, si les régions d'intérêt contiennent une concentration de joueurs très élevée (chevauchement d'une très grande quantité de joueurs dans un espace réduit), il arrive un point où l'algorithme ne parvient plus à gérer les joueurs. Il y a donc plusieurs séparations de zones successives jusqu'en arriver à un point où les

zones deviennent tellement petites que chaque joueur englobe plusieurs zones. Lorsque l'on en arrive à ce point, l'algorithme de séparation n'est plus efficace et les multiples zones demeurent surchargées. L'algorithme de rebalancement tente continuellement d'effectuer de nouvelles séparations à chaque seconde, ce qui fragmente encore davantage et consomme énormément de ressources. Il faudrait prévoir un moyen de gérer efficacement les zones à très forte densité de population.

5.3 Recherches futures

Il serait intéressant d'effectuer davantage de simulations en faisant varier davantage de paramètres et d'effectuer des analyses plus poussées sur certains critères. Quelques exemples de paramètres que l'on pourrait varier :

- le nombre de joueurs ;
- la taille de la carte (pour étudier le lien entre le nombre de joueurs et la taille de la carte) ;
- la vitesse de déplacement des joueurs et le temps de pause (pour différents types de jeux comme des jeux de course par exemple) ;
- la taille de la zone d'intérêt des joueurs ;
- les patrons de connectivité ;
- les différentes valeurs de pondérations pour les calculs.

Dans le cadre de recherches futures, il serait intéressant de concevoir un protocole permettant de modéliser et simuler plus adéquatement l'échange de message entre les différentes entités. Suite à la conception du protocole, un véritable prototype simulatoire pourrait être conçu. Ce prototype pourrait permettre d'intégrer à la fois de vrais clients de jeu et des clients simulés. La transmission de messages serait effectuée complètement. Il serait beaucoup plus facile de mesurer exactement le flot d'informations. Une conséquence de cette modélisation est que l'on pourrait définir de vraies capacités de connectivité réseau aux noeuds et vérifier si la capacité est respectée ou dépassée.

Un autre aspect concerne les clients mobiles. La tendance actuelle du marché est aux "smart phones", iPods et périphériques de jeux portables tous connectés à Internet. Chacun de ces périphériques possède des capacités et une bande passante différente. Il serait fort intéressant d'adapter le modèle pour prendre en compte les capacités de ces clients mobiles. La puissance de ces périphériques et leur connexion réseau sont telles qu'il est maintenant réaliste d'envisager que ces derniers puissent également jouer un rôle de serveur au sein d'un réseau pair-à-pair. Un modèle prenant en compte une telle hétérogénité de clients permettrait à tous de jouer et à tous de contribuer au bon fonctionnement du jeu.

Un aspect final important concerne certainement le volet sécurité. En effet, l'utilisation de réseaux pair-à-pair pour effectuer la gestion des jeux massivement multijoueurs en ligne pourrait poser certains problèmes d'intégrité des données. Rien n'empêche un noeud mal intentionné de modifier les données transmises pour favoriser ou défavoriser certains joueurs au détriment de d'autres. Un noeud malicieux pourrait également affecter la qualité du jeu pour les noeuds sous sa responsabilité, en retardant la livraison de données par exemple. L'aspect sécurité n'a pas été abordé du tout dans le cadre de ce travail, mais il est certain que dans la perspective de développement d'un vrai produit, cet aspect serait prioritaire.

Références

AHMED, D. T. et SHIRMOHAMMADI, S. (2008). A dynamic area of interest management and collaboration model for p2p mmogs. *DS-RT '08 : Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*. IEEE Computer Society, Washington, DC, USA, 27–34.

ARIYAKHAJORN, J., WANNAWILAI, P. et SATHITWIRIYAWONG, C. (2006). A comparative study of random waypoint and gauss-markov mobility models in the performance evaluation of manet. *Communications and Information Technologies, 2006. ISCIT '06. International Symposium on*. 894 –899.

ASSIOTIS, M. et TZANOV, V. (2006). A distributed architecture for mmorpg. *NetGames '06 : Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*. ACM, New York, NY, USA, 4.

AURENHAMMER, F. (1991). Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23, 345–405.

BALAKRISHNAN, H., KAASHOEK, M. F., KARGER, D., MORRIS, R. et STOICA, I. (2003). Looking up data in p2p systems. *Commun. ACM*, 46, 43–48.

CASTRO, M., DRUSCHEL, P., HU, Y. C. et ROWSTRON, A. (2002). Topology-aware routing in structured peer-to-peer overlay networks.

CHAN, L., YONG, J., BAI, J., LEONG, B. et TAN, R. (2007). Hydra : a massively-multiplayer peer-to-peer architecture for the game developer. *NetGames '07 : Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*. ACM, New York, NY, USA, 37–42.

CHEW, L. P. (1987). Constrained delaunay triangulations. *SCG '87 : Proceedings of the third annual symposium on Computational geometry*. ACM, New York, NY, USA, 215–222.

DE VLEESCHAUWER, B., VAN DEN BOSSCHE, B., VERDICKT, T., DE TURCK, F., DHOEDT, B. et DEMEESTER, P. (2005). Dynamic microcell assignment for massively

multiplayer online gaming. *NetGames '05 : Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. ACM, New York, NY, USA, 1–7.

GREENHALGH, C. et BENFORD, S. (1995). Massive : a distributed virtual reality system incorporating spatial trading. *Distributed Computing Systems, 1995., Proceedings of the 15th International Conference on*. 27–34.

GUMMADI, K. P., SAROIU, S. et GRIBBLE, S. D. (2002). King : estimating latency between arbitrary internet end hosts. *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, New York, NY, USA, IMW '02, 5–18.

KNUTSSON, B., LU, H., XU, W. et HOPKINS, B. (2004). Peer-to-peer support for massively multiplayer games. *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*. vol. 1, –107.

LITWIN, W. (1980). Linear hashing : A new tool for file and table addressing. *Sixth International Conference on Very Large Data Bases, October 1-3, 1980, Montreal, Quebec, Canada, Proceedings*. IEEE Computer Society, 212–223.

RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R. et SCHENKER, S. (2001). A scalable content-addressable network. *SIGCOMM '01 : Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, New York, NY, USA, 161–172.

ROWSTRON, A. I. T. et DRUSCHEL, P. (2001). Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Middleware '01 : Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*. Springer-Verlag, London, UK, 329–350.

TANENBAUM, A., LANGSAM, Y. et AUGENSTEIN, M. J. (1990). *Data Structures Using C.* Prentice Hall.

TARNG, P.-Y., CHEN, K.-T. et HUANG, P. (2008). An analysis of wow players' game hours. *NetGames '08 : Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*. ACM, New York, NY, USA, 47–52.

VAN DEN BOSSCHE, B., DE VLEESCHAUWER, B., VERDICKT, T., DE TURCK, F.,

DHOEDT, B. et DEMEESTER, P. (2009). Autonomic microcell assignment in massively distributed online virtual environments. *J. Netw. Comput. Appl.*, 32, 1242–1256.

VAN DEN BOSSCHE, B., VERDICKT, T., DE VLEESCHAUWER, B., DESMET, S., DE MULDER, S., DE TURCK, F., DHOEDT, B. et DEMEESTER, P. (2006). A platform for dynamic microcell redeployment in massively multiplayer online games. *NOSSDAV '06 : Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video*. ACM, New York, NY, USA, 1–6.

WEBB, S. D. et SOH, S. (2007). Cheating in networked computer games : a review. *DIMEA '07 : Proceedings of the 2nd international conference on Digital interactive media in entertainment and arts*. ACM, New York, NY, USA, 105–112.

YU, A. P. et VUONG, S. T. (2005). Mopar : a mobile peer-to-peer overlay rrchitecture for interest management of massively multiplayer online games. *NOSSDAV '05 : Proceedings of the international workshop on Network and operating systems support for digital audio and video*. ACM, New York, NY, USA, 99–104.