Universitat de Lleida

Modelado y Planificación de Aplicaciones de Juegos Masivos Multijugador en Red en Entornos Distribuidos

Ignasi Barri Vilardell

Escola Politècnica Superior
Departament d'Informàtica i Enginyeria Industrial
Universitat de Lleida

Modelado y Planificación de Aplicaciones de Juegos Masivos Multijugador en Red en Entornos Distribuidos

Memoria presentada para lograr el grado de
Doctor por la Universitat de Lleida
por

Ignasi Barri Vilardell

Dirigida por

Dr. Francesc Giné de Sola **Dra. Concepció Roig Mateu**

Programa de doctorado en Ingeniería

Lleida, enero de 2012

Agradecimientos

Cuando se empieza una tesis doctoral, pocos son los que se imaginan del reto al que se enfrentan. A medida que van avanzando los días y el trabajo va tomando forma, siendo el final cada vez más próximo, es entonces cuando se tiene consciencia del esfuerzo que ha supuesto a todos los niveles. Por este motivo quiero dedicar estas líneas a aquellas personas que han hecho posible que este momento llegue.

Deseo expresar mi más sincero agradecimiento al Dr. Francesc Giné de Sola y a la Dra. Concepció Roig Mateu por su magistral dirección y constante dedicación en la supervisión del trabajo, por la gran ayuda recibida en la redacción de artículos y el apoyo moral que me han ofrecido en todo momento.

Un especial agradecimiento al resto de profesores del Grupo de Computación Distribuida (GCD) de la Universitat de Lleida (UdL), Fernando Cores, Xavier Faus, Fernando Guirado, Josep Lluís Lérida, Valentí Pardo, Albert Saiz, Josep M. Solà y Francesc Solsona. Por haberme hecho sentir uno más del grupo, tanto a nivel de trabajo como a nivel personal.

A Josep Rius, amigo y compañero a lo largo de todas mis etapas como estudiante, por su apoyo, escuchando y aguantando mis teorías, altibajos y sueños imposibles. A Miquel Orobitg, compañero de despacho, por apoyarme en los momentos en los que los resultados no eran los esperados. A mi compañero Alberto Montañola, por compartir su sabiduría técnica en materia de MMOG y en gestión de máquinas virtuales, sin su ayuda, la monitorización efectuada en esta tesis no hubiera sido tan eficaz. A Ivan Teixidó por el gran

soporte técnico ofrecido y los buenos momentos en los ratos de descanso. A Héctor Blanco, con el cual compartimos buenos momentos juntos en congresos internacionales. A Damià Castellà, por haber marcado el camino a seguir al resto de becarios del área. Y finalmente un especial agradecimiento a la última compañera predoctoral llegada, Anabel Usié, una gran compañera.

Quiero agradecer de forma especial a mi pareja, Laura Martí, por entender que muchas veces mis deberes para con la tesis han primado por delante de nuestra relación, sin embargo, ella siempre me ha mostrado su apoyo moral a lo largo de estos años.

Y como no, a toda mi familia. A mis padres, Pepe y Teresa, mi hermano Enric, su mujer Andrea, mi sobrina Bruna y a mis tíos, porque a pesar de mi gran dedicación a la tesis, siempre han sabido entender que ellos, para mi, son lo más importante.

Propósito y desarrollo de la memoria

El sector de los videojuegos ha estado en continuo crecimiento desde su aparición. Una muestra de ello es que, por ejemplo, la economía de los EUA tuvo un crecimiento de su PIB entre los años 2001 y 2004 (última época de crecimiento del PIB desde el año 2000) próximo al 3 % [Mun10], en cambio, el mercado de los videojuegos creció en el mismo periodo cerca del 20 % [Cha10].

A principios de milenio, la industria del videojuego tenía unos volúmenes de ingresos, clientes y trabajadores similares a los de la industria del cine y la música [Kir04]. Actualmente, el mercado del videojuego genera más beneficios que la música y el cine. Se ha llegado a una situación donde el videojuego tiene una hegemonía total sobre el resto de competidores de ocio doméstico [Cha10]; muestra de ello, son los datos sobre beneficios de la industria del videojuego, aportados en el año 2002 por [HM02] y [Vak02], comprendidos entre 7 y 9 billones de dólares, sobrepasando así, los beneficios de la industria cinematográfica de Hollywood valorada en 8.1 billones [Sni04].

R. Jayakanthan en el año 2002 compara la influencia de la industria del videojuego en la juventud de hoy en día a la influencia que en décadas anteriores tenían la música, la religión y la política [Jay02]. Este hecho ha sido posible debido a que durante los últimos años los videojuegos han ido expandiendo su ámbito de uso, pasando de la práctica en casa, a un modelo de accesibilidad totalmente ubicuo, donde es posible jugar desde cualquier parte haciendo uso de dispositivos móviles conectados a través de redes de datos (Internet, 3G...). Es por eso que hoy en día los videojuegos ya no son cosa de niños. Por ejemplo, en los EUA un 67 % de los hogares hacen uso de los videojuegos en casa. Según

un estudio realizado por la *Entertaiment Software Education* titulado *Essential Facts About the Computer and Video Game Industry* en el año 2010, la mediana de edad de los jugadores se situaba alrededor de los 34 años. El 47 % de todos los jugadores tenían edades comprendidas entre 18 y 49 años, un 26 % eran mayores de 50 años y tan solo el 25 % eran menores de 18 años.

Dada la importancia económica y social de los videojuegos, más concretamente de los juegos masivos multijugador en red, en el presente trabajo de tesis nos hemos centrado en estudiar y modelar este tipo de aplicaciones con el fin de mejorar la eficiencia de su ejecución. Estos juegos son aplicaciones de ordenador o de videoconsola y son comúnmente conocidos bajo las siglas MMOG (*Massively Multiplayer Online Games*).

En la actualidad, la ejecución de este tipo de aplicaciones se lleva a cabo mayoritariamente mediante arquitecturas basadas en el paradigma Cliente-Servidor. Esto comporta que las infraestructuras de servidores y redes de conexión utilizadas para la ejecución de los MMOG sean frecuentemente actualizadas a costa de desembolsar grandes cantidades de dinero. Sin embargo, la afluencia masiva de nuevos usuarios (o jugadores) de este tipo de aplicaciones sigue comprometiendo día a día la escalabilidad de estas infraestructuras. Es por eso que existe una creciente línea de investigación, basada en solucionar estos problemas de escalabilidad presentes en las infraestructuras que dan servicio a los juegos MMOG.

Una de las posibles soluciones, en las que se está centrando la comunidad científica, es en la ejecución paralela de los juegos MMOG, con el fin de distribuir la carga computacional que se genera y mejorar, de este modo, la eficiencia en su ejecución. Sin embargo, aplicar técnicas de computación paralela a las aplicaciones tipo MMOG es una tarea compleja, ya que este tipo de aplicaciones tienen un grado de dinamismo muy elevado, hecho que dificulta la generación de un modelo abstracto donde se identifiquen fácilmente la procedencia u orden de tareas. Por tanto, dividir este tipo de aplicaciones en un conjunto de tareas con el fin de asignarlas (o mapearlas) a distintos nodos de procesamiento resulta ser todo un desafío para la comunidad científica.

Dentro de este marco, el presente trabajo de tesis se centra en la necesidad de presentar una solución para la ejecución de este tipo de aplicaciones

MMOG en un entorno distribuido, donde el sistema sea capaz de crecer acorde con la demanda de servicios, al mismo tiempo que la calidad de servicio no se vea afectada. En este sentido se ha desarrollado una solución que afronta el problema desde una perspectiva global basada en dos fases. En una primera fase se estudia el comportamiento y las necesidades de recursos de cómputo de las aplicaciones MMOG, y se identifican las características diferenciales de los distintos tipos de juegos que existen. En una segunda fase, y a partir del modelo de comportamiento, se propone una arquitectura híbrida para ser usada en la ejecución paralela de juegos MMOG, aportando escalabilidad y eficiencia. Para ello se desarrollan un conjunto de políticas de asignación de tareas de cómputo, para ser ejecutadas concurrentemente, que explotan el sistema paralelo, contemplando a la vez, las características intrínsecas de los juegos en su modo de ejecución.

Todo este proceso global se estructura en la presente tesis mediante los siguientes capítulos.

- **Capítulo 1**. Se realiza un análisis de las características generales de las aplicaciones tipo MMOG, ejecutadas bajo el modelo Cliente-Servidor y se describen los tipos de juegos MMOG existentes: MMOFPS (*Massively Multiplayer Online First Person Shooter*), MMORPG (*Massively Multiplayer Online Role-Playing Game*) y MMORTS (*Massively Multiplayer Online Real-Time Strategy*). Para cada uno de los tipos de juego se analiza cual es el modelo de servicio que se requiere y sus requisitos técnicos durante la ejecución. Finalmente se presentan los objetivos de trabajo que se abordan en esta tesis.

- **Capítulo 2**. A partir del conocimiento obtenido de las aplicaciones MMOG, se enmarcan las distintas propuestas existentes en la literatura. El abanico de trabajos que trabajan en la ejecución eficiente de los MMOG es muy amplio, sin embargo, estas propuestas difieren en sus objetivos finales. Por este motivo, en este capítulo se clasifican las propuestas en función del modelo de arquitectura que presentan: a) centralizado Cliente-Servidor, b) descentralizado Peer-to-Peer y c) híbrido entre Cliente-Servidor y Peer-to-Peer.

- **Capítulo 3**. Se realiza un estudio experimental de los requisitos de recursos de cómputo de los juegos MMOG. Dicho estudio se centra en los tipos MMOFPS y MMORPG, puesto que son los que tienen mayor interés en la comunidad de jugadores. El análisis que se realiza en este capítulo permite obtener un modelo de comportamiento de dichos juegos, que facilita la definición de nuevas propuestas para la ejecución eficiente de los mismos.

- **Capítulo 4**. Se propone una arquitectura denominada *OnDeGas (On Demand Game Service)* que constituye una plataforma de ejecución eficiente y escalable para los juegos MMOG. Dicha arquitectura se basa en un modelo híbrido Cliente-Servidor y Peer-to-Peer, que representa una solución de compromiso entre la facilidad y fiabilidad de control global del juego que aporta un sistema centralizado y la escalabilidad y bajo coste de un sistema distribuido. En este capítulo se desarrollan además los mecanismos de asignación de tareas de cómputo aplicables a los juegos MMOFPS y su evaluación en *OnDeGas*.

- **Capítulo 5**. En este capítulo se desarrollan los mecanismos de asignación de tareas de cómputo en la arquitectura *OnDeGas*, para ser usada en los juegos del tipo MMORPG. Para ello se realiza un análisis de cuales son los puntos cruciales y distintivos de los MMORPG que hay que contemplar para realizar una distribución de cómputo eficiente. Se realiza además un estudio experimental del rendimiento que nos lleva a valorar positivamente las propuestas realizadas.

- **Capítulo 6**. Finalmente, se presentan las principales conclusiones que se derivan del presente trabajo, junto a las contribuciones a que ha dado lugar el desarrollo del mismo. Asimismo, se indican algunas de las líneas de trabajo que quedan abiertas a partir de los resultados obtenidos.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Capítulo 1

Juegos multijugador masivos en red en un entorno Cliente-Servidor

Hoy en día los videojuegos multijugador masivos en red o MMOG (siglas en inglés de *Massively Multiplayer Online Games*) tienen la cuota más alta dentro del mercado del ocio electrónico debido a la gran expectación que crean en la sociedad. El hecho de que la industria de los MMOG genere más beneficios que industrias tan potentes como la de la música o el cine, sumado al fenómeno *gamer* [1] cada vez más acorde con edades adultas, abandonando así, el fenómeno adolescente, ha provocado que este tipo de aplicaciones estén en el punto de mira de muchas empresas tecnológicas.

El ocio basado en el uso de videojuegos MMOG es un fenómeno social de masas, que además genera importantes beneficios. Por tanto la repercusión socio-económica de este tipo de aplicaciones, con clara tendencia al alza, ha contribuido recientemente a que tanto la sociedad como la comunidad científica ponga su foco sobre ellas. Es por esta razón, que el presente trabajo se centra en la investigación de nuevas propuestas enfocadas a conseguir ejecuciones

[1] Un *gamer* o jugador de videojuegos es el término usado para definir al tipo de jugadores que se caracterizan por jugar con gran dedicación e interés y por tener una gama diversificada de conocimiento sobre videojuegos. Dicho de una manera más simple, un gamer es aquel que se dedica a los juegos una gran cantidad de horas (jugando o informándose).

más eficientes de los MMOG, solucionando algunos de los problemas que se presentan actualmente en su modo habitual de ejecución, basado un modelo Cliente-Servidor.

En este capítulo se clasifican las aplicaciones tipo MMOG, comúnmente ejecutadas bajo el paradigma Cliente-Servidor. Para ello se caracterizan los requerimientos de cómputo y la gestión de los jugadores de cada tipo de juego, con el fin de encontrar solución a problemas típicos que tienen estas aplicaciones. Por último, se subrayan los objetivos principales de este trabajo.

1.1. Juegos masivos multijugador en red: tipos y características generales

Los juegos tipo MMOG se caracterizan por ofrecer al jugador la posibilidad de participar e interactuar en un mundo virtual con un gran número de jugadores -del orden de cientos de miles o incluso millones- simultáneamente (multijugador) conectados a través de la red (en línea), normalmente Internet. Según Constance Steinkuehler [Ste04], un investigador que estudia las connotaciones socio-económicas de los MMOG, la definición acorde a los MMOG sería:

> *Massively multiplayer online games are highly graphical 3-D videogames played online, allowing individuals, through their self-created digital characters or "avatars", to interact not only with the gaming software (the designed environment of the game and the computer-controlled characters within it) but with other players' avatars as well.*

Estas aplicaciones están basadas en el procesamiento de un flujo continuo de datos, que se va generando a medida que servidor y cliente (usuario o jugador) de la aplicación interactúan para actualizar el estado del juego. Por tanto, los jugadores de los MMOG demandan unos compromisos de rendimiento enfocados a conseguir un tiempo de respuesta lo suficientemente corto, que permita realizar una interacción de forma ágil. Dichos compromisos de

rendimiento difieren de los habituales en las aplicaciones que procesan un flujo continuo de datos normalmente acotado. En estas últimas el principal reto se basa en mantener un ratio de procesamiento de datos constante que se pueda equiparar al ritmo de llegada de los datos.

Aún teniendo un marco común de definición de los MMOG, existen características diferenciadoras que hacen que dentro de la familia de los MMOG se distingan, a su vez, los tres tipos de juegos siguientes:

- **MMOFPS**: *Massively Multiplayer Online First Person Shooter Games* o Juegos de Acción en Primera Persona Multijugador Masivos en Línea.
- **MMORPG**: *Massively Multiplayer Online Role Playing Games* o Juegos de Rol Multijugador Masivos en Línea.
- **MMORTS**: *Massively Multiplayer Online Real Time Strategy Games* o Juegos de Estrategia en Tiempo Real Multijugador Masivos en Línea.

Cada tipo de juego tiene unas particularidades distintas que hacen que los requisitos de rendimiento durante la ejecución del mismo varíen de unos a otros. En los siguientes apartados se describen cada uno de ellos.

1.1.1. MMOFPS

Los videojuegos de acción en primera persona multijugador masivos en línea, o MMOFPS (*Massively Multiplayer Online First Person Shooter*), son videojuegos que permiten a miles de jugadores introducirse en múltiples instancias (o partidas) del juego, de forma concurrente a través de Internet, e interactuar entre ellos. Los jugadores de este tipo de juegos buscan una distracción momentánea, de modo que este tipo de jugadores no suelen ser fieles a un juego en concreto. Ejemplos de ofertas de tipo comercial son el *Counter Strike* [Ste] o el *Call of Duty* [Act] y de tipo gratuito el *Urban Terror* [Frob]. Éstos son juegos en los que los objetivos están definidos al inicio de la partida y su duración (dependiendo de la habilidad de los jugadores) es más bien corta (entre decenas de minutos o pocas horas). Cabe destacar que, aunque este tipo de juegos puede soportar gran cantidad de jugadores de forma concurrente, los jugadores están agrupados en partidas de entre 32 y 64 jugadores. La razón de

esta división de jugadores en múltiples partidas es evitar congestiones, puesto que este tipo de juegos precisan de una alta velocidad de movimientos por parte de los jugadores, precisando así, una elevada agilidad de respuesta por parte de los servidores que gestionan el juego. Cabe señalar que en los últimos años está surgiendo una nueva clase de juegos tipo MMOFPS, en los que se permite la interacción entre varios centenares de jugadores en el mismo escenario (como es el caso de PlanetSide [Ubi03]). Sin embargo, esta tendencia de aunar multitud de jugadores de un MMOFPS en un mismo escenario es actualmente muy poco usada, debido al poco éxito comercial y a la multitud de problemas técnicos que tienen este tipo de juegos.

En los juegos MMOFPS no hay una personalización muy marcada del personaje o avatar con el que se va a jugar, tampoco se puede ir progresando en el juego las habilidades que éste tiene dentro del mundo virtual, aunque sí, puede ir aumentado la pericia del usuario.

Modelo de servicio

Los juegos tipo MMOFPS son ejecutados siguiendo el modelo Cliente-Servidor. Los jugadores, que usan el programa cliente, son representados en el mundo del juego a través de un avatar. Los proveedores (normalmente los creadores del juego), no es necesario que guarden el estado de la partida, puesto que ésta tiene una duración limitada, y, una vez que los clientes se desconectan de ella, se elimina del servidor. Por tanto, una de las características más significativas de este tipo de juegos es la división de los jugadores en pequeños grupos, siendo cada uno de los grupos una partida independiente sin comunicación externa. Es decir, los jugadores de una partida están aislados del resto de jugadores de las otras partidas que hay ejecutándose en el servidor.

Una vez que un jugador entra en alguna de las partidas ofertadas, podrá colaborar con algunos jugadores de la misma, en caso de que tengan objetivos comunes.

Características técnicas

Las necesidades técnicas de los juegos tipo MMOFPS recaen básicamente en la necesidad de asegurar una latencia mínima. Los jugadores se mueven y realizan disparos o ataques de forma coordinada y a una alta velocidad. Diferentes valores de latencia entre jugadores pueden determinar el resultado de una partida, puesto que el disparo de un enemigo puede tener un resultado u otro en función del tiempo de respuesta que éste tenga respecto al servidor de la partida.

La importancia de la latencia está ilustrada en la Figura 1.1, donde se muestran dos jugadores A y B que ejecutan acciones en distintos instantes de la partida. Sin embargo, los valores de latencia pueden producir que los efectos de las acciones realizadas tengan lugar en distinto orden al que se produjeron. El *Jugador A* con latencia de 1,5 unidades de tiempo (ut) respecto del servidor, ejecuta un disparo al *Jugador B* en el instante $t = 0$ de la partida. Por otro lado el *Jugador B*, con latencia de 0,5 ut respecto del servidor, ejecuta un disparo al *Jugador A* en el instante $t = 0{,}5$ de partida. Considerando despreciable el tiempo de procesamiento del servidor, las acciones y sus efectos se sucederían según el orden que se indica en el diagrama de la Figura 1.2.

De este modo, el servidor actualizará el estado del juego en el instante $t = 1$, indicando que el *Jugador B* ha disparado al *Jugador A*. Sin embargo, en el instante $t = 1{,}5$ el *Jugador A* quizás ya esté muerto, o el *Jugador B* ya haya variado su posición. Por consiguiente, la acción efectuada por el *Jugador A* en el instante $t = 0$, no tendrá efecto alguno debido a la alta latencia que este jugador tiene respecto del servidor de la partida, en comparación a su enemigo, el *Jugador B*.

Por lo tanto, a través de este sencillo ejemplo, puede apreciarse que en los juegos MMOFPS es indispensable asegurar que todos los jugadores de una partida tengan una latencia que no sobrepase un cierto umbral respecto el servidor de la misma.

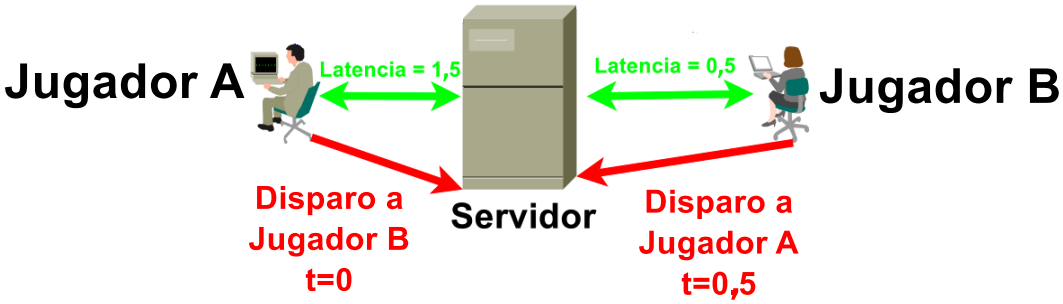


Figura 1.1: Importancia de la latencia en los juegos tipo MMOFPS.

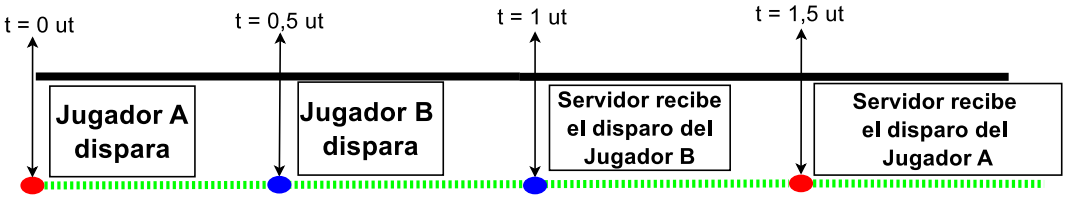


Figura 1.2: Diagrama de tiempo ejemplificando las acciones de la Figura 1.1.

1.1.2. MMORPG

Los videojuegos de rol multijugador masivos en línea o MMORPG (*Massively Multiplayer Online Role-Playing Games*) permiten a miles de jugadores introducirse en un mundo virtual de forma simultánea a través de Internet e interactuar entre ellos. Consisten, en un primer momento, en la creación de un personaje, del que el jugador puede elegir raza, profesión, posesiones, etc. Una vez creado el personaje, el jugador puede introducirlo en el juego e ir aumentando niveles y experiencia en batallas contra otros personajes (jugadores o no jugadores) o realizando diversas aventuras o misiones, habitualmente llamadas búsquedas (conocidas como *quests* en inglés).

Este género de videojuegos difiere de un juego de rol en línea multijugador no masivo, en que estos últimos tienen un número limitado de jugadores, en cambio, los MMORPGs están preparados y elaborados de tal manera que admiten cualquier número de jugadores simultáneos, aunque en la práctica dicho número está limitado por la conexión de red del servidor.

Modelo de servicio

En general, los MMORPG siguen el modelo Cliente-Servidor para su ejecución. Los jugadores, ejecutan el programa cliente y son representados en el mundo virtual a través de un avatar -una representación gráfica del personaje con el que juegan-. Los proveedores (normalmente los creadores del juego), guardan el mundo virtual persistente en el que habitan estos jugadores en uno o varios servidores centrales. Esta interacción entre un mundo virtual, siempre disponible para jugar, y un oscilante flujo mundial de jugadores es lo que caracteriza a los juegos MMORPG.

Una vez que un jugador entra en el mundo virtual puede participar en una amplia variedad de actividades con otros jugadores. Los desarrolladores de MMORPG se encargan de supervisar el mundo virtual y mantenerlo constantemente actualizado ofreciendo a los usuarios un conjunto de actividades y mejoras, para garantizar su interés.

Debido a que la mayoría de los MMORPG son comerciales, los jugadores muchas veces, además de comprar el programa cliente, deben pagar un precio mensual para acceder al mundo virtual. Existen juegos en línea gratuitos en Internet (*PlaneShift* [Corb] o *Runes of Magic* [Froa]), aunque la calidad de producción de éstos suele ser menor comparada con sus homólogos de pago. Asimismo, últimamente muchos de estos juegos de pago optan por ofrecer un servicio gratuito con posibilidad de realizar pequeños pagos con el objetivo de obtener mejoras de accesorios y otros. Por su parte, los juegos de acceso gratuito encuentran fuentes de financiación en los micropagos, es decir, compras con pequeñas cantidades de dinero real, por parte de los usuarios, de accesorios virtuales para el juego (por ejemplo, pociones de salud para el avatar, mascotas, elixires, ropajes, etc.).

Los MMORPG son inmensamente populares, con varios juegos comerciales superando los millones de suscriptores como *Lineage I* y *Lineage II* [NCS] o *World of Warcraft* [Bli].

Características técnicas

La mayoría de los MMORPGs necesitan de gran cantidad de recursos de desarrollo para superar los obstáculos logísticos asociados con un esfuerzo de producción tan enorme. Los juegos en línea requieren de la gestión de mundos virtuales, significativos requisitos de hardware (servidores y ancho de banda) y un departamento de soporte dedicado. A pesar de los esfuerzos de los desarrolladores -conscientes de estos temas- las reseñas frecuentemente citan, como los problemas más comunes de los juegos de este género, poblaciones no óptimas (mundos superpoblados o casi abandonados), así como la existencia de cierto retardo (o *lag*) causado por latencias demasiado elevadas y pobre soporte técnico.

Además de los desafíos propios de hacer un videojuego, los diseñadores también tienen que enfrentarse a problemas únicos de este género, de los cuales resaltamos los siguientes:

A *Persistencia del estado del mundo virtual.* Este tipo de juegos está dotado de un mundo virtual donde los personajes pueden conectarse cuando les apetezca. De forma inherente, el juego está obligado a una ejecución constante del mundo virtual, dado que se brinda la posibilidad a cualquier jugador a conectarse cuándo lo desee. Por tanto, esta característica implica tener un sistema muy robusto, tolerante a fallos y que asegure la persistencia de los datos, tanto de los jugadores como del estado global del mundo virtual, que es común para todos los jugadores.

B *Transiciones transparentes entre zonas del mundo virtual.* Los mundos virtuales tienen una gran superficie, debido a que deben soportar muchos jugadores conectados de forma concurrente. Para evitar aglomeraciones de jugadores, éstos se distribuyen a lo largo del mundo virtual. Existen dos métodos para gestionar este mundo virtual:

- *Mundo virtual dividido en zonas.* El mundo virtual está compuesto de zonas regionales. La transición de una zona a otra hace uso de un proceso conocido entre los jugadores como "zoneado" (*zoning* en inglés), mediante el cual los datos son cargados para la zona destino

en particular. Como consecuencia de esta técnica, el jugador acusa la percepción de cambio de zona.

- *Mundo virtual continuo.* El mundo virtual es continuo, uniforme "sin costuras" (*seamless* en inglés). En este tipo de mundos virtuales se permite viajar de un punto del mundo a otro sin zonas de transición. Sin embargo, en muchos de estos juegos existen ciertos momentos en los que hay períodos de carga, como por ejemplo cuando se ejecutan instancias auxiliares del juego principal, o cuando se debe viajar a zonas muy lejanas del mundo virtual.

Ambos métodos tienen sus limitaciones técnicas, principalmente relacionadas con los requerimientos de memoria o de uso de ancho de banda por parte del programa cliente. Por lo tanto, el modo de gestionar el mundo por parte de los servidores afectará a la calidad de servicio del juego. No obstante la característica técnica que es indispensable asegurar en los juegos MMORPG, es la persistencia e integridad de los datos de los jugadores y del mundo virtual a lo largo de las distintas sesiones.

1.1.3. MMORTS

Los videojuegos de estrategia en tiempo real multijugador masivos en línea o MMORTS (*Massively Multiplayer Online Real-Time Strategy*) son videojuegos de estrategia en tiempo real, que al igual que los MMORPG, permiten a miles de jugadores interactuar entre sí en tiempo real dentro de un mundo virtual de forma simultánea a través de Internet.

Se trata de videojuegos en los que el jugador debe desempeñar el papel de un líder, como un rey o caudillo en una ambientación fantástica o un general en una ambientación de ciencia ficción. Esto supone ocuparse de asuntos tales como la gestión de recursos económicos, la diplomacia o la formación de un ejército. El juego se desarrolla en un mundo persistente que evoluciona independientemente de que los jugadores estén o no conectados.

Cada jugador controla decenas de unidades (en contraste con los MMORPG) y puede luchar con o contra cualquier número de jugadores de forma simultánea. Los jugadores reciben información del entorno que rodea a sus uni-

dades, que pueden estar ubicadas en zonas muy separadas dentro del mundo virtual. Debido a que los usuarios constan, no de un avatar, sino que tienen a sus órdenes múltiples unidades, la cantidad de información que debe ser transferida a los clientes del juego para mantener el estado actualizado constantemente es mucho mayor que la cantidad de información que se debe actualizar en el resto de tipos de los MMOG. Hacer frente a este enorme flujo de información constituye uno de los mayores desafíos de los juegos MMORTS [KHC03].

Por tanto, se puede concluir, que los MMORTS tienen cierto parecido a los MMORPG. Ambos tienen multitud de jugadores, en un mismo mundo virtual que está siempre ejecutándose. Sin embargo, mayoritariamente los MMORTS son ejecutados desde un navegador web, donde la información es mostrada vía una interfaz web, mientras que los MMORPG tienen gráficos 3D que precisan de un cliente software instalados en el ordenador del jugador. Otra de las diferencias notables, es que los MMORTS despiertan menor interés que los MMORPG puesto que son menos atractivos visualmente.

Modelo de servicio

Los MMORTS siguen el modelo Cliente-Servidor, y son ejecutados comúnmente desde un navegador web. La mayoría de juegos de éxito de este género (*Ogame* [Gamb] o *Ikariam* [Gama]) están implementados con el lenguaje de programación *PHP*, requiriendo únicamente un navegador común. Puesto que no se necesita realizar instalaciones extra para jugar, este tipo de videojuegos ha crecido a la par que lo hacían los entornos web.

La principal diferencia con los videojuegos de videoconsola o de ordenador es que los MMORTS son normalmente independientes de la plataforma, basados exclusivamente en tecnologías usadas por el cliente o jugador (normalmente llamadas plugins). Habitualmente, todo lo que se requiere para jugar a un MMORTS es un navegador web y el plugin apropiado.

A esta nueva vertiente de juegos MMORTS se los ha bautizado recientemente como juegos tipo *PBBG* (*Persistent Browser-Based Game*). En ellos el progreso en el juego se logra a través de múltiples sesiones. Debido a unos requerimientos técnicos asumibles y a la facilidad de que los usuarios se co-

necten desde múltiples lugares con un simple navegador web y una conexión a Internet, estos juegos han alcanzado una gran popularidad en los últimos años.

Características técnicas

Del mismo modo que las otras categorías de MMOG, los MMORTS presentan algunos problemas comunes a todas ellas relacionados principalmente con la escalabilidad que se deriva del aumento de número de jugadores concurrentes.

A pesar de que los MMORTS incluyen el término *real-time strategy* en su nombre, es importante remarcar que la latencia y la interactividad de los jugadores no supone un problema ya que están basados en sistemas de órdenes por turnos. Una vez un jugador ha ejecutado una orden, ésta precisará de una ventana temporal para que tenga validez. A modo de ejemplo, si ordenamos la construcción de una nave espacial para nuestra flota, la orden no se ejecutará de forma inmediata, sino que el servidor nos especificará que se precisan "T" unidades de tiempo para disponer de la nave. Del mismo modo, un ataque a un enemigo requerirá un tiempo de procesamiento o turno para que se realice.

Este sistema de turnos permite a los servidores controlar de forma directa el volumen de acciones que se suceden en el juego, y por tanto, gestionar la carga computacional del mismo.

En los MMORTS, la característica técnica más importante que es imprescindible abordar, al igual que ocurre con los MMORPG, es la persistencia de los datos, puesto que aunque sea a través de un navegador, las partidas continúan a los largo de las diferentes sesiones de conexión al juego.

Una vez caracterizados a grandes rasgos los tres tipos de juegos MMOG: MMOFPS, MMORPG y MMORTS, en la siguiente sección se introducirán las principales características de la arquitectura Cliente-Servidor, que tal y como se ha comentado, es la base sobre la que actualmente se suelen ejecutar las aplicaciones MMOGs.

1.2. Arquitectura Cliente-Servidor

En esta sección se introducen los conceptos más importantes de la arquitectura Cliente-Servidor. Puesto que ésta es la arquitectura comúnmente usada para la ejecución de los juegos MMOG, es esencial conocer los aspectos principales de su funcionamiento, de cara a posibles propuestas de optimización para la ejecución de dichos juegos.

El término Cliente-Servidor fue usado por primera vez en 1980 para referirse a ordenadores en red [Mel04]. Su funcionamiento es el siguiente: precisa de una máquina cliente, que demanda un servicio, y una máquina servidor que ofrece uno o más servicios a clientes que los reclaman. Nótese que no necesariamente tiene que tratarse de máquinas diferentes; es decir, un ordenador por sí solo puede ser ambos, cliente y servidor, dependiendo del software de configuración.

Desde el punto de vista funcional, se puede definir la computación Cliente-Servidor como una arquitectura centralizada que permite a los usuarios finales obtener acceso a la información de forma transparente aún en entornos multiplataforma [Mel04].

En el modelo Cliente-Servidor, el cliente envía un mensaje solicitando un determinado servicio a un servidor (hace una petición). El servidor, una vez procesada la petición, envía uno o varios mensajes con la respuesta (provee el servicio), tal y como se ilustra en la Figura 1.3.

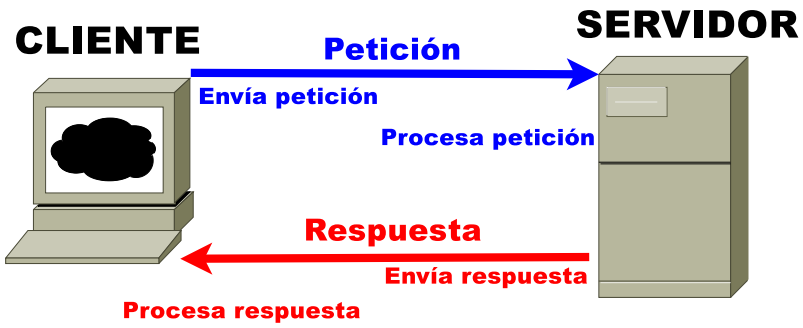


Figura 1.3: Arquitectura Cliente-Servidor.

Puede ser que por motivos de localización geográfica, o bien por motivos técnicos, un único servidor no sea capaz de procesar todas las peticiones de servicio que demandan los clientes. Por esta razón, se pueden agrupar varios

servidores para dar servicio a toda la demanda producida por las múltiples conexiones de clientes. Cabe destacar, que los clientes pueden conectarse a uno u otro servidor de forma transparente, en función de los parámetros de configuración que haya implementados en este grupo de servidores (por ejemplo parámetros como la congestión, latencia, información que gestiona, etc.). Este concepto puede visualizarse en la Figura 1.4, donde clientes de alrededor del mundo, buscan un mismo recurso, pero las peticiones son enviadas a distintos servidores que controlan el mismo recurso a lo largo de todo el mundo.

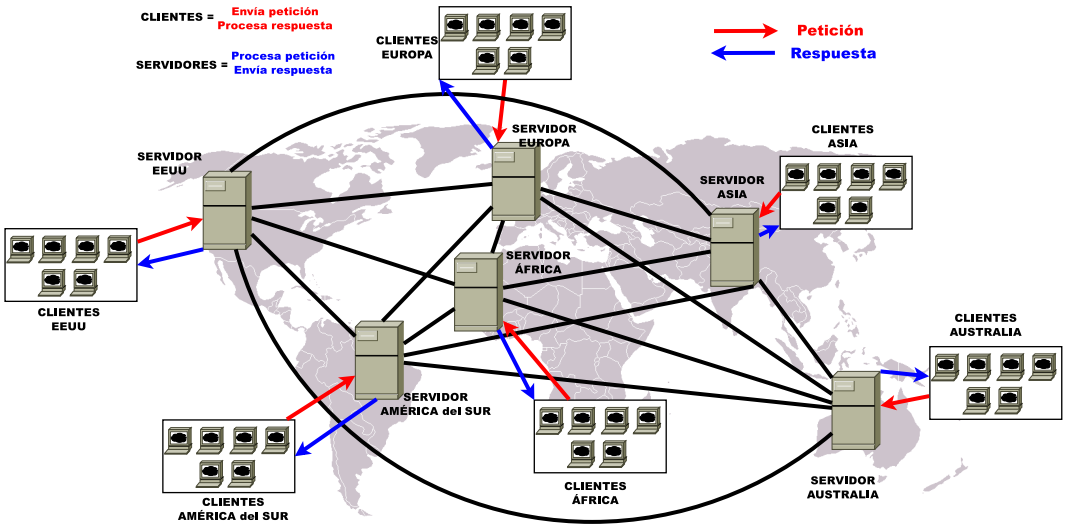


Figura 1.4: Arquitectura Cliente-Servidor con varios clientes conectándose a múltiples servidores.

Partiendo de la Figura 1.4, podemos definir el concepto de piscina de servidores. Cada uno de los servidores distribuidos a lo largo del mapa del mundo, a su vez, puede estar formado por un grupo de servidores, esta técnica es conocida con el concepto de "piscina de servidores" (*clúster* o *pool of servers* en inglés) y es mostrado en la Figura 1.5. En esta figura, un conjunto de clientes de Australia, se conectan al servidor de su zona, estando éste formado por un conjunto de servidores conectados entre ellos con redes de altas prestaciones y bajas latencias, con el fin de prestar servicio con la máxima calidad posible.

Los juegos MMOG se ejecutan tradicionalmente en una arquitectura Cliente-Servidor parecida a la que se muestra en la Figura 1.5, en la que se refleja como se precisa una piscina de servidores para afrontar el volumen de peticiones de

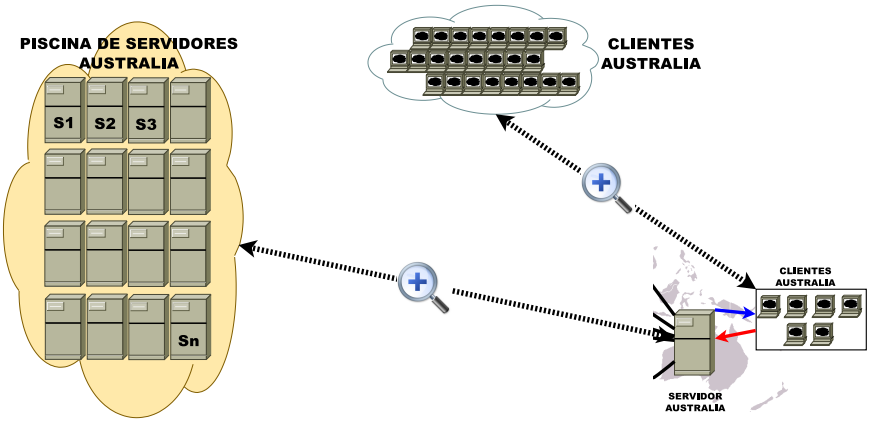


Figura 1.5: Arquitectura Cliente-Servidor con piscina de servidores.

los clientes, que en el caso de los MMOG son muy elevadas y concurrentes, hecho que provoca gran estrés a los servidores encargados de gestionar los servicios a prestar.

1.3. Objetivos del trabajo

A lo largo del capítulo hemos puesto de manifiesto el creciente interés económico y social que las aplicaciones MMOGs están teniendo durante los últimos años. También hemos repasado, a grandes rasgos, los principales problemas que este tipo de aplicaciones plantean en su modo de ejecución habitual sobre arquitecturas Cliente-Servidor. Si ponemos en común los problemas que se producen en todos los tipos de juegos MMOGs, se identifica una problemática coincidente basada en la gestión eficiente de la carga de cómputo generada por la afluencia masiva de jugadores, en otras palabras, la escalabilidad de los servidores que gestionan los juegos.

Para afrontar el problema de la escalabilidad, sin que esto suponga una inversión importante en la infraestructura del servidor del juego, en esta tesis nos marcamos como objetivo la propuesta de un nuevo paradigma de cómputo y de gestión de los juegos MMOG. Dentro de los tipos existentes, descritos en la sección anterior, nuestro trabajo tiene como objetivo centrarse en los

juegos MMOFPS y MMORPG. Éstos son los tipos de juego más usados y, por lo tanto, los que tienen mayor influencia en los jugadores; siendo además los que plantean mayores problemas de escalabilidad. Por tanto, en el marco de este objetivo global, se proponen los siguientes objetivos concretos a estudiar y desarrollar:

- Definir el modelo de comportamiento de las aplicaciones MMOG, y en concreto de sus categorías MMOFPS y MMORPG. Esto ha de permitir identificar los requisitos de recursos de cómputo de dichos juegos, en función del número de jugadores que acceden a ellos de forma concurrente.
- Proponer un nuevo paradigma de cómputo para la ejecución de los juegos MMOG que contemple el modelo obtenido.
- Definir una arquitectura apropiada para la ejecución de juegos MMOG, que compagine una parte centralizada con una distribuida, que permita asumir el paradigma de ejecución que corresponde y aporte escalabilidad.
- Desarrollar nuevas políticas de distribución de cómputo de los juegos en la arquitectura definida. Dichas políticas tendrán en cuenta las características específicas del modelo de comportamiento de los tipos de juego bajo estudio.
- Analizar el rendimiento y la escalabilidad del sistema propuesto y evaluar la calidad de servicio que se obtiene, con especial inciso en el estudio de la latencia y de la tolerancia a fallos.
- Estudiar el impacto de la heterogeneidad de los ordenadores del área distribuida en el rendimiento de las políticas de distribución de cómputo desarrolladas.

Capítulo 2

Estado del arte

En este capítulo se describen las principales aportaciones de la comunidad científica en el campo del diseño de entornos de ejecución de aplicaciones tipo MMOG, centrando nuestro foco principalmente en las soluciones que involucran arquitecturas distribuidas, bien sean puras o híbridas. Cabe destacar que los distintos trabajos de la literatura abordan el problema de la ejecución de los MMOG desde perspectivas muy diferentes como son la prevención de trampas, la propagación del estado del juego, la persistencia de los datos, etc. Para describir las propuestas estudiadas, éstas se han clasificado acorde con el tipo de arquitectura utilizada por los autores de cada trabajo. En concreto, se han distinguido los tres tipos de arquitectura siguientes:

1. Modelo centralizado Cliente-Servidor.
2. Modelo descentralizado puro Peer-to-Peer.
3. Modelo híbrido Cliente-Servidor y Peer-to-Peer.

2.1. Soluciones basadas en un modelo centralizado Cliente-Servidor

Dentro del abanico de propuestas y soluciones revisadas en la literatura, para la ejecución eficiente de juegos masivos multijugador en línea, hay una amplia gama de trabajos que afrontan los retos de ejecución de los juegos

MMOG, haciendo uso del paradigma de ejecución Cliente-Servidor. Dichos trabajos aplican al modelo cliente-servidor una serie de técnicas o heurísticas con el fin de mejorar uno o más aspectos de la ejecución, ofreciendo de este modo un mayor rendimiento de la aplicación.

A continuación vamos a describir las propuestas que se han realizado en este sentido. Cabe destacar que dichas propuestas giran en torno a la técnica utilizada para realizar la división del mundo virtual en regiones. Por tanto, en el siguiente apartado comentaremos los trabajos realizados siguiendo esta línea.

2.1.1. División del mundo virtual

Estas propuestas se basan en la división del mundo virtual en pequeñas regiones, llamadas celdas. Cada una de las celdas es gestionada por un servidor distinto. Sin embargo, el tamaño y asignación de estas celdas no se define de forma estática, puesto que el dinamismo de este tipo de aplicaciones implica una división y asignación dinámica de las celdas a los servidores del juego. Tal y como comentan Marios Assiotis et al. en [AT06] o Chen et al. en [CWD+05], con una división estática no es posible reaccionar a concentraciones de jugadores creadas de forma espontánea (conocidas como *peakloads* o *hotspots*); fenómenos impredecibles e inevitables en las aplicaciones tipo MMOG. Por tanto, una división y distribución dinámica de las celdas permite la migración de las mismas entre servidores, en tiempo de ejecución, en función de la carga de cómputo que gestiona cada uno de los servidores. Esto permite gestionar las concentraciones espontáneas de jugadores. Un ejemplo conceptual del tipo de arquitectura que engloba este tipo de propuestas se ilustra en la Figura 2.1.

Una vez visto que la división y distribución de las porciones del juego o celdas debe realizarse de forma dinámica, la comunidad científica ha ido estudiando el tamaño óptimo de las celdas que integran el mundo virtual.

Determinación del tamaño de las celdas

El tamaño de las celdas en las que se divide el mundo virtual puede afectar a diversos aspectos del juego, por ejemplo, al número total de balanceos a realizar

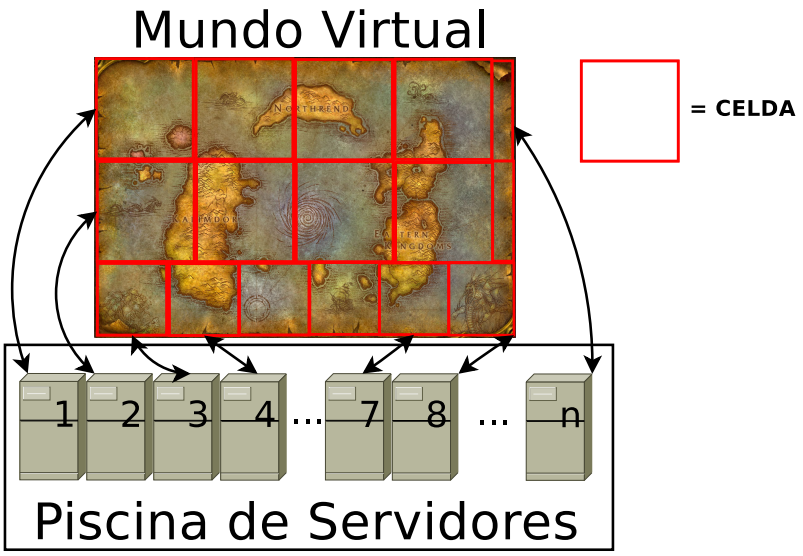


Figura 2.1: División MMOG en celdas/regiones/zonas.

una vez uno o más servidores llegan a un estado de sobrecarga. Determinar el tamaño de las celdas, por tanto, es uno de los puntos más importantes dentro de las propuestas englobadas bajo el marco del Cliente-Servidor. Una división del mundo virtual en pocas celdas de gran tamaño, puede implicar que grandes concentraciones de jugadores estén gestionadas por un mismo servidor, lo cual significa que los servidores deben gestionar una mayor carga de cómputo. Por contra, dividir el mundo virtual en un mayor número de celdas de menor densidad de jugadores, permite una asignación más flexible de estas micro-celdas entre todos los servidores del juego, habilitando, de este modo, un balanceo de la carga entre servidores más ecuánime, repartiendo los jugadores a lo largo de todos lo servidores.

Bärt De Vieeschauwer et al. proponen en [DVVDBV+05], dividir el mundo virtual en un conjunto de micro-celdas, es decir, celdas de un tamaño reducido, con el fin de asignarlas a los servidores. El esquema conceptual de la arquitectura se ilustra en la Figura 2.2. Los autores de esta propuesta comentan que es más eficiente y viable que una división en celdas de mayor tamaño, puesto que la asignación y migración de las micro-celdas entre servidores es más eficiente, ya que se manejan pequeñas porciones de jugadores (cómputo). Sin embargo, cabe destacar que fragmentar el mundo virtual de un modo excesivo en una

gran cantidad de micro-celdas, puede tener un efecto contraproducente. Esto es debido a que pequeños cambios en la carga de un servidor, causados por el dinamismo de los jugadores de este tipo de juegos, pueden provocar la migración de un gran conjunto de micro-celdas entre servidores, lo cual implica un incremento notable de comunicaciones entre servidores. De este modo la calidad de servicio que los jugadores experimentan a lo largo de todo el mundo virtual puede verse degradada.

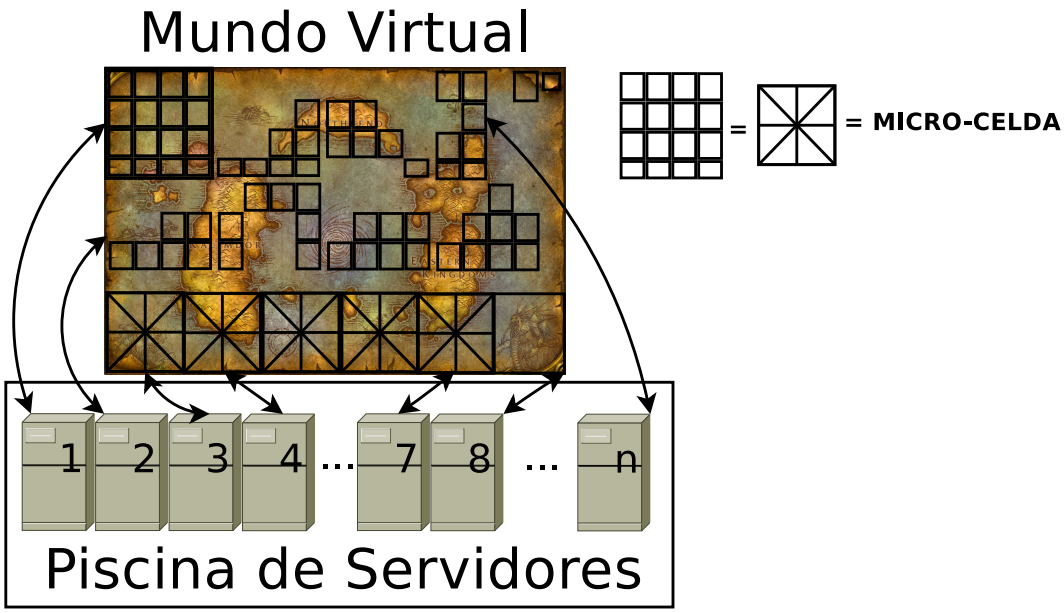


Figura 2.2: División MMOG en micro-celdas.

Una de las desventajas más notables de las arquitecturas centralizadas, a la que la comunidad científica debe hacer frente, se centra en la gestión de la base de datos del juego, continuamente usada con el fin de actualizar el estado del mismo, y por tanto, dónde se generan fácilmente cuellos de botella. Las propuestas basadas en celdas de mayor tamaño presentadas en la Figura 2.1, constan con un esquema de base de datos como el que se ejemplifica en la Figura 2.3, donde cada celda tiene una base de datos propia en la que se actualizan los estados de juego de sus jugadores. Tener una base de datos para cada celda descongestiona el acceso a una única base de datos por parte de todos los servidores del juego. Sin embargo, cada vez que un jugador migra entre servidores, debido a que se ha movido entre celdas gestionadas por distintos servidores,

se deberá realizar un traspaso de información entre las bases de datos de los servidores implicados. Por tanto, además de incrementar el volumen de comunicaciones entre servidores, también resulta complejo asegurar la consistencia de datos entre las distintas bases de datos existentes en la arquitectura. Otras soluciones basadas en micro-celdas [DVVDBV+05] siguen teniendo el mismo tipo de problemas potenciales de consistencia de datos entre las distintas bases de datos de regiones gestionadas por distintos servidores. Por tanto, el tamaño de la celda no tiene incidencia directa sobre la persistencia de los datos del juego.

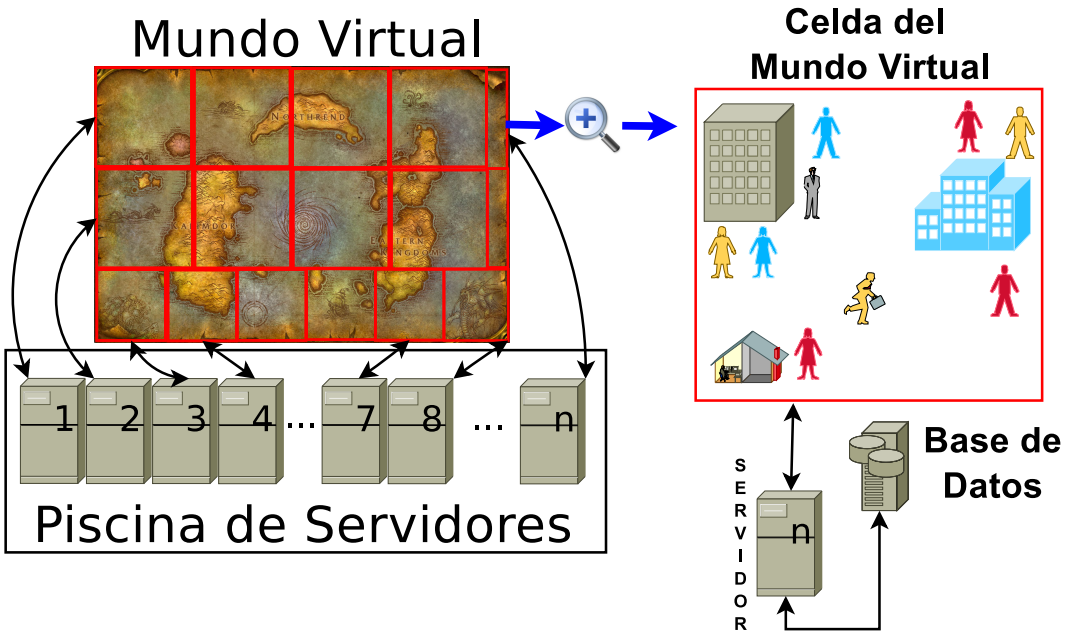


Figura 2.3: División MMOG en celdas; cada celda con un servidor que gestiona la base de datos.

En el campo de la división y distribución de la carga de cómputo entre servidores, existen numerosos trabajos, cuyas propuestas han sido extendidas y aplicadas a la gestión de la carga de cada una de las celdas o micro-celdas. Estas técnicas se basan en algoritmos o heurísticas que optimizan el rendimiento de las aplicaciones sin afectar a la calidad de servicio. En este ámbito, Lu et al. proponen en [LPM06] una serie de técnicas para balancear la carga entre servidores gestores de un juego masivo multijugador en línea. Los autores del artículo comentan que una solución ideal de balanceo de carga es aquella que

contempla una carga ecuánime entre todos los servidores del juego, con el fin de evitar tener servidores saturados y por contra otros servidores ociosos. Del mismo modo que Lu et al., Duong et al. proponen dos tipos de algoritmos [DZ03] para distribuir la carga de cómputo: algoritmos proactivos y algoritmos reactivos. Los primeros están basados en compartir la carga entre servidores al inicio del juego, con el fin de distribuir en cada servidor el mismo número de jugadores. Por contra, los algoritmos reactivos, se aplican una vez el sistema se encuentra desbalanceado. Sin embargo, todas las técnicas reactivas, encargadas de asegurar una carga ecuánime entre servidores, son aplicadas con frecuencia debido al alto dinamismo de este tipo de aplicaciones, lo cual conlleva un coste de cómputo adicional. De este modo se introduce un *overhead* considerable que reduce la efectividad de los propios algoritmos reactivos y la calidad de servicio que experimentan los jugadores.

Determinación del Área de Interés (AOI)

Otro de los problemas afrontado por la comunidad científica es la gestión de la actualización de los estados del juego. Uno de los primeros problemas se halla en determinar a qué jugadores se enviará la actualización del juego y a cuales no. Para decidir cuántos jugadores están interesados en una actualización se definen, en todo tipo de juegos, lo que se conoce como Área de Interés (en inglés *Area of Interest* (AOI)). El área de interés define el área física dentro del mundo virtual, en la que un jugador puede percibir y ejecutar una serie de acciones sobre un conjunto de objetos o jugadores. Dicho de otro modo, es el radio de acción de un jugador. Dentro de esta área de interés o radio de acción, un jugador podrá:

1. "Ver" las acciones que se producen con los objetos y jugadores existentes dentro de este radio de acción.

2. "Ver" las acciones que realizan otros jugadores y personajes no jugadores (en inglés *Non Player Characters* (*NPCs*)) dentro de este radio de acción.

3. "Ejecutar" un conjunto de acciones sobre los objetos y jugadores que están dentro de su radio de acción.

Por tanto, el área de interés de un jugador determina que actualizaciones del estado del juego son relevantes o de interés para el jugador en cuestión. En la Figura 2.4 podemos ver representados gráficamente tres áreas de interés mediante círculos de líneas de distinto estilo y colores para los jugadores A, B y C respectivamente. Podemos ver como el jugador C se encuentra en el área de interés de los jugadores A y B, sin embargo estos últimos no tienen ningún punto de intersección en sus áreas de interés. Por tanto, los jugadores A y B recibirán notificaciones del estado del juego del jugador C y viceversa. En cambio, el jugador A no conocerá nada relacionado con los movimientos que realiza el jugador B y viceversa.

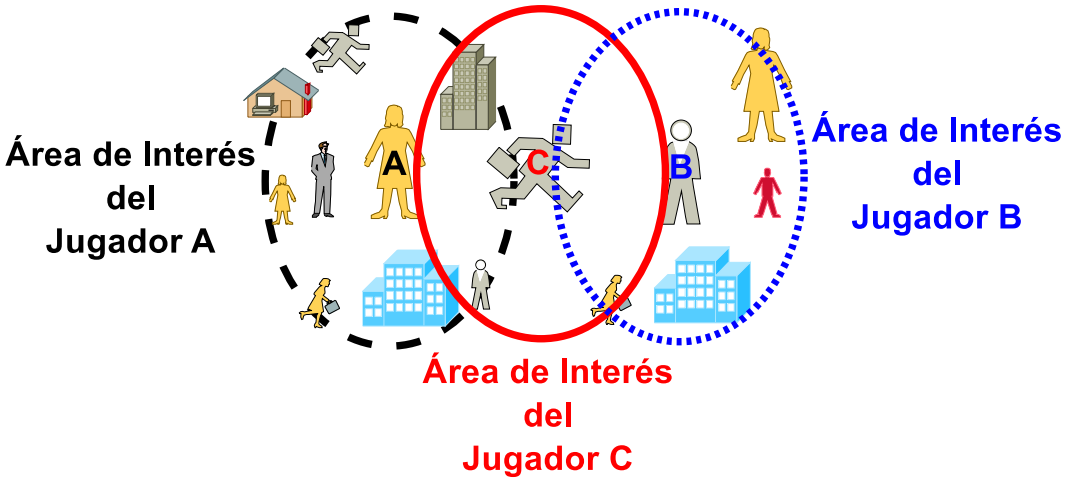


Figura 2.4: Área de Interés de tres jugadores de un MMOG.

El problema al que queremos hacer referencia está relacionado con las áreas de interés de los jugadores y el impacto negativo que pueden llegar a tener en función del modo en que se divida el mundo virtual. En caso que jugadores pertenecientes a distintas celdas, que por tanto, estarán gestionadas por distintos servidores, compartan ciertos intereses debido a su proximidad en el juego (normalmente los bordes de las celdas), el volumen de comunicaciones existente entre servidores será muy elevado, con el consecuente impacto negativo sobre la calidad de servicio que experimentan los jugadores. Este escenario comentado a modo de ejemplo, está representado gráficamente en la Figura 2.5 donde se reflejan tres áreas de interés. El jugador A se encuentra en la celda "X" gestionada por el servidor 1, el jugador B se halla en la celda "Y"

gestionada por el servidor 2 y el jugador C está en el borde de ambas regiones. Que A y B tengan áreas de interés colindantes entre ellos así como con el jugador C, sumado al hecho que A y B pertenecen a distintas celdas gestionadas por distintos servidores, implicará un alto volumen de comunicaciones entre servidores, lo que provocará una degradación de la calidad de servicio debido al *overhead* introducido por estas comunicaciones.

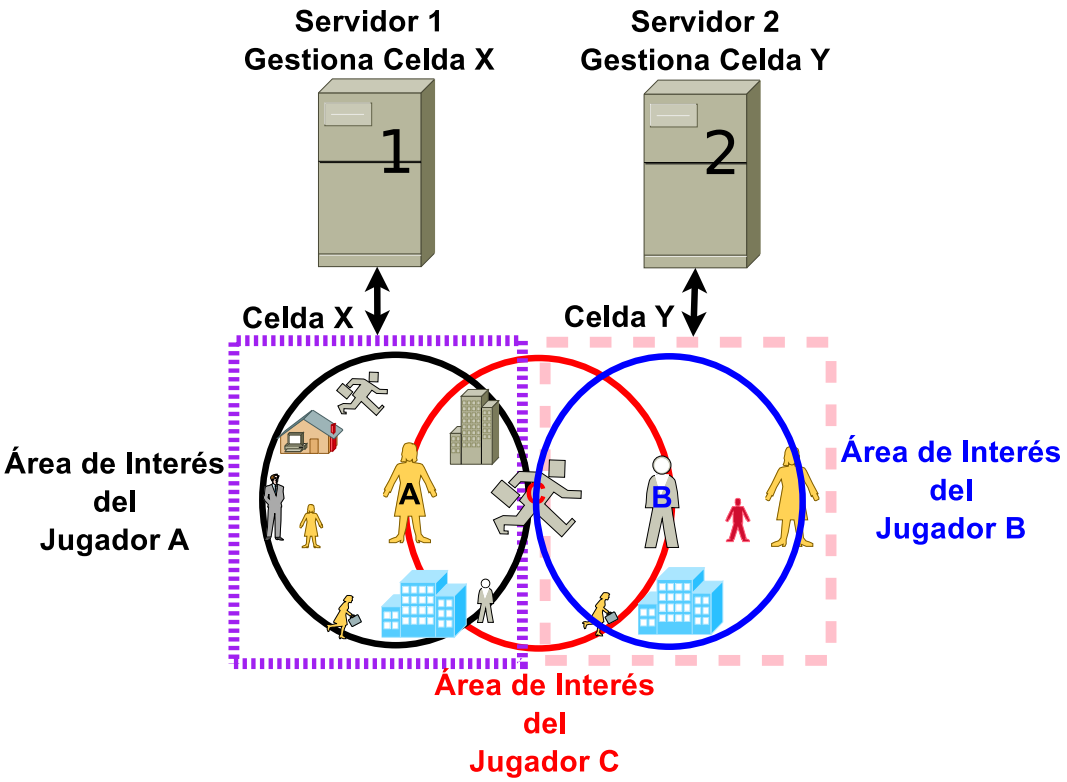


Figura 2.5: Dos servidores gestionando dos celdas con tres Área de Interés (jugadores A, B y C) de un MMOG cualquiera.

Por tanto, la definición del área de interés de los jugadores tendrá un impacto en el rendimiento del juego. Si bien a menor Área de Interés, menor es el *overhead* de comunicaciones del juego -especialmente la comunicación entre distintos servidores-, también es menor la satisfacción del jugador, puesto que se minimiza su posibilidad de interacción con su entorno. Por tanto, al ser el tamaño del AOI una opción de diseño interna del juego, que afecta al juego de forma directa, podemos concluir que se trata de un parámetro no modificable

[AS08].

A continuación, vamos a presentar las propuestas más relevantes que se han presentado, haciendo uso de una arquitectura totalmente distribuida, como es el caso de la arquitectura Peer-to-Peer.

2.2. Soluciones basadas en un modelo descentralizado puro Peer-to-Peer

Dentro del abanico de propuestas y soluciones revisadas en la literatura, para la ejecución eficiente de juegos masivos multijugador en línea, hay una amplia gama de artículos que intentan solucionar ciertos puntos flacos de la ejecución de los juegos tipo MMOG, haciendo uso del paradigma de ejecución Peer-to-Peer. Es decir, un modelo totalmente distribuido que constituye la antítesis del modelo centralizado basado en una arquitectura Cliente-Servidor.

Antes de revisar las propuestas pertenecientes a este campo de la literatura, debemos conocer los distintos desafíos que existen cuando se pretende migrar una aplicación tipo MMOG de un entorno centralizado a uno descentralizado puro.

2.2.1. Problemas en el diseño de MMOG descentralizados

Lu Fan et al. comentan en [FTT10] los puntos necesarios a tener en cuenta cuando se proponen arquitecturas totalmente distribuidas (Peer-to-Peer) para la ejecución de juegos masivos multijugador en línea. Estos puntos son los siguientes:

- Gestión de las Áreas de Interés de los jugadores (AOI).
- Propagación de los eventos del juego.
- Gestión de los Non Players Characters (*NPCs*) del juego.
- Persistencia de los datos del juego.

- Técnicas de mitigación de trampas (cheating).
- Mecanismos de incentivos para los jugadores.

A continuación vamos a realizar una explicación sobre cada uno de los puntos listados, y repasaremos las técnicas que, de forma común, son usadas para afrontar los problemas derivados.

Gestión de las Áreas de Interés de los jugadores (AOI)

El primer requerimiento para un MMOG que es ejecutado a través de una arquitectura totalmente descentralizada, es decir, a través de una red Peer-to-Peer, es gestionar el mundo virtual que comparten cientos de miles de jugadores sin hacer uso de un servidor. La gestión de las áreas de interés es un campo de investigación típico, que se inició a mediados de la década de los 90 por Macedonia et al. en el artículo [MZP+94]. La gestión del área de interés de los jugadores involucra dos aspectos:

1. Un jugador no necesita "saber" lo que sucede a lo largo de todo el mundo virtual, puesto que sólo aquellos eventos que suceden cerca de él son los que le afectan.
2. Los jugadores tienen una capacidad de movimiento y una capacidad de captar eventos limitado.

Por tanto, la cantidad de información que un jugador necesita y "es capaz de saber" está determinada por los dos ítems enumerados con anterioridad. El jugador, por consiguiente, necesita estar informado de los eventos que suceden dentro de su Área de Interés. Dentro de la gestión de las áreas de interés existen tres alternativas: (a) el modelo espacial (que puede gestionarse mediante diagramas *Voronoi* [BA08, HL04]), (b) el modelo de regiones basadas en la publicación/suscripción y (c) un modelo híbrido entre ambos.

(a) *Modelo espacial*. Con el modelo espacial [BF93, DTHK05, MKMA04] (ver Figura 2.6), los jugadores recibirán las actualizaciones de los objetos y jugadores que estén dentro de su área de interés. Por tanto, siguiendo el ejemplo de la imagen representada en la Figura 2.6, los únicos jugadores que recibirían

actualizaciones de estado del juego de otro jugador, serían los jugadores 2 y 3, al tener áreas de interés colindantes.

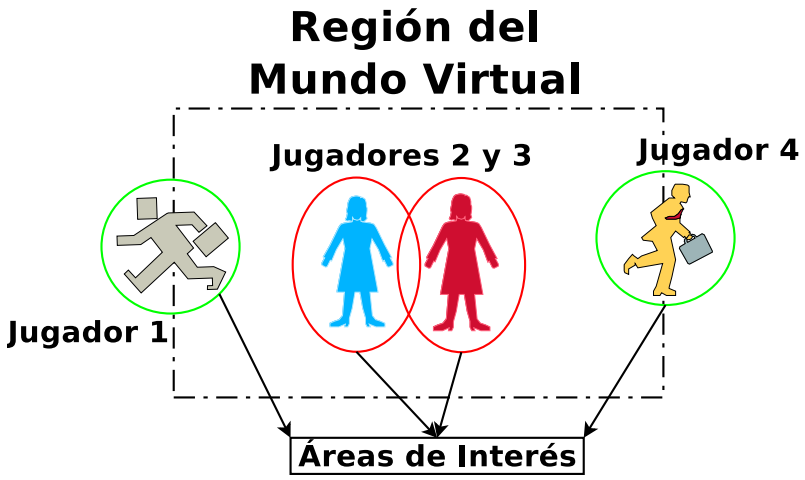


Figura 2.6: Gestión de las AOI mediante un modelo espacial.

(b) *Modelo basado en la publicación y suscripción de eventos.* En dicho modelo, ilustrado en la Figura 2.7, los jugadores recibirán las actualizaciones a las cuales se hayan suscrito [BKV06]. En este punto, la publicación y suscripción se realiza a nivel de regiones en las que está dividido el mundo virtual. Sin embargo, también es posible que los jugadores de las regiones puedan limitar estas actualizaciones a un conjunto de jugadores de la región. Por tanto, siguiendo el ejemplo de la imagen representada en la Figura 2.7, los jugadores 1, 2, 3 y 4 recibirán información unos de otros, ya que se encuentran dentro de la misma región. Sin embargo, una variante de este modelo permitiría, por ejemplo, que el jugador 2 estuviera suscrito a actualizaciones de primer nivel, es decir, a los eventos que se suceden a su alcance inmediato (por ejemplo notificaciones del jugador 3) .

(c) *Modelo híbrido.* Cada una de las alternativas anteriores, tiene sus puntos fuertes y débiles. El modelo espacial parece ser más adecuado, debido a su perfecto acoplamiento con técnicas de propagación de los eventos del juego que comentaremos al finalizar este apartado. Sin embargo, debido a la inexistencia de un servidor central que gestione las comunicaciones, la actualización de las "vistas" que los jugadores deben tener del mundo virtual acarrea un *overhead* de comunicaciones demasiado elevado para que un jugador

Figura 2.7: Gestión de las AOI mediante un modelo basado en publicación y suscripción.

lo gestione. Por consiguiente, para evitar que sea un jugador el que gestione este alto volumen de comunicaciones, se proponen alternativas híbridas [YV05, RM06, YGM03, SLM04] entre el modelo espacial y las regiones basadas en publicación y suscripción, ya que en estos modelos híbridos se utilizan ordenadores con gran capacidad de cómputo, o supernodos, para realizar estas tareas. No obstante, se deberá asegurar la búsqueda de supernodos en base a requerimientos de cómputo, de balanceo de carga y por último, mecanismos de tolerancia a fallos.

Propagación de los eventos del juego

Mientras la gestión del área de interés de los jugadores se centra en qué información es relevante para cada jugador, la propagación de los eventos del juego se centra en cómo propagar o enviar a los jugadores estas informaciones relevantes. Por consiguiente, la propagación de los eventos del juego no deja de ser una capa envoltorio usada para gestionar las áreas de interés de los jugadores. Las técnicas usadas en este campo son: (a) *Unicast/Multicast* y (b) *Application Level Multicast* (*ALM*).

(a) *Unicast/Multicast*. En la Figura 2.8 se ejemplifican gráficamente los mensajes tipo *unicast* y *multicast*. El método *unicast* [BA08, HL04, HCJ08, MKMA04] se basa en el envío de información desde un único emisor a un único receptor.

Por otro lado, el método *multicast* (o multidifusión) [CK05, FWW02, IHK04, RBD04, DC90], es el envío de la información en una red a múltiples destinos simultáneamente. Antes del envío de la información deben establecerse una serie de parámetros. Para poder recibirla, es necesario establecer lo que se denomina "grupo *multicast*"; grupo que tiene asociada una dirección de Internet.

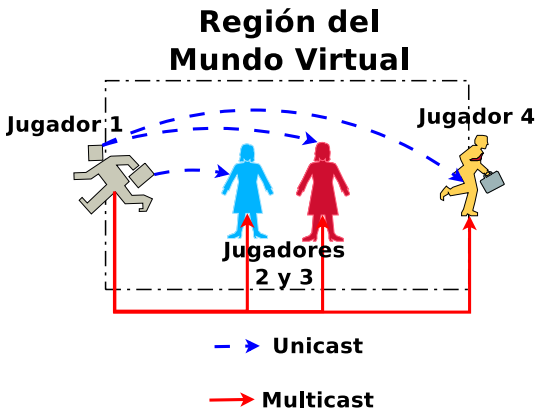


Figura 2.8: Propagación de los eventos del juego: mensajes unicast y multicast.

(b) *Application Level Multicast*. El mecanismo *ALM* [ESRM03, HBH06, KLXH04, GDZL04] se propuso para solventar las deficiencias de la técnica *multicast* [DNBB00]. El *ALM* está centrado como un servicio a nivel de aplicación, en vez de a nivel de capa de red. Sin embargo, este mecanismo introduce un aumento de la latencia en las comunicaciones en comparación con el método *unicast*. No obstante, los métodos *unicast* exigen a los jugadores enviar las notificaciones a un gran número de destinatarios, lo cual puede resultar inviable debido al ancho de banda requerido. *ALM* es utilizado comúnmente sobre redes estructuradas Peer-to-Peer (Bayeux [TTT+01] usa Tapestry [ZKJ01], CAN Multicast [RHKS01] usa CAN [RFH+01], mientras que Borg [ZH03] y Scribe [CDKR02] utilizan Pastry [RD01a]), dado que proporcionan una buena infraestructura de comunicaciones.

Tal y como sucede con la gestión del área de interés de los jugadores, un mecanismo híbrido es la solución. Usando el mecanismo *unicast* para los jugadores cercanos y usando mecanismos basados en *ALM* cuando se trata de jugadores lejanos, donde una falta de sincronización debido a las altas latencias no acarrea un grave problema.

Gestión de los Non Players Characters (NPC) del juego

Además de los jugadores que son controlados por los clientes de las aplicaciones MMOG, existen un conjunto de jugadores llamados *Non Player Characters* (*NPCs*) controlados por algoritmos de Inteligencia Artificial que tradicionalmente han sido ejecutados en los servidores del juego. En cualquier MMOG es requerida la existencia de este tipo de jugadores para darle atractivo y empaque al producto comercial. La gestión de este tipo de jugadores exige un alto consumo de potencia de cómputo y ancho de banda. Por consiguiente, cualquier MMOG ejecutado bajo el paradigma Peer-to-Peer deberá hospedar la gestión de los *NPCs* en la red Peer-to-Peer haciendo uso de los recursos de cómputo existentes en ella, formados por los ordenadores de los clientes del juego. Los mecanismos más extendidos para realizar este cometido son: (a) basado en la región, (b) basado en la distancia virtual y (c) compartición de tareas heterogéneas.

(a) *Método de gestión de los* NPCs *basado en la región* [IHK04, KLXH04]. Este método, ilustrado en la Figura 2.9 se basa en el hecho que se selecciona un nodo o jugador de una región como gestor de la misma. Una de las tareas que debe realizar este nodo o jugador gestor de la región es hospedar y gestionar los *NPCs* de la región. El problema de este método surge cuando el nodo a cargo de estos *NPCs* se desconecta voluntaria o involuntariamente del juego y por tanto, se deberán aplicar políticas de tolerancia a fallos para asegurar la presencia permanente de los *NPCs* en el juego.

(b) *Método de gestión de los* NPCs *basado en la distancia virtual* [ABS06, HCJ08, YKH04]. Este método, ilustrado en la Figura 2.10, se basa en el hecho de que se selecciona el nodo más cercano al *NPC* para que se haga cargo de su gestión. Este método considera que el jugador más cercano a un *NPC* es más probable que vaya a interactuar con él, y por tanto, precisará comunicación. La ventaja es que, al gestionar el propio jugador el *NPC*, se eliminan terceras partes para gestionar la comunicación entre ambos entes del juego, minimizando de este modo las comunicaciones y la latencia. En el ejemplo que se muestra en la Figura 2.10 el *NPCs* realiza movimientos a través de la región y su gestión iría traspasando de los jugadores 1 al 2, del 2 al 3 y de este último al 4, a medida que el *NPC* se fuese aproximando a cada uno de ellos.

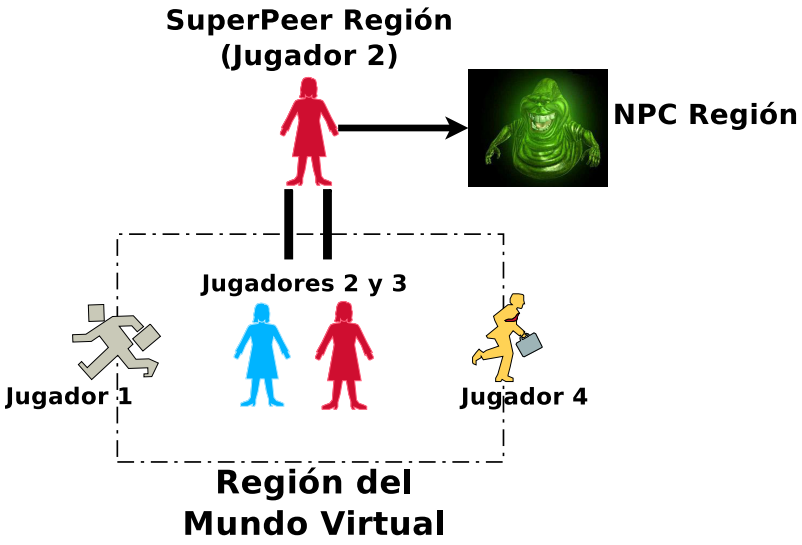


Figura 2.9: Gestión de los *NPCs*: método basado en la región.

Sin embargo, esta técnica tiene ciertas carencias. El hecho que el jugador más cercano al *NPC* interactúe con él, no implica que otros jugadores no lo hagan, dejando a relucir de nuevo problemas de comunicaciones y latencia. Otro problema se produce cuando varios jugadores se mueven alrededor del *NPC*, variando de forma dinámica y frecuente el jugador más cercano al *NPC*. Esto provocará que se incremente de forma notable el *overhead* de comunicaciones y de recursos de cómputo a utilizar para calcular el jugador que hospedará al *NPC*.

(c) *Técnica de compartición de tareas heterogéneas* [FTT10]. Esta técnica, ejemplificada en la Figura 2.11, se basa en la distribución de los *NPCs* de una región a los jugadores que forman parte de ella en función de los recursos de cómputo y de interactividad disponibles que tienen los jugadores, y los recursos de cómputo necesarios para realizar una correcta gestión de los *NPCs*. Para realizar la selección del jugador que hospedará el *NPC*, un nodo de la red conoce los recursos ociosos de los jugadores de la región, así como los recursos necesarios para llevar a cabo la gestión del *NPC* de forma eficiente. De este modo, el jugador con mejor potencial será el que controle el *NPC*. Por tanto, la asignación siempre es buena, ya que se tienen en cuenta las capacidades de cómputo de los jugadores, explotando de este modo la red Peer-to-Peer. Sin

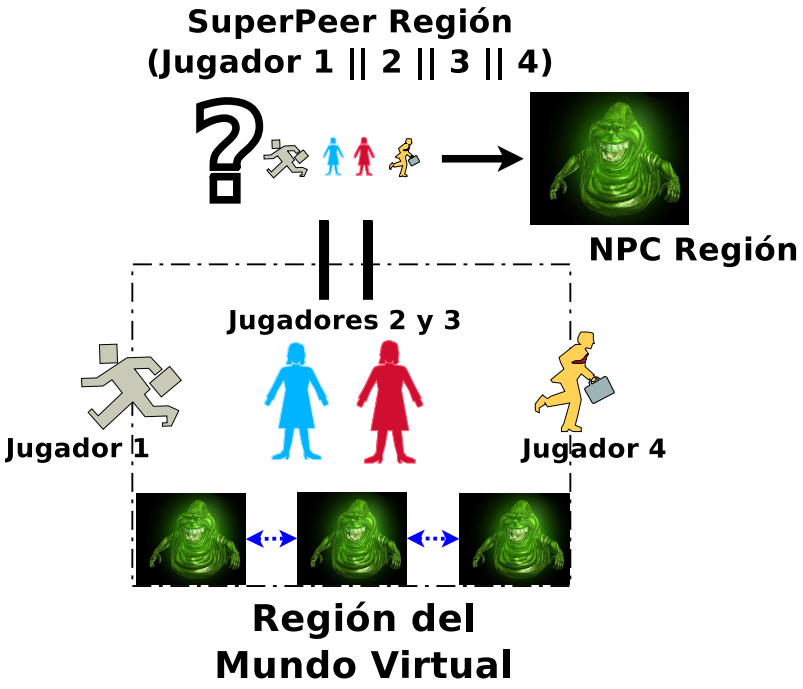


Figura 2.10: Gestión de los *NPCs*: método basado en la distancia virtual.

embargo, en este caso también se debe asegurar que el jugador que hospeda el *NPC* no se desconecte de forma abrupta, para poder realizar la migración del *NPC* de forma adecuada.

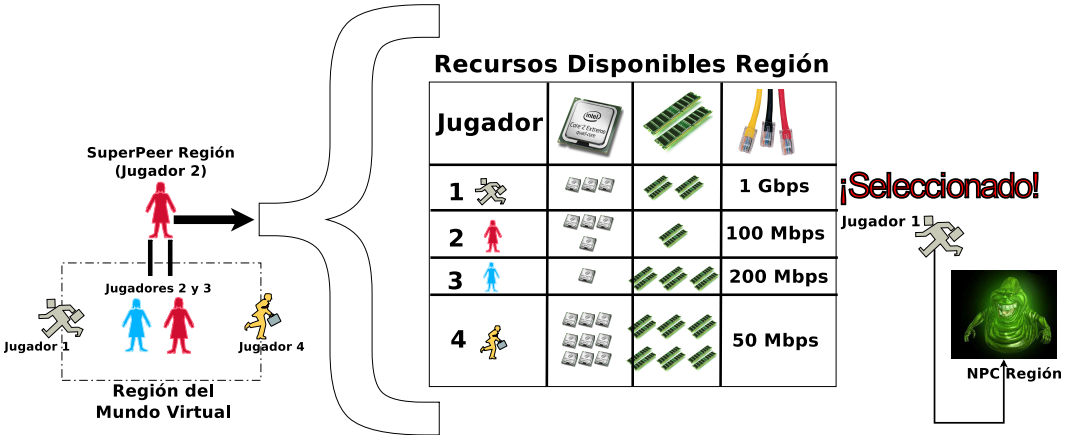


Figura 2.11: Gestión de los *NPCs* del juego: método de compartición de tareas heterogéneas.

Actualmente, el mecanismo basado en la distancia virtual es la técnica

más usada, debido a que minimiza la latencia para los anfitriones (hosts) de los *NPCs*. No obstante, el mecanismo de compartición de tareas heterogéneas tiene su interés, ya que optimiza el total de comunicaciones existentes cuando los *NPCs* interactúan con múltiples jugadores. De este modo, el uso de ambas técnicas parece ser lo más adecuado para escenarios Peer-to-Peer.

Persistencia de los datos del juego

Los juegos tipo MMORPG y MMORTS, pertenecientes a los MMOG, tienen como característica fundamental la persistencia del mundo virtual. Esto significa que el mundo virtual está siempre accesible y en constante evolución a pesar que los jugadores no estén conectados de forma temporal. En estos casos se precisa almacenar todos los perfiles de los jugadores del juego, así como los cambios que se producen entre conexión y conexión, para que los jugadores cuando se conecten de nuevo al mundo virtual, puedan acceder al último estado del juego para continuar jugando. Las técnicas utilizadas en entornos totalmente distribuidos son basadas fundamentalmente en las infraestructuras distribuidas de almacenamiento.

Asegurar una persistencia de datos del juego en una red Peer-to-Peer es un desafío mayúsculo debido a que las infraestructuras distribuidas para el almacenaje de información están enfocadas a redes Peer-to-Peer de compartición de ficheros (*file sharing*) [HBH06, RD01b, RD01a] y rara vez cumplen con los requisitos de funcionamiento y seguridad que precisan las aplicaciones tipo MMOG. En comparación con los hitos anteriores, con un campo de investigación bien explorado, el área de persistencia de datos para aplicaciones tipo MMOG es aún un campo de investigación muy inmaduro con múltiples problemas en espera de encontrar solución.

Técnicas de mitigación de trampas (cheating)

En una arquitectura Cliente-Servidor es sencillo asegurar que los jugadores no realizan trampas, puesto que el servidor es la entidad que valida cada acción del juego antes de actualizar el estado del mismo. Sin embargo, sin la existencia de este ente central que valide los eventos del juego, la prevención de trampas

se convierte en un desafío complicado en entornos distribuidos Peer-to-Peer. En la literatura, algunos trabajos proponen prevenir las trampas mediante algoritmos que reordenan los eventos del juego en conjunción con protocolos de exposición del estado de los jugadores. Otros trabajos en cambio, se basan en la detección y reparación de datos inconsistentes derivados de comportamientos sospechosos. Por tanto, existen dos ramas de investigación centradas en la mitigación de trampas: (a) las técnicas proactivas basadas en detectar los jugadores tramposos antes que cometan la trampa mediante protocolos de exposición de información [BLL07, CFFS05, GZLM04, GS06, CHJ08, CM04, Fun06] y (b) las reactivas [KTCB05, IYM+06, LL08, CRJ+04, LPBC07], que una vez detectada la trampa, deshacen los efectos que ésta ha provocado.

La comunidad científica es consciente de la dificultad que comporta asegurar un sistema libre de trampas en una red Peer-to-Peer, siendo más fácil proveer un sistema seguro bajo arquitecturas convencionales como el Cliente-Servidor. Por otra parte, los problemas de seguridad en entornos totalmente distribuidos se presentan en todas las etapas del diseño e implementación de aplicaciones MMOGs. Aún siendo éste un campo poco explorado, los trabajos basados en los mecanismos proactivos y reactivos de mitigación de trampas empiezan a dar sus frutos.

Mecanismos de incentivos para los jugadores

Los sistemas Peer-to-Peer son por naturaleza sistemas voluntarios de compartición de recursos, donde existe siempre un compromiso entre los intereses individuales y colectivos. Sin embargo, no se pueden obtener beneficios sin una cooperación de todos los entes de la red. Por tanto se debe asegurar que existe un comportamiento cooperativo de los peers que la forman. Para conseguir este tipo de cooperación se requiere incentivar a los cooperantes de un modo u otro [ZA07]. Ejemplos de cooperación de recursos son los siguientes: el ancho de banda es necesario para la propagación de los eventos del juego, la capacidad de almacenamiento es necesaria para guardar el estado del juego de forma permanente, los ciclos de microprocesador son necesarios para computar las áreas de interés de los jugadores, hospedar *NPCs* o solventar posibles trampas. Dentro de los mecanismos de incentivos se destacan los siguientes: (a)

mecanismos de contabilidad [GJA03] y (b) mecanismos basados en reputación [HHJ08, KSGM03, LLW07, SZA08].

El éxito de un juego MMOG ejecutado bajo el paradigma Peer-to-Peer, depende de forma directa de la capacidad que tengan los mecanismos de incentivos de recolectar recursos de cómputo para la correcta gestión de los principales hitos comentados en esta sección. De forma paralela, se precisa de mecanismos de reputación que minimicen comportamientos antisociales de los jugadores. Sin embargo, tal y como veremos a continuación, la mayoría de entornos de juegos realizados en redes Peer-to-Peer y juegos MMOG, carecen de mecanismos de incentivos. Por consiguiente, queda de manifiesto el hecho que, hasta el momento, la comunidad científica tiende a ignorar este campo de investigación y que se requieren mayores esfuerzos dedicados a esta línea.

2.2.2. Comparación de arquitecturas Peer-to-Peer para juegos tipo MMOG

En esta subsección repasaremos y compararemos seis arquitecturas diferentes orientadas a la ejecución de juegos tipo MMOG. La Tabla 2.1 resume las características de cada una de las arquitecturas e ilustra como solventan los hitos esenciales de diseño que hemos comentado previamente. Cabe destacar que la mitigación de trampas, no aparece en esta tabla, debido a que es un tema relativamente separado y todas las arquitecturas pueden adaptar cualquiera de las técnicas de mitigación de trampas mencionadas.

A continuación vamos a repasar de forma resumida cada una de las arquitecturas reflejadas en la Tabla 2.1.

Knutsson et al. [KLXH04]

En el artículo [KLXH04], titulado "*P2P Support for Massively Multiplayer Games*", se realizan particiones del mundo virtual en múltiples regiones, gestionando las áreas de interés de cada una de las regiones mediante el método de publicación y suscripción de eventos. Cada región se asocia a un canal de publicación, al cual todos los participantes de la misma se suscriben. Para realizar la diseminación o entrega de los eventos, se utiliza *Scribe* como herra-

mienta principal. Finalmente, destacar que para el hospedaje y gestión de los *NPCs*, se selecciona un nodo, llamado coordinador de la región, entre todos los nodos existentes en aquella región.

Esta propuesta fue pionera en el ámbito de arquitecturas P2P para MMOGs. Es una propuesta que consta de un buen diseño de la arquitectura, que aún sin constar de muchos componentes, sigue siendo efectiva comparada con otros trabajos más actuales. La gestión de las *AOI* de los jugadores es muy genérica, de modo que no afronta las distintas gestiones de las *AOI* que tienen los distintos tipos de juegos MMOG. Un punto débil, desde el punto de vista de los juegos tipo MMOFPS, es la utilización de un sistema de comunicación tipo *ALM* (*Application-Level Multicast*) que implica tener de forma inherente problemas de latencia (demasiado altas). Referente a la gestión de los *NPCs* comentar que contempla pocos escenarios reales, lo cual implica demasiada sencillez en este hito. Finalmente comentar que la arquitectura no presenta un modelo de persistencia de datos ni de mecanismos de incentivos, lo cual está relacionado con la temporalidad de la propuesta, que recordamos, se realizó varios años atrás, siendo ambos campos aún muy poco explorados.

Dickey et al. [GDZL04]

En el artículo [GDZL04], titulado "*A Fully Distributed Architecture for Massively Multiplayer Online Games*", se propone limitar el tamaño del con-

Propuestas P2P	**Gestión AOI**	**Diseminación Eventos**	**Gestión NPCs**	**Persistencia Datos**	**Mecanismo Incentivos**
Knutsson	Regiones	ALM	Regiones	—	—
Dickey	Regiones	Unicast	Dist. Virtual	Distribuido	—
Douglas	AOI	Unicast	—	Centralizado	—
Hampel	Regiones	ALM	—	PAST	—
Hu	Voronoi	Unicast	Dist. Virtual	Centralizado	REPS
Fan	Híbrido	Unicast	T. Heterogéneas	Distribuido	DCRC

Tabla 2.1: Arquitecturas Peer-to-Peer relevantes para aplicaciones tipo MMOG.

junto de jugadores de cada una de las regiones del mundo virtual a un valor bajo. De este modo, los jugadores pueden comunicarse directamente usando mensajes tipo *unicast*. Sin embargo, en caso que los eventos de una región sean colindantes con jugadores de una región vecina, puede suponer un problema importante. Es por eso que un supernodo es seleccionado en cada región con el fin de propagar los eventos del juego que suceden en una región específica, a los supernodos vecinos, en los casos en los que se requiera. También propone que cada jugador almacene su información permanente, mientras que la información volátil es almacenada en una Tabla de Hash Distribuida (DHT) pública, y por tanto, accesible.

En este caso vemos que existe una mejora en el diseño de la gestión de las áreas de interés de los jugadores, así como en la diseminación de los eventos en el juego. Sin embargo, la distribución de los *NPCs*, selección de los supernodos y los mecanismos de incentivos son simples comparado con trabajos más actuales.

Douglas et al. [DTHK05]

En el artículo [DTHK05], titulado "*Enabling Massively Multiplayer Online Gaming Applications on a P2P Architecture: OPeN Architecture*", se presenta una arquitectura llamada *OPeN*. Esta arquitectura propone un servicio de espacios de índices distribuidos, construido o desplegado en la capa superior de la infraestructura P2P. Con este servicio, los jugadores pueden registrar su posición actual en el mundo virtual y realizar peticiones sobre jugadores cercanos a su área de interés. Entidades cercanas establecen conexiones usando el protocolo *UDP* para intercambiar eventos del juego. La información persistente es almacenada y gestionada por un servidor central.

Los mecanismos de gestión del área de interés de los jugadores del juego, así como la diseminación de los eventos del mismo, es adecuada. Sin embargo, la infraestructura sigue dependiendo de un servidor central para almacenar la información persistente del juego. Además, la distribución de los *NPCs* y los mecanismos basados en incentivos no están presentes en este trabajo.

Hampel et al. [HBH06]

En el artículo [HBH06], titulado "*A P2P Architecture for Massive Multiplayer Online Games*", se presenta una arquitectura basada fundamentalmente en *Scribe* [CDKR02] como capa de diseminación de los eventos del juego, mientras que utiliza *PAST* [RD01b] como herramienta para almacenar la información persistente del juego. *PAST* es un sistema de archivos distribuido, implementado en la capa superior a *Pastry*. En este caso, las regiones en que se divide el mundo virtual constan de un número no limitado de jugadores; por tanto, la gestión de las áreas de interés con más densidad de jugadores será más compleja. En este artículo, sin embargo, no se comenta nada acerca de la gestión de los *NPCs* ni de los mecanismos de incentivos.

La arquitectura usa mensajes tipo *ALM* y un sistema de almacenaje construido en la parte superior de *Pastry*. Sin embargo, estos middlewares están definidos para una arquitectura genérica P2P y no adaptados a las demandas específicas de aplicaciones tipo MMOG. Además, no se argumenta la elección de los middlewares, ni tampoco se demuestra su efectividad.

Hu et al. [HCC06]

En el artículo [HCC06], titulado "*VON: A Scalable Peer-to-Peer Network for Virtual Environments*", se propone un mecanismo de gestión de las áreas de interés basado en diagramas de *Voronoi*. Estos métodos están sustentados en interpolaciones simples, basados en la distancia euclidiana, siendo especialmente apropiados cuando los datos son cualitativos. Al mismo tiempo, estos diagramas resultan adecuados para la distribución de los *NPCs* a lo largo de la red. Además, este artículo describe un sistema de incentivos orientado a la arquitectura propuesta, siendo de los pocos autores que tratan este tema.

Aunque la propuesta basada en diagramas de Voronoi es novedosa y remarcable, debido a que permite gestionar las áreas de interés, propagar los eventos del juego y hospedar los *NPCs* de forma eficiente; la arquitectura sigue requiriendo un nodo centralizado como punto de entrada al juego. Además, este nodo central sigue realizando tareas de balanceo de carga, tolerancia a fallos y almacenaje de la información persistente del juego.

Fan et al. [FTT07]

En el artículo [FTT07], titulado "*Mediator: A Design Framework for P2P MMOGs*", se propone un modelo híbrido para la definición de las *AOI* como el que presentaron A.P Yu y S. T. Vuong en [YV05], basado en la distribución de eventos del juego vía comunicaciones tipo *unicast*. Un nuevo sistema de mapeo de tareas de los *NPCs* (denominado *Deadline-Driven Auctions*) es presentado con el fin de compartir en tiempo real tareas entre *NPCs*. Además, este sistema es de forma inherente compatible con sistemas de prevención de trampas como *Log Auditing*[KTCB05]. Asimismo el entorno es compatible con mecanismos basados en incentivos como *DCRC*[GJA03]. Finalmente se ha desarrollado un sistema de afiliación a multicast (*MAMBO*) [FTT08] para gestionar las zonas del juego de forma eficiente. Se utiliza *PAST* como sistema de gestión de datos persistentes del juego, siendo totalmente compatible con *MAMBO*, ya que comparten la misma capa de red.

Otras arquitecturas

Paralelamente a las arquitecturas distribuidas comentadas anteriormente, existen trabajos remarcables en la literatura que afrontan ciertos campos de la gestión de los MMOGs que son esenciales. Por consiguiente, a continuación se repasan aquellos trabajos que han sido relevantes dentro del campo de la actualización o diseminación del estado del juego.

Vamos a realizar una diferenciación entre las arquitecturas basadas en: (a) Tablas de Hash Distribuidas (DHT) y (b) árboles jerárquicos.

(a) *Tablas de Hash Distribuidas* El funcionamiento general de varias arquitecturas P2P está basado en un sistema de publicación y suscripción construido sobre una Tabla de Hash Distribuida (DHT). Los jugadores se suscriben para recibir actualizaciones de distintas regiones del juego que se encuentran mapeadas en la DHT. A medida que los jugadores modifican el estado del juego, publican sus cambios en la DHT, que son enviados a los jugadores que están suscritos en aquella región.

Uno de los artículos más relevantes en la diseminación de eventos del juego,

basado en la suscripción y publicación, es el trabajo realizado por Bharambe et al. en [BAS04]. En este trabajo presentan un sistema llamado *Mercury* que permite peticiones en rango eficientes, lo que habilita a los jugadores a especificar el número de actualizaciones que quieren recibir, es decir, filtrar estas actualizaciones. El sistema *Mercury* es utilizado por una arquitectura llamada *Colyseus* por los autores Bharambe et al en [ABS06], en la que se demuestra como es capaz de tolerar o gestionar muchos más jugadores que una arquitectura Cliente-Servidor clásica, mientras se mantienen unos valores de latencia aptos para este tipo de juegos. En esta arquitectura, todos los objetos mutables, incluyendo los jugadores, son gestionados por una arquitectura global de almacenamiento, que mapea cada objeto en un único nodo de la tabla DHT. Todas las actualizaciones son manejadas a través del propietario del objeto, haciendo uso del sistema de publicación y suscripción de *Mercury*. Para asegurar un buen rendimiento, *Colyseus* realiza réplicas de los objetos y coordina su consistencia. Eso sí, cabe destacar que *Colyseus* logra este rendimiento modificando el código interno del juego *Quake II*, un juego del tipo MMOFPS.

Otro artículo relevante, basado en tablas de hash distribuidas, es el presentado por Chan et al en [CYB+07]. En este artículo se utilizan las tablas de hash distribuidas de *Pastry* para propagar o distribuir las actualizaciones del estado del juego a través de toda la red de peers. El mundo virtual es dividido en regiones y cada una de ellas, así como los estados del juego de los jugadores que se encuentran en ella, se mapean en una entrada de la DHT. Esta es coordinada por el jugador que tiene en su nodo la ejecución o posesión de la misma. Cada jugador de forma regular envía su localización actual al coordinador de la región, que actualiza su posición al resto de jugadores de la región mediante mensajes tipo *multicast*. Las interacciones entre jugadores y objetos son esencialmente actualizaciones del estado del juego y son gestionadas por el coordinador de la región.

Por último, destacamos la aportación de Iimura et al. en [IHK04] basado en un modelo de zonas federadas. En este trabajo también se divide el mundo virtual en regiones y se almacena el estado del juego de cada región en una DHT. En este caso, sin embargo, la DHT es utilizada sólo como sistema de "*backup*", es decir, de copia de seguridad en caso de que el coordinador de la

zona falle o quisiera renunciar de su rol. La comunicación, por otro lado, se realiza directamente entre el jugador y el gestor o "propietario" de la zona donde reside.

(b) *Árboles jerárquicos* Otro modo de distribuir las actualizaciones del estado del juego entre los jugadores es a través de una jerarquía. Tal y como refleja la literatura repasada a lo largo de este capítulo, una propuesta común es dividir el mundo virtual en regiones y designar a un jugador para gestionar el estado de cada una de las regiones. Los jugadores pueden intercambiar actualizaciones con el responsable de cada una de las regiones en cuestión.

Uno de los problemas principales de este tipo de arquitecturas es distribuir la carga de una región, cuando ésta se convierte en una zona sobrecargada debido a un gran número de jugadores residiendo en ella. Con árboles con carga balanceada, como la propuesta de Yamamoto et al. en [YMYI05], el nodo responsable construye un árbol de propagación con los jugadores existentes en la región, utilizándolo para hacer entrega de la actualización de los eventos de la región. Con la propuesta basada en subservidores, propuesta por Lee et al. en [LS06], el nodo responsable subdivide su región en áreas pequeñas y asigna a cada una de las áreas un servidor, siendo de este modo un árbol jerarquizado.

Una alternativa a estas jerarquías *top-down*, es decir, que se construyen de arriba hacia abajo, es construirlas en sentido inverso de abajo hacia arriba o *bottom-up* y basadas en el alcance de los eventos, como la propuesta de GauthierDickey et al. en [GLZ05]. En estas propuestas, los eventos tienen distinto alcance, por ejemplo: encender unas luces en una habitación, cortar la luz a un edificio entero o sufrir una tormenta de nieve en toda la ciudad. Los jugadores se unen al árbol, teniendo en las hojas del mismo los eventos con menor alcance. Los jugadores pertenecientes a eventos de bajo alcance, intercambian directamente entre ellos las actualizaciones del juego, utilizando un protocolo criptográfico que envía los eventos locales y evita las trampas (GauthierDickey et al. en [GZLM04]). Cuando los eventos son de mayor alcance, éstos son enviados a todos los nodos afectados utilizando el árbol de alcance.

Como se ha descrito en la presente sección, son muchas las soluciones aportadas en el ámbito Peer-to-Peer; sin embargo, aún existen ciertas carencias

que deben ser solventadas. Con el fin de solucionar estas carencias, recientemente, han surgido nuevas líneas de investigación que mezclan el paradigma Cliente-Servidor y Peer-to-Peer para dar solución a los problemas existentes de la ejecución de aplicaciones tipo MMOG en ambos entornos en su estado puro.

2.3. Soluciones basadas en un modelo híbrido Cliente-Servidor y Peer-to-Peer

Tal y como hemos comentado en los apartados anteriores, ni las arquitecturas Cliente-Servidor ni las arquitecturas Peer-to-Peer solventan enteramente los problemas que generan aplicaciones tan dinámicas y complejas como los juegos MMOG. Es por esta razón que recientemente se están explorando soluciones mixtas que incorporan elementos centralizados y distribuidos, con el fin de lograr un rendimiento eficiente y económico para dichas aplicaciones. Dentro de esta área de investigación se enmarca el presente trabajo de tesis doctoral.

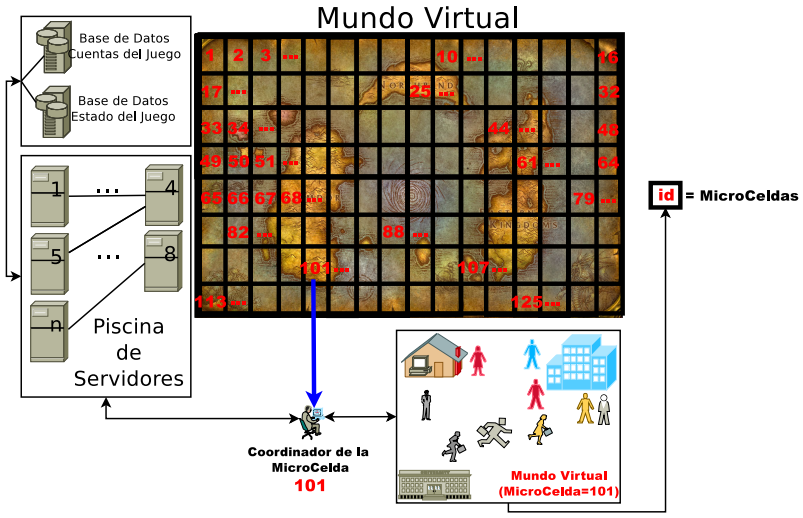


Figura 2.12: División MMOG en micro-celdas/regiones/zonas con arquitectura Híbrida.

Uno de los problemas conocidos asociados a las arquitecturas Peer-to-Peer

específicas para aplicaciones tipo MMOGs es que comúnmente carecen de mecanismos de detección y supresión de engaños o trampas. Cualquier arquitectura que permita a los jugadores almacenar y gestionar el estado del juego es susceptible a la desestabilización del juego por parte de los jugadores. Por tanto, cualquier arquitectura que distribuya las actualizaciones del juego a través de los propios jugadores - ya sea a través de tablas de hash distribuidas, haciendo uso de una capa de difusión basada en mensajes tipo *multicast* o mediante arquitecturas jerarquizadas – permite cualquier mal comportamiento por parte de algún jugador malicioso. Además será difícil para los jugadores determinar quién tiene la culpa de este mal comportamiento.

Estas situaciones se pueden solventar si la organización encargada de la gestión del juego despliega la ejecución del mismo en máquinas que le pertenecen o en un servicio de algún proveedor. En este caso, los jugadores se podrían conectar a la infraestructura distribuida desde varios puntos de acceso; de este modo los jugadores no se harían cargo de la distribución del estado del juego o de eventos del mismo. Sin embargo, en este caso, la organización no se beneficia del recorte de gastos que supone usar los recursos computacionales de los ordenadores de los propios jugadores del juego.

Por tanto, por un lado parece ser indicada la inclusión de entes controladores centrales siempre accesibles, pero por otro lado es también sumamente interesante utilizar la capacidad de cómputo ociosa de los ordenadores de los jugadores para ahorrar estos trabajos a los servidores centrales de los juegos. En este sentido, Jardine et al propusieron en [JZ08] una arquitectura híbrida basada en los paradigmas Cliente-Servidor y Peer-to-Peer, con el fin de ejecutar de forma eficiente aplicaciones tipo MMORPG.

Los autores del artículo comentan que el punto crítico de las arquitecturas Cliente-Servidor aplicadas a este tipo de juegos se halla en el consumo de ancho de banda, que es cuadrático con el número de jugadores que debe gestionar cada servidor del juego. Sin embargo, reconocen que el Cliente-Servidor es esencial para asegurar la consistencia de datos, mecanismos anti-trampas y asegurar a los jugadores que todas sus horas de dedicación al juego no serán en vano, es decir se trata de asentar la persistencia de las cuentas y progresos de los jugadores.

Para solventar los problemas de ancho de banda proponen un modelo híbrido donde se seleccionarán ciertos nodos para actuar como controladores de regiones en función de la calidad de los mismos, que es evaluada en base al ancho de banda y a la latencia que estos nodos tienen. Estos controladores recibirán las actualizaciones de estado de los jugadores que tienen a su cargo. A este tipo de actualizaciones se las denomina "*positional moves*" o cambios de posición, es decir cambios de posición del jugador sin que ninguna entidad del juego se vea involucrada (*NPCs*, otro objeto o jugador). Sin embargo se seguirán enviando al servidor central las actualizaciones de estado llamadas "*state-changing moves*", puesto que en estos casos sí que hay una implicación con otro elemento del juego como puede ser un objeto del mismo, un jugador o un *NPC*.

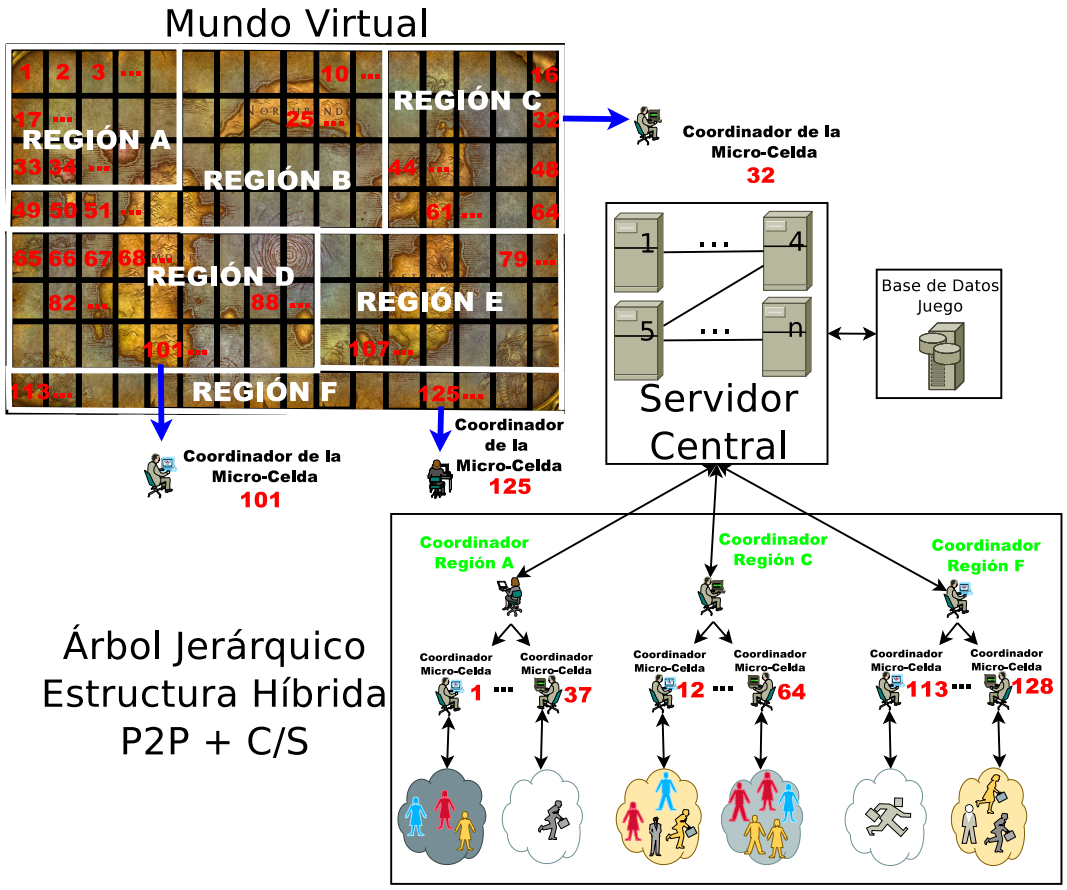


Figura 2.13: División MMOG en micro-celdas con arquitectura híbrida y estructura en árbol jerárquico.

El modelo híbrido propuesto por Jardine et al., mostrado en la Figura 2.13, provee de seguridad usando un servidor central que controla el acceso al estado del juego. Esto proporciona al servidor un gran potencial, ya que el objetivo último de la mayoría de los juegos es obtener la gestión o control del estado del juego. Por tanto, cualquier interacción entre los jugadores y el estado del juego es arbitrada por el servidor, que gestiona los eventos y los cambios en el estado del juego. Además, el servidor central registra a los jugadores y monitoriza sus acciones, pudiendo expulsar del juego a aquellos jugadores que no tengan un comportamiento apropiado, o directamente a los jugadores que atenten contra la seguridad e integridad del juego.

Asimismo, la arquitectura híbrida que repasamos es totalmente escalable distribuyendo el estado del juego a través de los jugadores. El servidor divide el mundo virtual en regiones y selecciona un jugador para distribuir las actualizaciones de esta porción o región del juego previamente particionada. Asimismo, en cualquier momento que un jugador realiza un simple cambio de posición se envía este cambio al servidor regional, que a su vez, distribuye el movimiento al resto de jugadores de la región. Cuando un jugador realiza un cambio en su estado de juego, es decir, un cambio que afecta al entorno del jugador (objetos, escenario, otros jugadores, etc.), es enviado directamente al servidor central. Si el cambio de estado es aceptado, el servidor central envía el resultado al jugador y actualiza el mismo al servidor regional pertinente, que a su vez distribuye el nuevo cambio de estado al resto de jugadores de la región.

Los autores comentan que esta propuesta permite realizar un ahorro importante de ancho de banda, ya que los cambios posicionales son gestionados por el servidor regional, evitando que el servidor central entre en acción. El ahorro es proporcional al porcentaje de movimientos posicionales que se realizan, que dicho sea, son la mayoría. El ahorro de ancho de banda también proviene de las actualizaciones que realiza el servidor central a los servidores regionales cada vez que se acepta un cambio de estado de juego, en vez de hacerlo a todos los jugadores de la región afectada. En este caso, el ahorro es proporcional a la densidad media de la región.

2.4. Valoración de las propuestas

De los trabajos expuestos se deduce que es más sencillo llevar a cabo un control centralizado de todo el juego MMOG, con el fin de asegurar los distintos hitos repasados como las trampas, la persistencia de datos, etc., ya que los servidores del juego tienen control absoluto en todo momento de los jugadores, el estado del juego, la información del mismo, los datos, etc. Por contra, en un escenario totalmente distribuido, donde el control lo gestionan todos los jugadores, resulta ser mucho más complejo afrontar estos hitos para la correcta ejecución de una aplicación tipo MMOG.

Las soluciones aportadas bajo entornos Cliente-Servidor para juegos MMOG, si bien es cierto que mejoran el rendimiento de la ejecución, mediante mecanismos de división y balanceo de carga del juego entre los servidores, carecen de aplicabilidad cuando se trata de diseñar sistemas escalables de bajo coste. Resumiendo, el paradigma Cliente-Servidor puede ser mejorado con el fin de mejorar su eficiencia, pero sigue careciendo de una escalabilidad ilimitada que teóricamente, sí pueden aportar los sistemas distribuidos. Sin embargo, adaptar la ejecución de aplicaciones MMOG de las arquitecturas convencionales, como es el Cliente-Servidor, a nuevos paradigmas como el Peer-to-Peer, es un desafío activo para toda la comunidad científica.

No obstante, aún cuando una arquitectura totalmente distribuida o Peer-to-Peer parece ser la solución natural al problema de escalabilidad de este tipo de aplicaciones tan dinámicas, con millones de jugadores que acceden desde todas partes del mundo, este tipo de soluciones resultan ser poco pragmáticas para su explotación a nivel comercial debido fundamentalmente a las siguientes razones:

- Los creadores de un juego, normalmente una empresa comercial, quieren hacer negocio con la explotación de su producto. En cambio no pueden controlar la ejecución, sucesos y estados del juego en un sistema totalmente distribuido lo cual implica la pérdida absoluta de la gestión del mismo, y por consiguiente, una posible pérdida del negocio.

- El estado del juego es almacenado en los ordenadores de los clientes (jugadores) del juego, siendo cada uno de ellos responsable de una pequeña

porción del juego. Por consiguiente, si un jugador no se encuentra en línea, esta porción del juego no es accesible por el resto de jugadores del juego.

- Los clientes/jugadores del juego deben gestionar los cambios de estado del juego. Para realizar dicha tarea realizan un envío masivo a todos los jugadores, lo cual representa un incremento excesivo del consumo del ancho de banda.

- Gestionar la latencia de zonas de un juego gestionado por uno o más clientes del mismo resulta complejo, debido a la aleatoriedad y variabilidad de la latencia que experimentan los jugadores en función de la localización geográfica de su proveedor de Internet. Por tanto, pueden producirse problemas con la calidad de servicio derivados de altas latencias.

- El hecho que sean los propios clientes del juego los encargados de realizar una gestión absoluta de la dinámica del juego, también facilita que jugadores maliciosos puedan realizar trampas, a la vez que dificulta su detección, al ser una red totalmente distribuida.

A la vista de las razones argumentadas a lo largo del capítulo, vemos que los sistemas P2P resultan ser la solución natural para conseguir la escalabilidad necesaria que comporta el aumento dinámico de jugadores, y poco controlado en ocasiones, que puede acontecer en un juego MMOG. Sin embargo, vemos por otro lado que la ejecución totalmente distribuida, que es inherente a los sistemas P2P, plantea retos muy importantes a resolver desde el punto de vista del control global de cada juego. Por esta razón, en la presente tesis doctoral, se propone una arquitectura híbrida cliente-servidor y P2P, que por una parte es capaz de llevar a cabo un control centralizado de los juegos en el sistema cliente-servidor y por otra es capaz de escalar según la demanda de los jugadores, traspasando parte de la ejecución al área distribuida P2P. De esta forma se llega a una solución de compromiso entre eficiencia y escalabilidad, que supone una propuesta con características diferenciales respecto a las existentes en la literatura previamente analizadas en este capítulo.

Capítulo 3

Análisis del comportamiento de ejecución de los juegos multijugador masivos en línea

De los tres tipos de juegos MMOG que hemos repasado a lo largo de los capítulos anteriores, esta tesis doctoral, se centra en los MMOFPS y los MMORPG, puesto que tienen características diferenciadas y al mismo tiempo cuentan con gran interés dentro de las comunidades de jugadores de MMOGs.

En este capítulo se analizarán los requerimientos de recursos de cómputo de los juegos MMOFPS y MMORPG en un entorno de ejecución Cliente-Servidor. En concreto se evaluará el consumo de CPU, memoria y ancho de banda en un entorno de monitorización controlado. Asimismo se estudiará el grado de satisfacción de los jugadores de los juegos analizados. Cabe destacar que este análisis se centrará en el estudio empírico de las necesidades de recursos de cómputo que tienen este tipo de aplicaciones.

3.1. Métricas de rendimiento evaluadas

En el capítulo 1 se ha explicado que los requerimientos de recursos de cómputo de las aplicaciones MMOG han ido incrementando con el paso del tiempo. Los MMOGs cada vez ofrecen un mayor número de opciones a los jugadores, al mismo tiempo que los entornos o escenarios de los mundos virtuales

son cada vez más complejos y detallistas. Estas mejoras, que por un lado son muy atractivas para los jugadores, tienen como contrapartida un incremento de los recursos de cómputo necesarios para gestionar este gran abanico de posibilidades y recursos gráficos.

Por consiguiente, si se quiere aportar un nuevo paradigma de ejecución de aplicaciones tipo MMOG, este proceso de análisis es necesario con el fin de proponer un entorno de ejecución eficiente que concuerde con los requerimientos de las aplicaciones, es decir, que se adapte al comportamiento que tienen estas aplicaciones y al uso que hacen de los recursos de cómputo. De forma paralela, deberemos identificar los parámetros de estos juegos que causan un mayor o menor consumo de los recursos de cómputo de un servidor. Por ejemplo, tendremos que estudiar si el parámetro *número de jugadores* es un factor clave, o bien otros parámetros como *número de partidas*, *localización geográfica de los jugadores* u otros parámetros son los que determinan de forma más directa un incremento o disminución del uso de los recursos de cómputo.

Acorde con lo razonado anteriormente, el análisis de los juegos tipo MMOFPS y MMORPG se efectuará en función del consumo de los siguientes recursos:

- *Procesador (CPU).* Este parámetro es vital, puesto que determinará el tiempo de procesamiento de los eventos del servidor, permitiendo de este modo una mayor o menor fluidez en la ejecución continua del juego, así como en el número máximo de partidas y jugadores que pueden ser gestionados concurrentemente.

- *Memoria RAM.* Estos juegos necesitan cargar estructuras de datos en la memoria. De este modo, el tamaño de la memoria RAM del servidor, junto con los requerimientos de memoria de cada partida, determinarán el número de instancias de juego, o partidas, que se pueden llegar a albergar en el servidor de forma concurrente.

- *Red (Networking).* Al ser juegos multijugador masivos en línea, requieren de una conexión a Internet para el envío y recibo de paquetes de red que contienen las actualizaciones del estado del juego. A mayor capacidad de envío y recepción de paquetes, mayor será la capacidad del servidor para gestionar un mayor número de juegos y jugadores de forma concurrente.

Asimismo, en este análisis se evaluará también el *Grado de Satisfacción de los jugadores (GdS)*. Esta métrica se estudiará en función del *ping*[1] que éstos tienen respecto al servidor del juego, con el fin de clarificar cuales son los umbrales de latencia aceptables. Por tanto, en función de la latencia que los jugadores tienen respecto al servidor de la partida, éstos experimentarán una mayor o menor calidad de servicio, hecho que hará variar su *GdS*, y en definitiva, el que un juego (o servidor de un juego) tenga mayor o menor interés.

Por tanto, vemos que de los cuatro parámetros analizados, tres hacen referencia a aspectos de cómputo y uno a la calidad de servicio *GdS*. Sin embargo, todos ellos están condicionados por aspectos como: (a) la situación geográfica de los jugadores, puesto que ésta determinará la latencia que tienen respecto al servidor central del juego y (b) el volumen total de jugadores, ya que tanto el tiempo de procesamiento del servidor, como la capacidad de albergar nuevas partidas, dependerá del uso de CPU, memoria RAM y ancho de banda de los juegos albergados en el servidor. De la percepción que los jugadores tengan del rendimiento del servidor en base a los tres parámetros descritos, especialmente al que a latencia se refiere, éstos percibirán una mayor o menor calidad de servicio *GdS*.

3.2. Metodología de estudio

Una vez descritos los parámetros en los que se basará el análisis, en esta sección se describirá la metodología seguida para la monitorización de los juegos.

Con el fin de estudiar el comportamiento de servidores y clientes de los juegos MMOFPS y MMORPG se ha optado por configurar un entorno de pruebas. Para la monitorización del servidor de juegos tipo MMOFPS se han utilizado los juegos *Counter Strike 1.6* y *Urban Terror*, los cuales fueron ejecutados bajo dos entornos distintos:

- *Entorno local o LAN*. En el que habrán jugadores ubicados en dos labo-

[1]Muchas veces se utiliza para medir la latencia o tiempo que tardan en comunicarse dos puntos remotos, y por ello, se utiliza el término *ping* para referirse al lag o latencia de la conexión en los juegos en red.

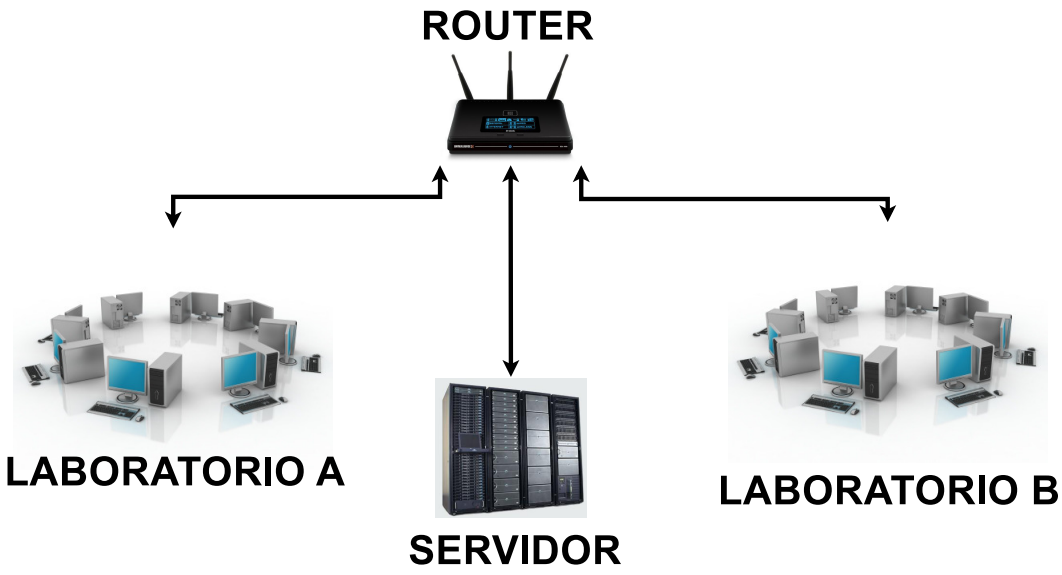


Figura 3.1: Esquema de monitorización en red local (*LAN*).

ratorios de la Escuela Politécnica Superior de la Universitat de Lleida, separados físicamente, se conectan a un servidor a través de red local. En la Figura 3.1 se ilustra esta situación, en la que los jugadores del *laboratorio A* tienen un *ping* más elevado que los jugadores del *laboratorio B*, hecho que se ha logrado mediante el uso de reglas de red (*iptables*[2]).

- *Entorno global o Internet*: En el que habrán jugadores ubicados en distintos países se conectan a un servidor a través de red global o *Internet*. En la Figura 3.2 se ilustra esta situación. Cabe destacar que por el idioma y el tipo de juego, la mayoría de jugadores se conectaban desde España y Sudamérica, siendo los primeros, jugadores con mejores latencias por su proximidad geográfica respecto del servidor de la partida.

Para el análisis de los juegos tipo MMORPG, ante la imposibilidad de obtener trazas reales de servidores comerciales en línea de los juegos más representativos, y siendo los resultados del análisis de los juegos tipo MMOFPS suficientemente representativos y similares a los juegos de rol, tal y como indican

[2]Las iptables constituyen una herramienta de cortafuegos que permite no solamente filtrar paquetes, sino también realizar traducción de direcciones de red (NAT) para IPv4 o mantener registros de log.

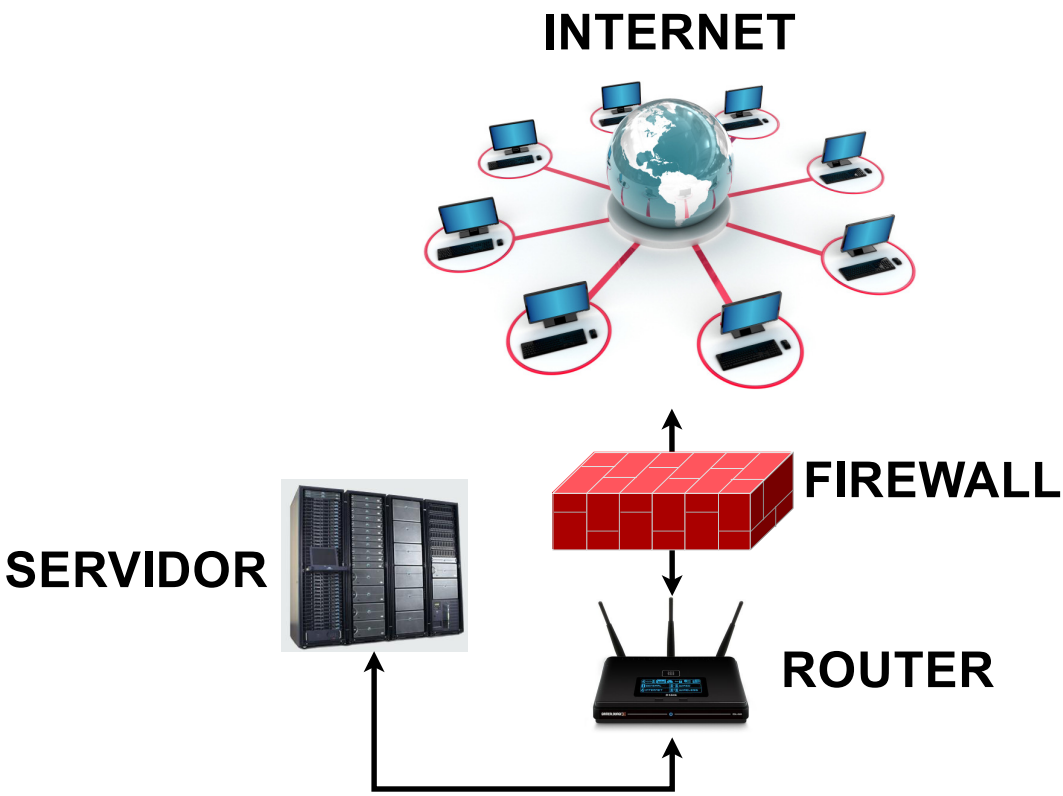


Figura 3.2: Esquema de monitorización de red global (*Internet*).

trabajos que formalizan matemáticamente estos tipos de juegos [YC06, LC], se ha optado por realizar un análisis centrado en el estudio de los mecanismos del juego. Por tanto, la metodología en este caso ha sido: (a) instalación y pago de suscripción del cliente *World of Warcraft* (versión *Wrath of the Lich King*) con el objetivo de conocer la dinámica y funcionamiento de este tipo de juegos y (b) obtención de datos estadísticos de consumo de CPU, concurrencia de jugadores, picos de conexión, etc., de un servidor del juego *PlaneShift*[Cora].

Para la monitorización de los juegos MMOFPS en los entornos mencionados anteriormente, los servidores usados han sido los siguientes:

1. *Equipo doméstico de sobremesa* para el entorno local, que está equipado con los siguientes componentes:

   - Procesador Intel Core 2 Duo a 2.2GHz y 800MHz.
   - 2GB DDR2 SDRAM.

Tipo de prueba	HP ProLiant	Equipo doméstico
Entorno Local		√
Entorno Global	√	
Grado de Satisfacción	√	√
Cliente *Urban Terror*		√
Cliente *World of Warcraft*		√

Tabla 3.1: Resumen de la relación de pruebas efectuadas y el equipamiento de servidores usados.

- 250GB SATA.
- Ethernet 10BASE-T/100BASE-TX con interfaz RJ-45.

2. *HP ProLiant DL 380 G5* para el entorno global, que está equipado con los siguientes componentes:
   - 2 Intel Quad Core XEON E5420 a 2.5GHz y 1333MHz.
   - 16GB RAM HP PC2-5300.
   - 146GB SAS HPLU SFF.
   - 4 x 250GB SATA 5.4K SFF ETY 3G.
   - 3 Interfaces de red: ILO, ETH0 y ETH1 (Gigabit Ethernet 1.000BASE-T/1.000BASE-TX con interfaz RJ-45).

En la Tabla 3.1 resumimos el equipamiento usado en los distintos tipos de pruebas efectuadas a lo largo del análisis del comportamiento de ejecución de los juegos. Marcamos con el símbolo "√" las pruebas realizadas con uno u otro servidor. En esta tabla se muestra como el servidor *ProLiant* ha sido usado en las pruebas con los juegos tipo MMOFPS para el entorno global, así como para obtener el *GdS* de los jugadores conectados a través de Internet. Del mismo modo, se ha utilizado el equipo doméstico para las pruebas de ámbito local, así como para la obtención del grado de satisfacción de los jugadores conectados a través de LAN. De forma paralela se ha estudiado el comportamiento de la parte cliente del juego MMOFPS *Urban Terror* y del juego MMORPG *World of Warcraft*, con el fin de experimentar en primera persona las sensaciones y

Resumen datos obtenidos	
Fecha Inicio	3-03-2009
Fecha Fin	15-06-2009
Número de partidas	306
Duración media de las partidas (segundos)	2.850
Número medio de jugadores por partida	24,64
% partidas bajo entorno local	65,4 %
Número medio de jugadores por partidas (entorno local)	30,29
% partidas bajo entorno global	34,6 %
Número medio de jugadores por partida (entorno global)	17,3

Tabla 3.2: Resumen de los datos obtenidos con la monitorización con los servidores *ProLiant* y *doméstico* del juego *Urban Terror*.

distintos grados de satisfacción que un jugador tiene cuando se varía la latencia o ping respecto del servidor. Estos análisis se han llevado a cabo durante un periodo de tres meses. Esto ha permitido obtener información relevante acerca del número de partidas de juego iniciadas, la duración media de cada una de ellas y el número medio de jugadores por instancia (o partida). Un resumen de los datos obtenidos para el juego *Urban Terror* son mostrados en la Tabla 3.2.

Los resultados que mostraremos en las Secciones 3.3, 3.4 y 3.5 están basados en las pruebas realizadas con el servidor doméstico en el entorno local. El hecho de centrarnos en los resultados obtenidos con el servidor doméstico y el entorno local tiene como objetivo evaluar el impacto de una única ejecución de un juego tipo MMOFPS sobre las prestaciones de un ordenador doméstico, como caso representativo de un nodo de la red P2P. Por tanto, queremos estudiar si los recursos de cómputo ociosos, que hoy en día existen en un ordenador doméstico, son suficientes para proveer, con calidad de servicio, una partida (o instancia) de un juego tipo MMOFPS. Asimismo cabe remarcar que la diferencia entre el entorno global y local, por lo que a consumo de CPU, memoria RAM y ancho de banda se refiere, son prácticamente nulas, puesto que no importa el entorno de las pruebas, sino el número de jugadores y partidas concurrentes, tal y como mostraremos en las futuras secciones del presente capítulo.

Los parámetros de cómputo, así como el grado de satisfacción de los juga-

dores, descritos en la sección anterior, han sido monitorizados utilizando las siguientes herramientas:

- *CPU.* Fundamentalmente se ha utilizado la herramienta *SAR*[3], la cual permite conocer en tiempo real el consumo de procesador, memoria, ancho de banda y un sinfín de parámetros de sistema más. Es una herramienta implementada bajo el entorno *UNIX*, que permite medir el consumo del procesador (incluyendo sus núcleos) en cada instante de ejecución del juego.

- *Memoria.* Del mismo modo que la medida del consumo de CPU, la memoria ha sido monitorizada con la herramienta *SAR*.

- *Networking.* Para medir el número de paquetes entrantes y salientes, el consumo de ancho de banda, latencias, etc., se ha utilizado el comando *tcpdump*[4] del sistema *UNIX*, lo que nos ha permitido analizar los paquetes que circulan por la red. También se ha usado la herramienta gráfica *WireShark*[5], que es un analizador de protocolos (anteriormente llamado *Ethereal*) utilizado para realizar análisis y solucionar problemas en redes de comunicaciones, así como para desarrollo de software y protocolos. Por último, la medida de la latencia que el servidor tiene respecto a los jugadores se ha efectuado mediante el mecanismo *ping*, ejecutado entre las direcciones *IP* del jugador y del servidor.

- *Grado de Satisfacción (GdS).* Para valorar el grado de satisfacción de los jugadores se les ha preguntado que valoren su experiencia de juego, siendo 1 el grado de satisfacción más bajo y 5 el más alto. Esta valoración estará totalmente condicionada por la latencia que cada uno de ellos tiene respecto del servidor, es decir, en función de la rapidez con la que el servidor procesa sus acciones en la partida, así como las acciones del resto de los jugadores, puesto que el resto de características (CPU y memoria) son las mismas para todos. Cabe destacar que la valoración del *GdS* será mayor o menor, en función de los resultados que va obteniendo

[3] `http://linux.die.net/man/1/sar`.
[4] `http://www.tcpdump.org/`.
[5] `http://www.wireshark.org`.

el jugador, es decir, si gana o pierde. El método usado para la obtención del *GdS* ha sido la encuesta, que se ha colgado en el portal web del servidor, con el fin que los jugadores rellenaran las puntuaciones que hemos definido[6].

Para el tratamiento de los datos se ha utilizado *Shell Scripting*, diseñando e implementando *scripts* para el intérprete de línea de comandos del sistema operativo. Para la representación gráfica de los datos se ha utilizado *GnuPlot*, ya que se trata de un programa muy flexible para generar gráficas de funciones y datos[7]. Por último, también se ha usado un sistema de monitorización de clústers llamado *Ganglia*[8], que nos ha permitido ver en tiempo real información de consumo de CPU, memoria, ancho de banda, espacio en disco y un sinfín de parámetros de sistema más.

En las siguientes secciones se mostrarán los datos obtenidos de la ejecución en el entorno local del juego MMOFPS *Urban Terror*. Este juego permite partidas con un máximo de 32 jugadores y por tanto, los resultados mostrados están caracterizados por esta limitación, que por otro lado es comúnmente extendida al resto de juegos de su género. Finalmente remarcar que los resultados obtenidos con el otro juego MMOFPS monitorizado *Counter Strike 1.6* han sido prácticamente similares y por esta razón no son mostrados en este análisis.

3.3. Consumo de CPU

La Figura 3.3 muestra el porcentaje de uso de CPU en el servidor del juego MMOFPS al ir incrementando el número de jugadores (de 5 en 5) a lo largo del tiempo. Tal y como se aprecia en la figura, el consumo de CPU es proporcional al número de jugadores existentes en el servidor, puesto que va aumentando a medida que se conectan nuevos jugadores. Este hecho se refleja en la Tabla 3.3, en la que se muestra el número de jugadores concurrentes en el servidor y el rango medio de consumo de CPU. A modo de ejemplo podemos ver como el

[6] `http://games0.gcd.udl.cat/wordpress/`.
[7] `http://www.gnuplot.info`.
[8] `http://ganglia.sourceforge.net/`.

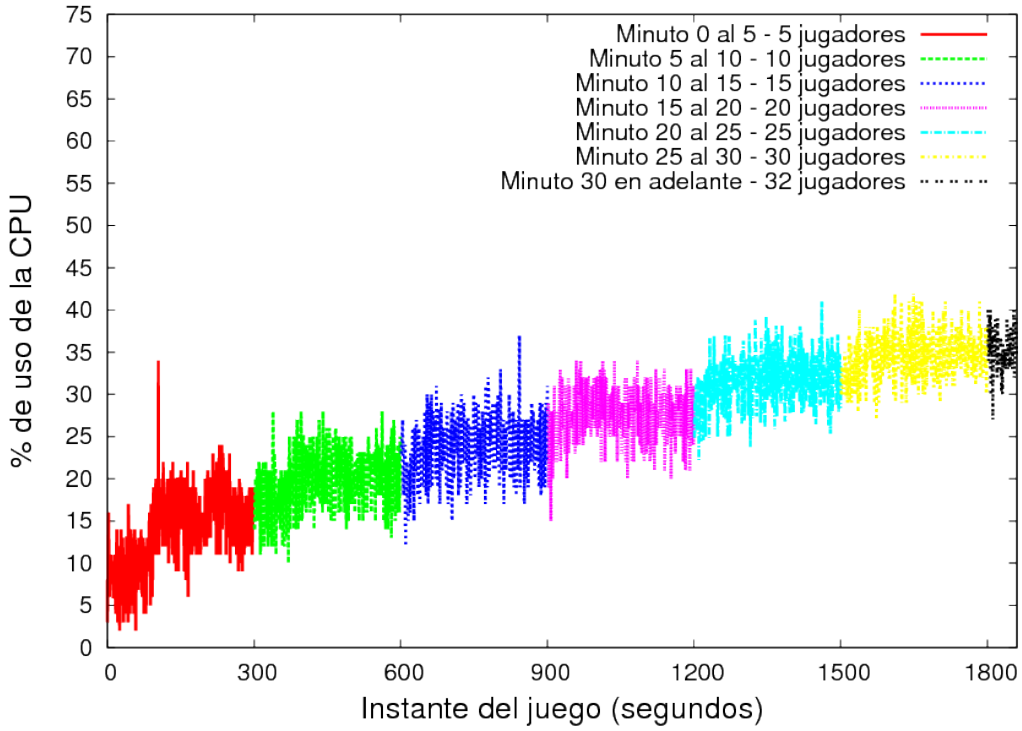


Figura 3.3: Porcentaje de uso de CPU en función del número de jugadores concurrentes.

rango medio de consumo de CPU para 10 jugadores oscila entre un consumo del 14,85 % para los 5 jugadores hasta los 20,9 % para los 10 jugadores; por tanto el incremento de 5 jugadores (de los 5 a los 10) provoca un incremento gradual del consumo de CPU del $6,08\,\%$.

Sin embargo, el nivel de estrés al que se somete el servidor es radicalmente distinto dependiendo de la actividad que generan los jugadores, es decir, si se trata de jugadores muy activos o inactivos. Los jugadores son considerados inactivos cuando, una vez conectados al juego e iniciado éste, no realizan ningún movimiento, ni acción. Por contra, los jugadores son considerados activos cuando, una vez conectados e iniciado el juego, realizan las acciones propias del juego; por ejemplo moverse por el mundo virtual, recoger objetos, disparar, etc. A modo de ejemplo, las Figuras 3.4 y 3.5 muestran el consumo de CPU del servidor, tanto para procesos de usuario (los del juego) como del sistema, cuando los 32 jugadores están inactivos o por contra activos, respectivamente.

Rango de consumo de CPU (%)	Número de jugadores concurrentes en el servidor
5	[0-14,85[
10	[14,85-20,9[
15	[20,9-25,2[
20	[25,2-28,45[
25	[28,45-32,04[
30	[32,04-35,2[
32	[35,2-38,65]

Tabla 3.3: Relación consumo de CPU con el número de jugadores concurrentes de la Figura 3.3.

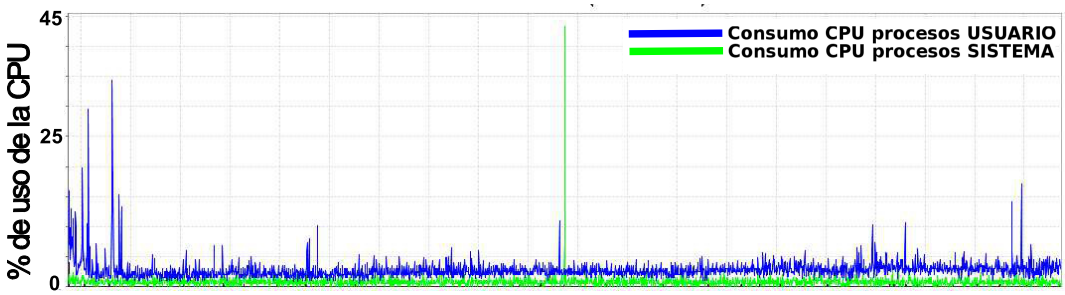


Figura 3.4: Uso de CPU de un servidor MMOFPS con 32 jugadores inactivos.

De los resultados mostrados en ambas figuras vemos como la actividad de los jugadores es determinante en el porcentaje de uso de CPU del servidor. Sin embargo cuando se gestiona una partida de jugadores activos hay un consumo significativo, tanto de procesos de usuario como del sistema. Aún así, de los datos reflejados en la Figura 3.5, se desprende que aún considerando todos los jugadores activos, el servidor sería capaz de gestionar la partida sin llegar a rebasar un uso de CPU del 45 %.

Finalmente remarcar, que tal y como otros artículos de la literatura apuntan [YC06], existe una fuerte correlación entre el consumo de la CPU y el número total de jugadores jugando de forma concurrente. Estas afirmaciones presentadas en otros trabajos de la literatura refuerzan nuestro análisis, puesto que hemos mostrado como el volumen y comportamiento de los jugadores (activos o inactivos), tienen una incidencia directa en el consumo de CPU del servidor, denotando una fuerte correlación entre ambos.

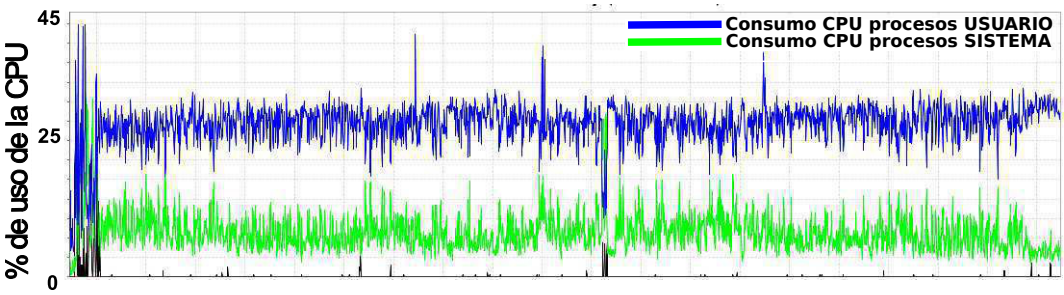


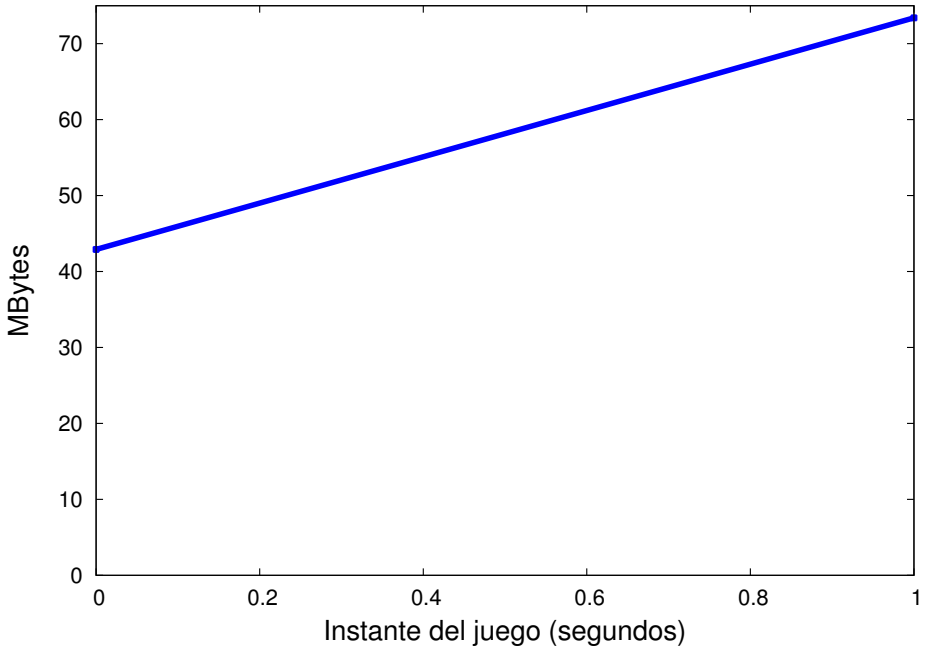
Figura 3.5: Uso de CPU de un servidor MMOFPS con 32 jugadores activos.

Cabe destacar que el análisis de los datos empíricos obtenidos de los servidores que gestionan el juego MMORPG *PlaneShift* nos han mostrado exactamente este mismo comportamiento, dado que también revelan que a medida que se incrementa el número de jugadores concurrentes del juego, la carga de los servidores asciende y por tanto, vemos como existe una relación directa entre la carga del sistema y los jugadores concurrentes del juego.

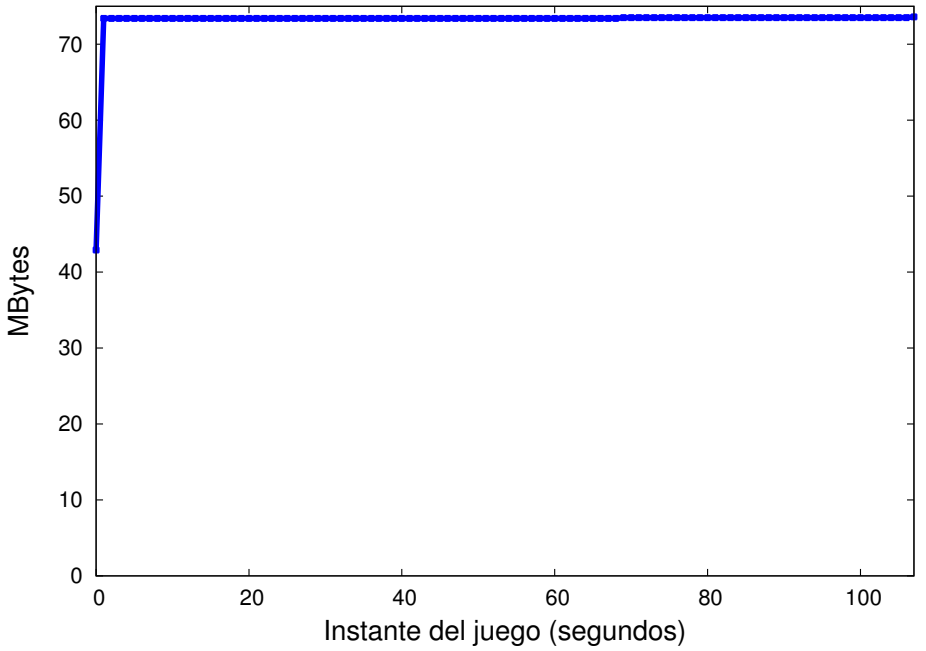
3.4. Consumo de memoria (RAM)

El análisis realizado del consumo de memoria del servidor, cuando hospeda un juego tipo MMOFPS, ha mostrado que éste es muy bajo y constante. Esto es debido a que el juego realiza una petición de memoria al sistema para crear la instancia del juego, pero una vez creada, no hay una relación entre el número de jugadores gestionados y el consumo de memoria, puesto que ésta será constante a lo largo del mismo. En la Figura 3.6 se muestra el consumo de memoria RAM para una partida de 32 jugadores. En concreto, la Figura 3.6a muestra como durante el primer segundo, el consumo aumenta hasta reservar la memoria necesaria para crear el juego; mientras que la Figura 3.6b muestra, como a partir del momento inicial, se hace un consumo constante en todas las etapas posteriores del juego. Naturalmente, al final del juego, de nuevo, se liberarán los recursos de memoria utilizados.

Los datos obtenidos demuestran que el servidor puede gestionar un alto volumen de jugadores, sin que la memoria RAM del sistema se vea afectada. En la Tabla 3.4 mostramos el consumo de memoria RAM del servidor en el



(a) Durante arranque ejecución.



(b) Durante toda ejecución.

Figura 3.6: Consumo de memoria RAM de una instancia de un MMOFPS con 32 jugadores durante el arranque y la ejecución de la partida.

Consumo medio de Memoria RAM	Número de instancias concurrentes en el servidor
75 MB	1
143 MB	2
210 MB	3
279 MB	4
368 MB	5

Tabla 3.4: Relación entre consumo de memoria RAM y el número de instancias concurrentes en el servidor.

caso de ejecutar de forma concurrente entre una y cinco instancias del juego. Se puede apreciar que la relación entre el consumo de memoria RAM y el número de instancias en el servidor es lineal.

Por tanto podemos concluir que el consumo final de memoria RAM, por parte de un juego MMOFPS, está determinado por el número de instancias de juego que el servidor maneja. Se ha comprobado además que este consumo de memoria es independiente del número de jugadores existentes en cada una de ellas. La razón es que el servidor debe cargar el mapa, con una serie de objetos tales como paisajes, muebles, vehículos, etc., en definitiva, estructuras de datos, que precisan ser cargadas en memoria en el momento de iniciar la partida.

3.5. Uso del ancho de banda

Respecto del uso de ancho del banda los resultados obtenidos han mostrado que en este tipo de juegos el comportamiento es similar al de la CPU, aunque en esto caso la linealidad se produce con una exactitud matemática y totalmente proporcional al número de jugadores concurrentes. A diferencia del consumo de CPU, en el caso del ancho de banda no existen grandes diferencias entre jugadores activos o inactivos, puesto que periódicamente todos los jugadores envían al servidor actualizaciones sobre su estado dentro del juego y el servidor debe actualizar esta información con el resto de jugadores de la partida. En las Figuras 3.7 y 3.8 se representan gráficamente el número de paquetes entrantes

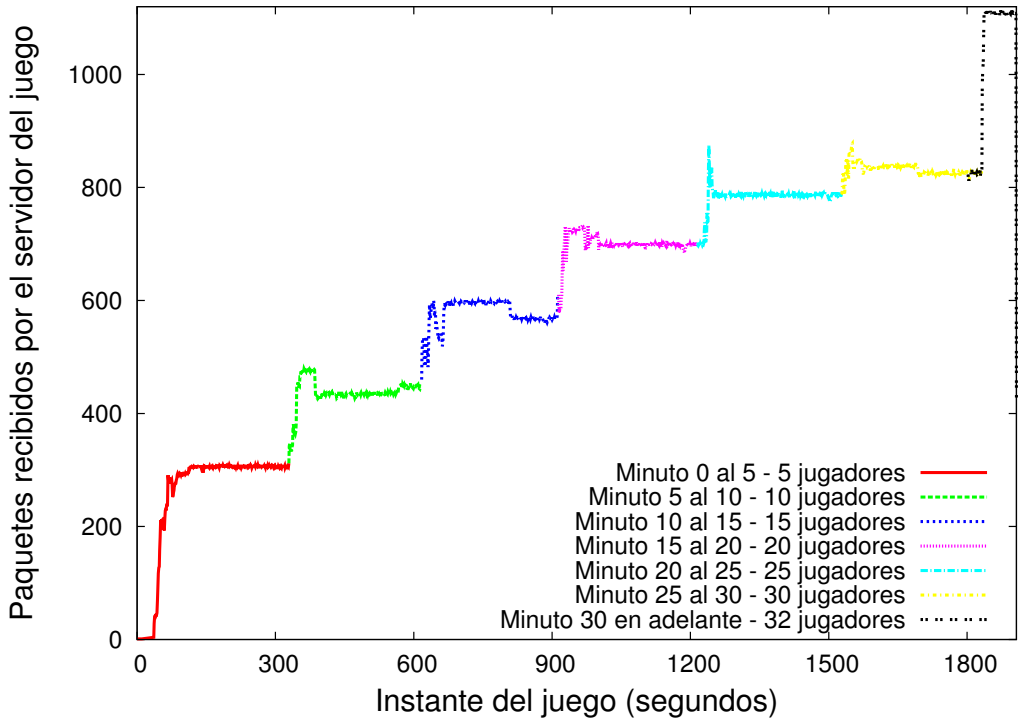


Figura 3.7: Número de paquetes entrantes de un servidor MMOFPS con 32 jugadores.

y salientes del servidor, cuando de forma escalada 32 jugadores, de una única instancia, se van conectando al juego. Con objeto de destacar esta linealidad, la Tabla 3.5 muestra, el número de paquetes salientes del servidor en función del número de jugadores conectados al juego. En esta tabla se observa que el número de paquetes que el servidor envía va en función del número de jugadores en línea. En este caso, cada jugador en el servidor se traduce como 20 paquetes salientes y por tanto, 32 jugadores equivalen a un envío total de 640 paquetes.

Con objeto de analizar la conveniencia de usar un servidor *doméstico* para albergar una instancia de un juego tipo MMOFPS o MMORPG se han realizando una serie de cálculos, teniendo en cuenta que las conexiones de alta velocidad a Internet (ADSL) son cada día más accesibles para todos los públicos. Atendiendo a este hecho y a los resultados obtenidos en el presente análisis, se han tomado como referencia los siguientes valores:

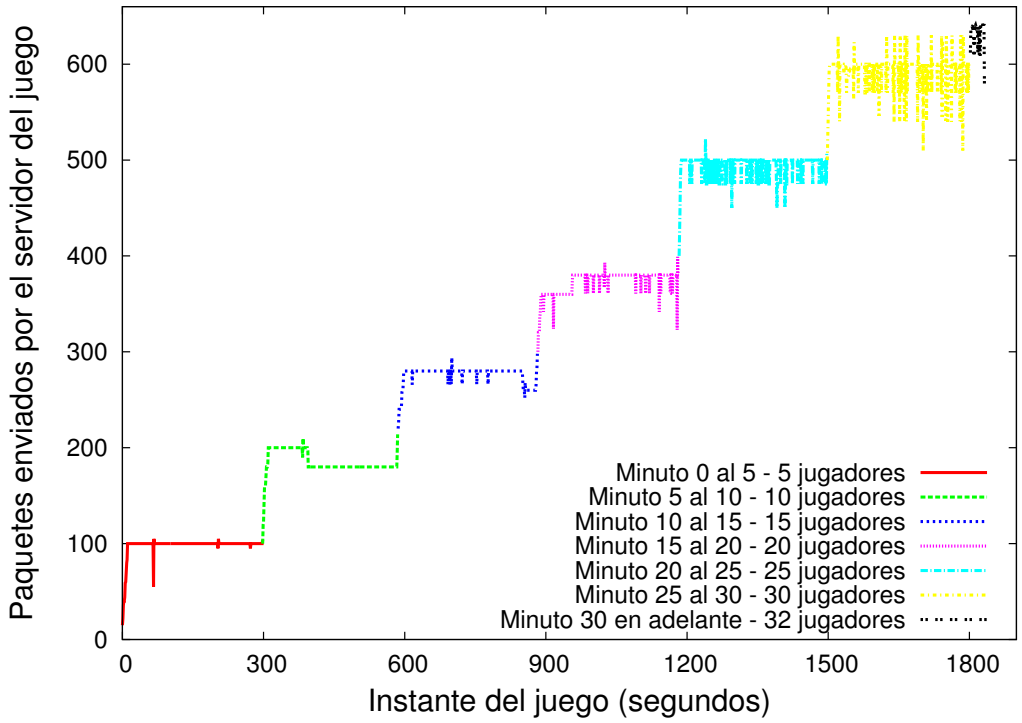


Figura 3.8: Número de paquetes salientes de un servidor MMOFPS con 32 jugadores.

- Una línea ADSL convencional de 512 kilobits por segundo (kbps) puede enviar teóricamente 512.000 bits por segundo (bps).
- Según las observaciones realizadas de forma empírica con la monitorización de juegos tipo MMOFPS, mostradas en la captura de imagen de la Figura 3.9, podemos determinar que:
  - Cada paquete enviado tiene un tamaño medio de 141 Bytes (1.128 bits) (129,5 Bytes de media según [FCFW02]).
  - El servidor envía dos veces por segundo a cada jugador del juego un paquete, siendo el volumen de datos 2.256 bits para cada jugador de juego.
  - Teniendo en cuenta que el número medio de jugadores por instancia de juego es de 32 jugadores, el volumen total será de: $32 \cdot 2.256 = 72.192$ bits/segundo.

Número jugadores en el servidor	**Número paquetes salientes del Servidor**
1	20
2	40
3	60
4	80
5	100
20	400
30	600
31	620
32	640

Tabla 3.5: Relación entre jugadores concurrentes y el número de paquetes enviados por el servidor.

Por tanto, la tasa de envío representa un 14,1 % ($(\frac{72.192}{512.000} \cdot 100)$) de la capacidad total de tráfico que puede soportar la línea. Por consiguiente, podemos concluir que no hay problemas de red, en el caso que un ordenador *doméstico*, con una conexión *ADSL* convencional, sea el servidor del juego.

Finalmente, con objeto de visualizar la gran cantidad de ancho de banda disponible en comparación al uso que requieren los juegos tipos MMOFPS, la Figura 3.10 muestra la representación gráfica del procedimiento analítico comentado en las líneas previas, es decir, muestra cual sería el consumo de ancho de banda del servidor de un juego, en escala logarítmica, en relación al incremento del tamaño del paquete y del número de jugadores concurrentes. Se han determinado tamaños de paquete de 141, 212 y 282 bytes, respectivamente y una tasa de envío de dos paquetes por segundo para cada jugador. Los resultados obtenidos reflejan la gran diferencia existente entre el máximo consumo de ancho de banda (282 bytes/paquete $\cdot$ 32 jugadores = 144.384 bps) con respecto el ancho de banda total (512.000 bps), lo que representa un 28,2 % del ancho de banda total.

En conclusión, los datos obtenidos en este análisis revelan que el factor que influye de forma directa en el consumo de ancho de banda del servidor del juego es el número de jugadores concurrentes que éste debe gestionar, independientemente de su distribución en una o más instancias de juego, ya

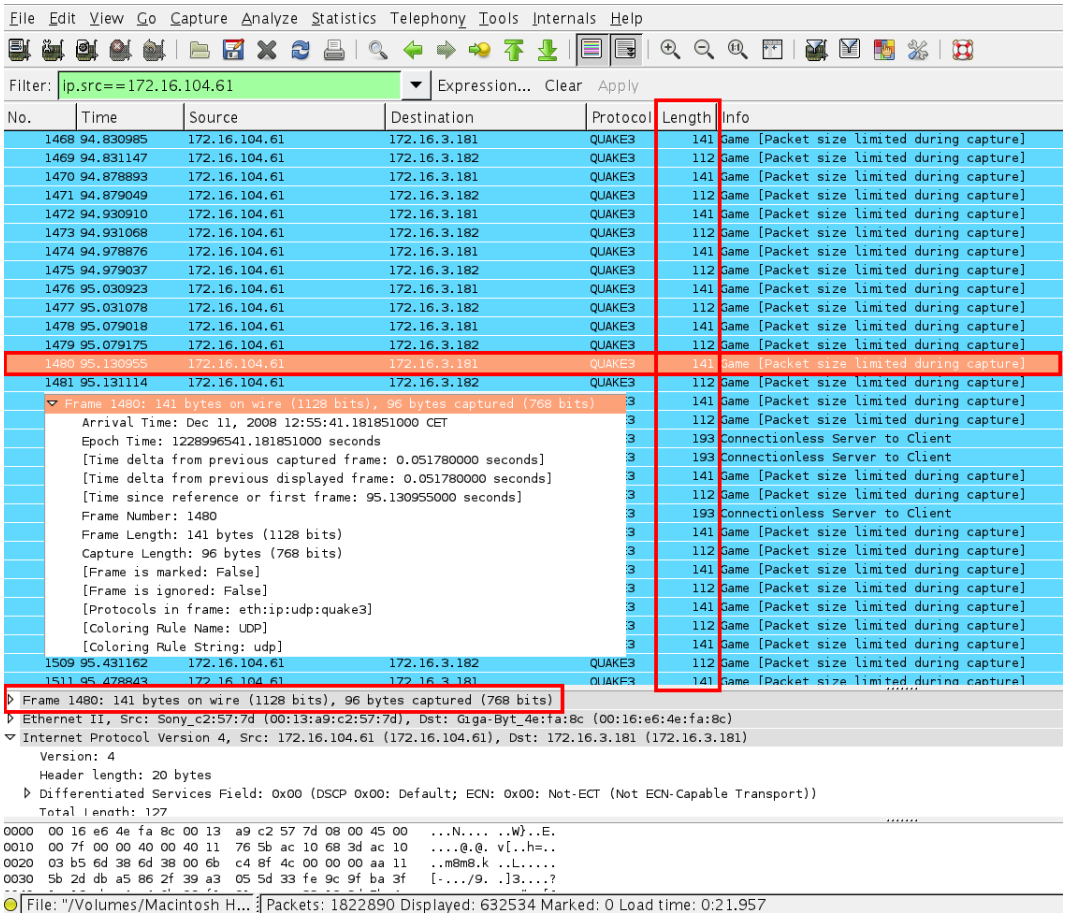


Figura 3.9: Captura del analizador de redes *Wireshark* de un servidor MMOFPS con 32 jugadores activos, filtrado de paquetes del juego del servidor a los clientes.

que el servidor efectúa una comunicación directa con los jugadores mediante el uso del protocolo *UDP*, sin importar la instancia del juego a la que pertenecen.

3.6. Grado de satisfacción (*GdS*)

Aparte del análisis de necesidad de recursos del sistema, para la ejecución de juegos MMOFPS, visto hasta el momento, se ha realizado también un análisis de la satisfacción de los jugadores respecto de los juegos. A dicha apreciación la hemos denominado *Grado de Satisfacción* (*GdS*) y es un valor entre 1 y 5, de menor a mayor *GdS*, que han proporcionado los propios jugadores preguntados

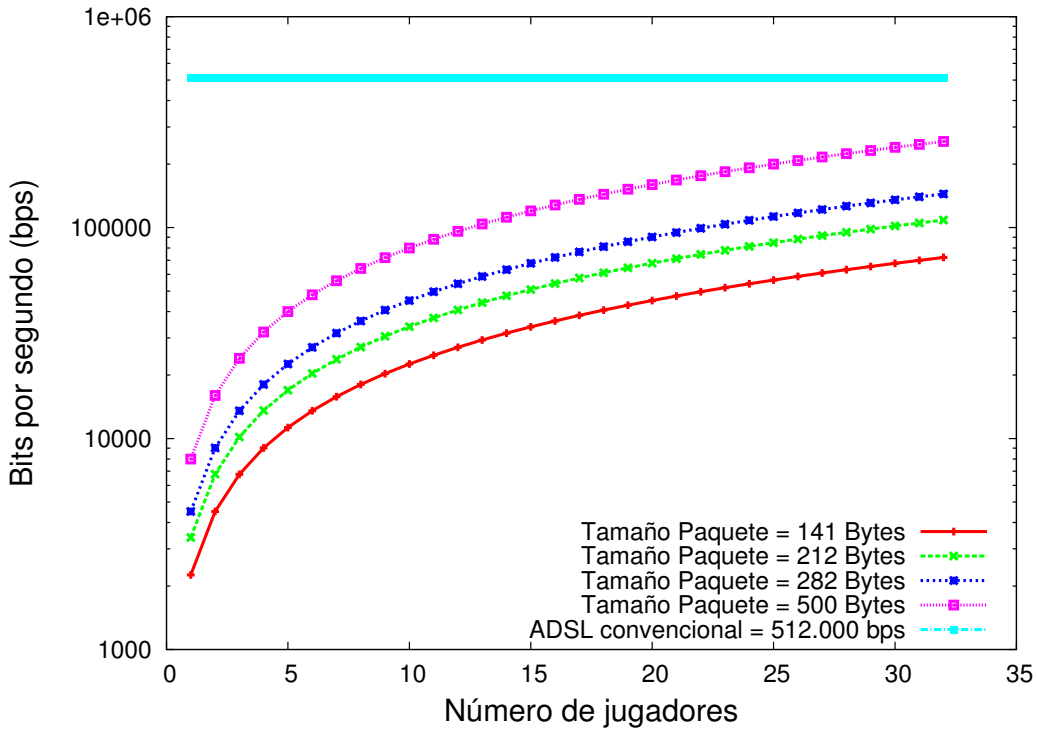


Figura 3.10: Consumo del ancho de banda del servidor con distinto número de jugadores y tamaño de paquete.

mediante encuesta. Si bien el *GdS* es un valor subjetivo, su conocimiento nos sirve para ver como influyen los valores de rendimiento, que sí son medibles objetivamente, en la apreciación de los jugadores.

En la Tabla 3.6 presentamos el estudio del grado de satisfacción medio de 100 jugadores distintos pertenecientes a la red local y global, que durante 3 meses han estado jugando de forma asidua a los juegos tipo MMOFPS habilitados para la realización de esta tesis doctoral y gestionados por el servidor *ProLiant* para la red global y el servidor *doméstico* para la red local. El estudio se centra en analizar el *GdS* de los jugadores en función del *delay* de las comunicaciones (latencia o *ping*) medio, que éstos tienen respecto al servidor del juego. Tal y como se aprecia en la Tabla 3.6, los jugadores con la latencia más baja y por tanto, con mejor respuesta a las acciones que ejecutan, son en orden ascendente los jugadores de: *Laboratorio B* (*LAB. B*), *Laboratorio A* (*LAB. A*), *España* y *Sudamérica*. De este modo, los jugadores que experimentaron un

	LAN		INTERNET	
	LAB. A	**LAB. B**	**ESPAÑA**	**SUDAMÉRICA**
Ping (ms)	65	53	78	94
GdS (1-5)	4	4,75	4,5	3,75

Tabla 3.6: Ping y grado de satisfacción de los jugadores ubicados en red local o global.

mayor grado de satisfacción fueron los jugadores del *Laboratorio B*, en el caso de partidas en red local, con un *GdS* igual a 4,75 y los jugadores de *España*, en el caso de partidas ejecutadas a través de *Internet*, con un *GdS* de 4,5. Por último, los jugadores conectados desde *Sudamérica* siempre tienen latencias altas respecto al servidor del juego, comparado con el resto de jugadores conectados desde España, siendo de este modo, los jugadores más perjudicados por la latencia, hecho que queda de manifiesto en su grado de satisfacción de 3,75 en la partida global a través de *Internet*.

Se ha comprobado también, de forma experimental, que aunque todos los jugadores analizados tenían una latencia muy inferior a la que la literatura determina como aceptable (no superior a los 160 milisegundos), vemos como existe otro factor importante en este tipo de juegos: la latencia del resto de jugadores de una partida en relación a la de uno mismo. Este factor tiene gran incidencia puesto que una latencia, por muy baja que sea, y por tanto un factor positivo a priori, no se traducirá en un valor alto de *GdS* por parte del jugador, si el resto de jugadores tienen una latencia aún inferior, puesto que las acciones del resto de jugadores se procesarán con más rapidez y por tanto, tendrán un efecto más inmediato. En definitiva, las latencias del juego tienen un efecto inmediato en el *GdS* de los jugadores, más aún cuando hay mucha diferencia entre valores de latencias de los jugadores del mismo juego.

Finalmente hemos buscado aquellos valores de latencia a partir de los cuales el juego pierde todo interés para el jugador. Para ello, hemos forzado en una prueba local, latencias superiores a los 200 milisegundos, comprobando como la diversión de los jugadores de los MMOFPS decrecía de forma drástica, debido a que las órdenes que requerían más rapidez de ejecución fracasaban por culpa de este retardo (por ejemplo disparos). Por tanto hemos constatado en primera

persona, que tal y como indican Knutsson et al. en [KLXH04], los juegos tipo MMOFPS requieren latencias inferiores a los 160 milisegundos para asegurar una buena calidad de servicio.

3.7. Valoración de los resultados

A partir de los datos obtenidos de los servidores del MMORPG *Planeshift*, así como de los resultados obtenidos en el análisis de comportamiento de los juegos MMOFPS, que hemos realizado en el presente capítulo, se pueden destacar, a modo de resumen, las siguientes valoraciones:

1. **MMOFPS y MMORPG**. Hemos visto que, en términos generales, los resultados de ambos MMOGs son extrapolables de un tipo de juego al otro. Hemos identificado que el factor principal que determina la carga de los sistemas que gestionan este tipo de juegos es el número de jugadores concurrentes.

2. **Requerimiento de recursos**. Las necesidades de CPU y ancho de banda tienen una relación directa con el número de jugadores concurrentes actuando en el sistema. Por contra, el uso de memoria RAM depende del número de partidas iniciadas, teniendo poca influencia cuántos jugadores hay en dichas partidas.

3. **Capacidad de los ordenadores actuales**. Se ha comprobado que el conjunto de recursos de cómputo necesarios para dar servicio a una partida convencional de juegos MMOFPS, quedan satisfechos en un ordenador doméstico actual. Esto permite validar su utilización como servidor temporal de las partidas de MMOGs, que es uno de los objetivos de esta tesis.

4. **El grado de satisfacción de los jugadores**. Es directamente dependiente del valor de latencia de los jugadores con respecto al servidor del juego, y aún más, depende de como es el valor relativo de la latencia de cada jugador con respecto a la del resto de jugadores de la partida.

Una vez analizado el comportamiento de los servidores de juegos tipo MMOG, centrando el foco en el rendimiento y la fluctuación de los parámetros de cómputo (CPU, ancho de banda y memoria RAM), así como en el estudio de la incidencia en la calidad de servicio o *grado de satisfacción* de los jugadores bajo distintas condiciones de latencia, podemos concluir que el número de jugadores y el número de instancias de juegos a gestionar, son los factores clave que influyen de forma determinante en la correcta gestión de este tipo de juegos bajo un entorno Cliente-Servidor tradicional.

Asimismo, la experimentación realizada ha mostrado como el volumen de recursos de cómputo ociosos en un ordenador doméstico son suficientes para gestionar, sin problemas, una única instancia (o partida) de juego, hecho que corrobora la hipótesis inicial de esta tesis para afrontar los problemas de escalabilidad de los juegos MMOG considerando una arquitectura híbrida, en la que se saque provecho de los recursos no usados en los ordenadores de los propios jugadores.

Atendiendo a los requerimientos de las aplicaciones MMOFPS y MMORPG analizados en este capítulo, en los Capítulos 4 y 5 se propondrá un modelo híbrido que sea plausible para este tipo de juegos, junto con los mecanismos necesarios para gestionar de modo eficiente a los jugadores e instancias del juego.

Capítulo 4

Arquitectura híbrida CS/P2P para juegos MMOFPS

En este capítulo se expone la arquitectura propuesta en el presente trabajo de tesis, para dar servicio de un modo eficiente y escalable a los juegos MMOFPS. Dicha arquitectura se basa en un modelo híbrido formado por un entorno centralizado Cliente-Servidor (CS), junto con un entorno distribuido que actúa bajo el paradigma Peer-to-Peer (P2P).

Junto con la propuesta del entorno se realiza también una descripción detallada del sistema y de su funcionamiento, es decir, de los mecanismos o algoritmos aplicados para la gestión del mismo. Al conjunto formado por el modelo del sistema y su funcionamiento, lo hemos denominado *OnDeGaS*, que proviene del inglés *On Demand Game Service*, es decir, servicio de juegos bajo demanda.

Finalmente, el rendimiento del entorno *OnDeGaS*, tanto en lo que se refiere a su escalabilidad, como a la calidad de servicio que aporta a los jugadores del mismo, ha sido evaluado mediante simulación.

4.1. Descripción del sistema

En el Capítulo 1 de la presente memoria se ha comentado que el modelo tradicional de ejecución de los juegos MMOFPS está basado en el paradigma Cliente-Servidor. En la Figura 4.1 se ilustra un ejemplo del mismo, en el que un

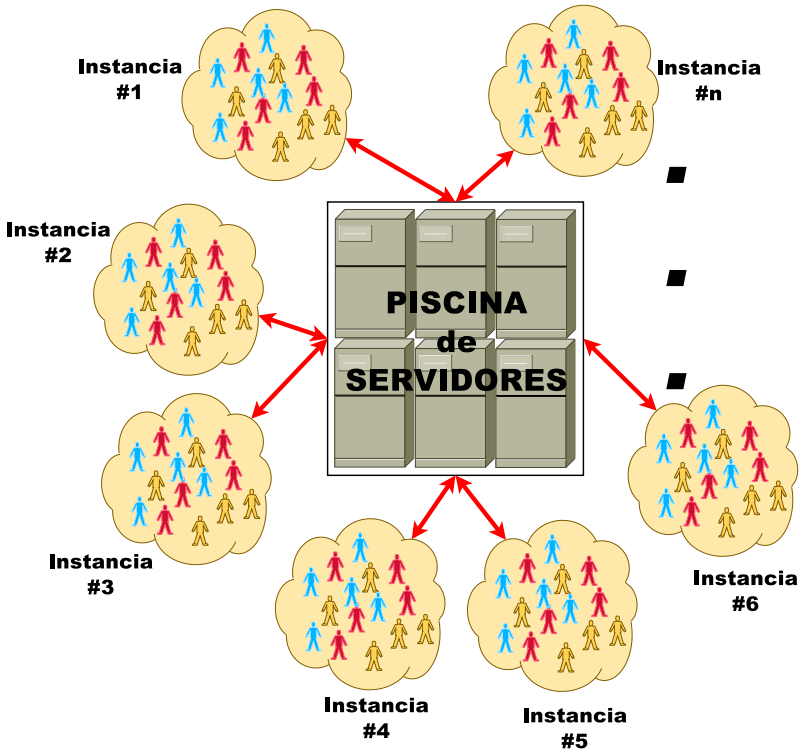


Figura 4.1: Modelo de gestión Cliente-Servidor de los juegos tipo MMOFPS.

conjunto de jugadores (clientes) agrupados en distintas instancias o partidas son gestionados por un servidor, que en este caso lo hemos ilustrado formado por una piscina de servidores ya que es una configuración habitual. Sin embargo, el uso del modelo Cliente-Servidor para los MMOFPS tiene ciertos problemas asociados, entre los cuales destaca la gestión de la escalabilidad al aumentar el número de jugadores. Con objeto de solucionar dicho problema, en esta tesis se presenta el modelo híbrido *OnDeGaS*, representado en la Figura 4.2. El sistema consta de dos áreas principales: el área central y el área distribuida. El área central se usa tanto para la ejecución de los MMOFPS en situaciones de no sobrecarga, como para la gestión de los servicios centrales del entorno, mientras que el área distribuida es explotada en situaciones de sobrecarga del sistema, momentos en los que se crea, bajo demanda, un conjunto de zonas para albergar a las instancias no gestionadas por el servidor central.

Por consiguiente, el área central se dedica a controlar todo el sistema, al mismo tiempo que da servicio a los jugadores. Los componentes del área central

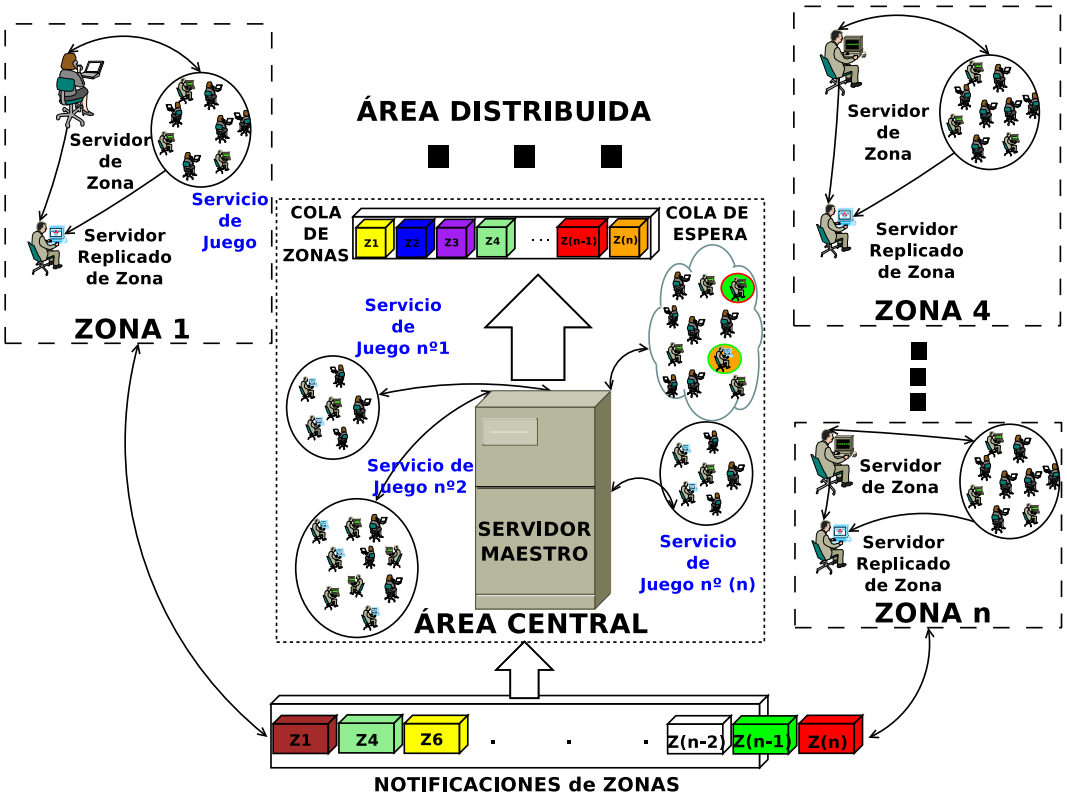


Figura 4.2: Visión global del sistema *OnDeGaS*.

son los siguientes:

- *Servidor Maestro (SM)*. Se trata del servidor principal del sistema y actúa de punto de entrada al juego. Por tanto, todos los jugadores que quieran jugar intentarán la conexión con el *SM*. Este *SM* se encarga de gestionar el juego en el área central, siempre que su carga global, es decir el número de jugadores concurrentes que gestiona, no sobrepase un cierto umbral, momento a partir del cual, los jugadores serán distribuidos hacia el área distribuida.

- *Cola de Espera (CE)*. Se trata de un espacio lógico usado por el *SM* con el fin de encolar aquellos jugadores que no pueden ser servidos debido a situaciones de sobrecarga. Es un estado transitorio para los jugadores, que una vez encolados en ella, serán distribuidos en un espacio corto de tiempo.

- *Cola de zonas (CZ).* Se trata de un espacio lógico usado por el *SM* con el fin de almacenar información actualizada de las zonas creadas; por ejemplo, cuales son los servidores de estas zonas, con el fin de poder contactar con ellos en el futuro. Por tanto, esta información podrá ser usada para distribuir jugadores de la *CE* a zonas ya creadas que tienen posibilidad de albergar nuevos jugadores.

- *Notificaciones de zonas (Z^N).* Es un espacio lógico formado por el conjunto de zonas que han enviado una notificación al *SM*, con el fin de informar que las partidas que se llevaban a cabo en las mismas han finalizado.

El área distribuida está compuesta por un conjunto de zonas que ejecutan partidas al margen del *SM*. Cada *Zona* i, Z_i, que forma parte del sistema, consta de los siguientes componentes:

- *Servidor de Zona (SZ).* Es el servidor de la *Zona*, que será seleccionado de entre un grupo de jugadores ubicados en la *CE*.

- *Servidor Replicado de la Zona (SRZ).* Es el servidor replicado de la *Zona*. El SRZ está a cargo de la tolerancia a fallos de la *Zona*, es decir, deberá ejecutar las tareas de su *SZ* en caso que éste se desconecte.

En cuanto a los juegos, que serán ejecutados bajo la plataforma presentada, distinguimos los siguientes elementos:

- *Jugador (J_i).* Un Jugador número i, J_i, es un cliente que se conecta al sistema con el fin de participar en alguna partida de tipo MMOFPS ofertada por el *SM*.

- *Servicio de Juego (SJ).* Se trata de una instancia (partida) del juego, donde un conjunto de jugadores se hallan conectados con el fin de participar en la misma. Cada *SJ* será hospedado por un servidor, que puede ser o bien el *SM* o un *SZ*. Los estados que puede tener un *SJ* son dos:

    1. *Activo*: si la partida aún sigue sucediéndose, es decir, los jugadores están interactuando entre ellos para lograr los objetivos del juego.

2. *Inactivo*: si la partida ha finalizado debido a la desconexión de los jugadores o bien debido a la consecución de los objetivos del juego, es decir, debido a las normas internas del juego.

Normalmente, el número máximo de jugadores por *SJ* en un MMOFPS es de unas pocas decenas, mientras que la duración media de las partidas oscila entre unos minutos hasta, como máximo, unas pocas horas.

4.2. Funcionamiento del sistema

El funcionamiento del sistema *OnDeGaS* está basado en una ejecución que combina el modelo híbrido Cliente-Servidor en el área central, con el paradigma P2P en el área distribuida. La idea principal está basada en la ejecución de un conjunto de *SJ*s en el área central hasta que ésta no es capaz de gestionar nuevos jugadores, es decir, hasta que el *SM* alcanza su capacidad máxima de servicio. Llegado a este punto, en el que el *SM* no es capaz de gestionar nuevos jugadores, nuevas zonas serán creadas de forma dinámica en el área distribuida, con el fin de evitar esperas prolongadas a los jugadores que quieren jugar nuevas partidas, y dotar al sistema de mayor escalabilidad.

En función de como se explotan los recursos de los ordenadores de los jugadores del sistema, aplicaremos una metodología de funcionamiento homogénea, si consideramos que todos los ordenadores de los jugadores tienen las misma características o bien un funcionamiento heterogéneo, si consideramos que los ordenadores de los jugadores tienen distintas capacidades de cómputo. La diferencia principal entre ambas alternativas es que mientras la alternativa homogénea contempla la ejecución de un solo *SJ* (instancia o partida) por cada Zona del área distribuida, la alternativa heterogénea considerará la ejecución de uno o más *SJs* por cada *Zona* del área distribuida, en función de los núcleos (cores) libres que tienen los SZ y SRZ.

A continuación, vamos a detallar el funcionamiento de ambas alternativas.

4.2.1. Alternativa homogénea

En el caso de la alternativa homogénea, cada *Zona* creada en el área distribuida será capaz de gestionar un único *SJ*, es decir, una única partida.

Algoritmo 4.1: Algoritmo principal de funcionamiento del sistema OnDeGaS (alternativa homogénea).

Input: $\forall J_i$ conectando al SM
Input: $Z^N = \{Z_i, Z_{i+1}, \ldots, Z_{n-i}, Z_n\}$ notificando al SM

```
1  while True do
2      switch SM.estado() do
3          case SM.estado() == SOBRECARGADO
4              if ∃ J_i then SM.encolar(CE,J_i);
5              if (CE.tamaño() ≥ α) or (CE.tiempo_D_espera() ≥ β)
               then Z_i = SM.crearZona(CE);
6          end
7          case SM.estado() == NO SOBRECARGADO
8              if CE.tiempo_D_espera() ≥ β then
9                  forall J_i in CE do
10                     SM.aceptar(J_i);
11                 end
12             if Z^N ≠ ∅ then SM.reaceptar(Z^N);
13             if ∃ J_i then SM.aceptar(J_i);
14         end
15     end
16 end
```

El funcionamiento del sistema está controlado por la continua ejecución del Algoritmo 4.1, que tiene dos flujos de datos de entrada: nuevas conexiones de jugadores J_i y aquellas zonas (Z_i) que han finalizado sus respectivos *SJs* y pretenden entrar de nuevo en el área central controlada por el *SM*.

En cada iteración del Algoritmo 4.1, el *SM* comprueba su estado (*SM.estado()*). En caso que esté sobrecargado, las nuevas peticiones de conexión de jugadores, serán encoladas en la *CE*. Después de encolar, el *SM* comprueba si el número de jugadores que esperan en la *CE* es mayor o igual a un valor predefinido α, o si los jugadores de esta cola han permanecido esperando por un tiempo superior o igual a un parámetro predefinido β. Si alguna de estas dos condicio-

nes es cierta, entonces el *SM* creará una nueva *Zona*, Z_i mediante la función *SM.crearZona()* descrita en el Algoritmo 4.2.

La función *SM.crearZona()*, tiene como objetivo principal la búsqueda del *SZ* y *SRZ* óptimos basándose en la latencia de los jugadores de la *CE* respecto del *SM* (en el apartado de detalles de la implementación en la Sección 4.2.3 discutimos los detalles de la función de búsqueda). Una vez encontrados los jugadores que actuarán como *SZ* y *SZR*, el resto de jugadores de la *CE* son enlazados con ambos *SZ.aceptar()* y *SRZ.aceptar()*. Esto asegura que tanto el *SZ* como el *SRZ* tengan la misma copia del estado del juego y por tanto, una posible migración entre servidores, debido a una desconexión del *SZ*, puede ser ejecutada de forma transparente a los jugadores ubicados en la Zona. De este modo, se dota al sistema de un mecanismo de tolerancia a fallos. Los detalles de implementación de dicho mecanismo se ampliarán en la Sección 4.2.3.

Algoritmo 4.2: Algoritmo de creación de una Zona del sistema *OnDeGaS*.

Input: CE
Output: Z_i

```
1 SM.crearZona(CE):
2 begin
3 |   SZ, SRZ = SM.búsqueda(CE);
4 |   foreach J_i ∈ CE do
5 |   |   SZ.aceptar(J_i);
6 |   |   SRZ.aceptar(J_i);
7 |   end
8 |   Z_i = {SZ ∪ SRZ};
9 end
```

En el caso que el *SM* no esté en una situación de sobrecarga, el Algoritmo 4.1 evalúa los siguientes tres condicionales:

- La primera condición evalúa si el tiempo de espera de los jugadores encolados en la *CE* es mayor o igual al parámetro β; en cuyo caso, los jugadores emplazados en la *CE* serán aceptados por el *SM* para que puedan jugar en el área central (*SM.aceptar()*). La aceptación de jugadores encolados en la *CE* se lleva a cabo como si de una cola *FIFO* se

tratara, es decir, el primer jugador encolado en la *CE* será el primer jugador aceptado por el *SM*, siempre y cuando éste tenga capacidad para albergar un nuevo jugador. De este modo, el sistema premia a los jugadores más pacientes. En los casos en los que el *SM* no pueda aceptar a todos los jugadores ubicados en la *CE*, entonces el sistema reiniciará el tiempo de espera a cero de nuevo para reintentar la operación pasados β segundos. No obstante, cabe destacar que con las situaciones de sobrecarga contempladas, el porcentaje de casos en los que se sucede esta situación es despreciable. Además, el número de jugadores afectados por esta eventualidad es reducido, puesto que el el número de jugadores que pueden permanecer en espera está limitado a un máximo de α.

- El segundo condicional es usado para dar preferencia de conexión al área central a aquellos jugadores que provienen de zonas que han notificado al *SM* que sus juegos han finalizado. Estos casos suceden cuando una o más zonas del área distribuida han finalizado su partida y los jugadores de dichas zonas permanecen a la espera de iniciar una nueva partida. Por tanto, si el conjunto de zonas que han notificado al *SM*, Z^N, resulta ser un conjunto no vacío, el *SM* ejecuta la función *reaceptar*, mostrada en el Algoritmo 4.3. El procedimiento que sigue este algoritmo es el mismo para todas y cada una de las zonas Z_i, que se encuentran dentro del conjunto de zonas notificantes (Z^N). De este modo, el *SM* reaceptará la Zona Z_i, si tiene espacio suficiente en el área central (*SM.espacioLibre()*) para albergar a todos los jugadores de Z_i (*Z_i.nJugadores()*). Una vez aceptados todos los jugadores contenidos en Z_i, ésta será eliminada del conjunto Z^N (*SM.eliminar()*). Cabe remarcar que el algoritmo trata de priorizar la reaceptación en el área central de los jugadores de una misma Z_i, para evitar fragmentar jugadores pertenecientes a una misma partida. En los casos en los que el *SM* no disponga de espacio libre suficiente para albergar a todos los jugadores de una Z_i notificante, un mensaje de denegación de servicio será enviado a dicha Z_i, momento a partir del cual, los jugadores de la misma podrán decidir si empezar a jugar otro *SJ* en el área distribuida o desconectarse del juego.

- El último condicional evalúa la existencia de nuevos jugadores que requieren conectarse al sistema. Estos nuevos jugadores serán aceptados por el *SM* si éste no está sobrecargado.

Algoritmo 4.3: Algoritmo de reaceptación de Zonas en el área central del sistema *OnDeGaS*.

Input: Z^N

```
1  SM.reaceptar(Z^N):
2  begin
3      foreach Z_i ∈ Z^N do
4          if SM.espacioLibre() < Z_i.nJugadores() then SM.denegar(Z_i);
5          else
6              foreach J_i ∈ Z_i do SM.aceptar(J_i);
7              SM.eliminar(Z_i);
8          end
9      end
10 end
```

4.2.2. Alternativa heterogénea

A diferencia de la alternativa homogénea, descrita en la sección anterior, en este caso la política de distribución de *SJ*s tiene en cuenta que los nodos que forman el área distribuida puedan ser heterogéneos. De este modo, consideraremos como atributos de cómputo primordiales, tanto la latencia con respecto al *SM* (considerada como único atributo en el sistema homogéneo), como el número de cores o núcleos que forman la CPU de las máquinas de los jugadores. Por tanto, en esta alternativa heterogénea se tendrán en cuenta ambos atributos para encontrar los *SZ* y *SRZ* más adecuados para el proceso de creación de las zonas entre el conjunto de jugadores encolados en la *CE*. No consideraremos la memoria como parámetro de cómputo esencial, puesto que tal y como hemos introducido en el capítulo anterior, no se trata de un parámetro crítico en los juegos MMOFPS.

Otra de las diferencias de la versión que contempla la heterogeneidad, respecto a la homogénea, recae en el número de *SJ*s que podrán ser ejecutados

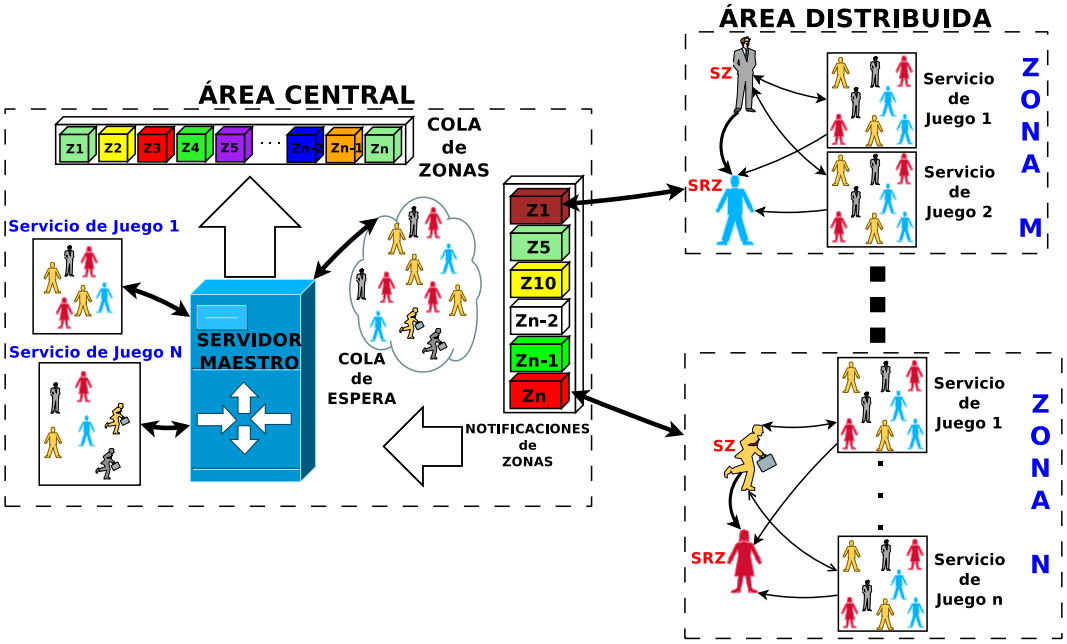


Figura 4.3: Visión global del sistema *OnDeGaS* (alternativa heterogénea).

en cada Zona, que en este caso podrán ser hasta un total de N *SJ*s como máximo, siendo N el número de cores disponibles en el *SZ*. Por tanto, en una Zona podrá ejecutarse de forma concurrente más de un *SJ* tal y como ilustra la Figura 4.3.

El funcionamiento de esta alternativa heterogénea es controlado por la ejecución continuada del Algoritmo 4.4, el cual tiene como la versión homogénea, dos flujos de entrada de datos: nuevas conexiones de jugadores (J_i) y aquellas zonas (Z_i) que han finalizado sus respectivos *SJ* y pretenden entrar de nuevo al área central controlada por el *SM*. En cada iteración, el *SM* comprueba su estado de carga (*SM.estado()*). Teniendo en cuenta el estado del *SM*, se contemplan los dos casos que comentamos a continuación:

1. **Estado de sobrecarga.** Si el *SM* está sobrecargado, entonces los nuevos jugadores entrantes (J_i) serán encolados en la *CE*. A continuación, el *SM* comprueba si el número de jugadores de la *CE* es mayor o igual que un valor predefinido α, o si el tiempo de espera desde la inserción del primer jugador en la *CE* es igual o mayor que un valor predefinido β.

   Si se cumple alguna de las condiciones comentadas, el algoritmo trata

Algoritmo 4.4: Algoritmo principal de funcionamiento del sistema *OnDeGaS* (alternativa heterogénea).

Input: $\forall J_i$ conectando al SM
Input: $Z^N = \{Z_i, Z_{i+1}, \ldots, Z_{n-i}, Z_n\}$ notificando al SM

```
1  while True do
2      switch SM.estado() do
3          case SM.estado() == SOBRECARGADO
4              if ∃ J_i then SM.encolar(J_i,CE);
5              if ((CE.tamaño() ≥ α) or (CE.tiempo_D_espera() ≥ β))
               then
6                  if (SM.distribuir_Jugadores(CZ,CE) == FALSE) then
7                      Z_i=SM.crearZona(CE);
8                      CZ=CZ+{Z_i};
9
10
11         end
12         case SM.estado() == NO SOBRECARGADO
13             if CE.tiempo_D_espera() ≥ β then
14                 forall J_i in CE do
15                     SM.aceptar(J_i);
16                 end
17             if Z^N ≠ ∅ then SM.reaceptar(J_i);
18             if ∃ J_i then SM.aceptar(J_i);
19         end
20     end
21 end
```

de distribuir a todos los jugadores de la *CE* mediante la función *distribuir_Jugadores(CZ,CE)*. La función *SM.distribuir_Jugadores*, descrita en el Algoritmo 4.5, busca aquellas zonas existentes en la *CZ* que tienen como mínimo un núcleo libre (*SZ.coresLibres()* y *SRZ.coresLibres()*). En caso de que exista más de una Zona con al menos un núcleo libre, tanto en el *SZ* como en el *SRZ*, entonces la función seleccionará aquella que tenga el valor de latencia más bajo entre el *SZ* y *SM* (*SM.latenciaMínima()*). A continuación los *SZ* y *SRZ* aceptarán los nuevos jugadores provenientes de la *CE*, retornando finalmente un valor *booleano*, siendo verdadero (*TRUE*) si los jugadores se han distribuido a una Zona existente, o falso

Algoritmo 4.5: Función de distribución de jugadores a zonas creadas del sistema *OnDeGaS* (alternativa heterogénea).

```
   Input: CZ,CE
   Output: Boolean
1  Zonas Disponibles = ZD = ∅;
2  SM.distribuir_Jugadores(CZ,CE):
3  begin
   |                                  /* Bluce ejecutado al inicio... */
4  |  forall ((SZ and SRZ) ∈ Z_i) ∈ CZ do
5  |  |  if (SZ.coresLibres() and SRZ.coresLibres()) then
6  |  |  |  ZD = ZD + {Z_i};
7  |  |
8  |  end
   |                        /* y periódicamente para actualizar datos. */
9  |  if (ZD != ∅) then
10 |  |  Z_i=SM.latenciaMínima(ZD);
11 |  |  Z_i.aceptar(CE);
12 |  |  return TRUE;
13 |  else
14 |  |  return FALSE;
15 |  end
16 end
```

(*FALSE*), si no existe ninguna Zona que cumpla con las características necesarias para albergar a los jugadores de la *CE*. En caso que la función *SM.distribuir_Jugadores(CZ,CE)* falle, entonces el *SM*, acorde con el Algoritmo 4.4 creará una nueva Zona con todos los jugadores de la *CE* mediante la función *SM.crearZona(CE)*. La función *SM.crearZona(CE)*, detallada en el Algoritmo 4.6, ejecuta la función de *SM.latenciaMínima()* con el fin de hallar los mejores *SZ* y *SRZ* referente a los valores de latencia respecto al *SM*. De forma adicional, esta función comprueba que el *SRZ* seleccionado sea capaz de servir el mismo número de *SJ*s que el *SZ*, con el fin de evitar problemas relacionados con la tolerancia a fallos, en caso de que *SRZ* tenga que reemplazar al *SZ*. En caso de que el *SZ* sea capaz de gestionar un mayor número de *SJ*s, entonces mediante la llamada a la función *cambiar* se intercambiarán los roles entre *SZ* y *SRZ*

Algoritmo 4.6: Algoritmo de creación de una Zona del sistema OnDeGaS (alternativa heterogénea).

Input: CE
Output: Z_i

```
1  SM.crearZona(CE):
2  begin
3      SZ=SM.latenciaMínima(CE);
4      SRZ=SM.latenciaMínima(CE-{SZ}) ;
5      if (SZ.coresLibres()>SRZ.coresLibres()) then cambiar(SZ,SRZ);
6      SZ.aceptar(CE);
7      SRZ.aceptar(CE);
8      Z_i = {SZ ∪ SRZ};
9      return Z_i;
10 end
```

de forma permanente. Finalmente, todos los jugadores de la *CE* se enlazan con ambos servidores de la Zona *SZ* y *SRZ* (*SZ/SRZ.aceptar(CE)*), con el fin de mantener el estado del juego consistente entre el *SZ* y *SRZ*.

Finalmente, cada vez que una nueva Zona es creada, ésta se añade a la Cola de Zonas (*CZ*), con el fin de disponer de los recursos de cómputo ociosos en ellas, para distribuir nuevos jugadores de la *CE*, en los momentos de sobrecarga del área central.

2. **Estado de no sobrecarga.** Si el *SM* no está sobrecargado, en el Algoritmo 4.4 se evalúan los siguientes tres condicionales:

   - El primer condicional evalúa si el tiempo trascurrido desde la primera inserción de un jugador en la *CE* es mayor o igual que un valor predefinido β. En caso afirmativo todos los jugadores encolados en la *CE* serán aceptados por el *SM* (*SM.aceptar()*), con el fin de que puedan empezar a jugar en el área central. Tal y como sucede en el caso homogéneo, los primeros jugadores aceptados por el *SM* son los primeros jugadores encolados en la *CE*, siempre que el *SM* tenga espacio suficiente para hacerlo. En caso contrario, el sistema seguirá el mismo procedimiento introducido para el caso homogéneo.

- El segundo condicional da prioridad de entrada al *SM* a aquellos jugadores, distribuidos en zonas, ejecutándose en el área P2P. Para poder aceptar jugadores de esta área de juego se comprobará cuales son las zonas que han notificado al *SM* que sus partidas han finalizado, mediante el envío del mensaje de partida finalizada, tal y como sucede en el caso homogéneo. Sin embargo, este caso difiere del homogéneo, en el hecho de que pueden haber más jugadores a aceptar por el *SM* ya que más de una partida o *SJ* puede ser ejecutada por una misma Zona. Cabe destacar que el sistema aceptará los jugadores ubicados en la misma partida en bloque, abortando la operación en los casos en los que la aceptación completa no sea posible.
- Finalmente, tal y como sucede en el caso *homogéneo*, el *SM* aceptará nuevas peticiones de conexión de jugadores entrantes al sistema.

4.2.3. Detalles de implementación

A continuación, vamos a describir, aspectos adicionales relevantes de las funciones usadas por el sistema *OnDeGaS* en su versión homogénea y heterogénea, para incrementar el nivel de detalle de la exposición de la operación del sistema en su conjunto.

Algoritmo de búsqueda de *SZ* y *SRZ* basado en latencia

El mecanismo de búsqueda de *SZ* y *SRZ* (*SM.búsqueda(CE)*), introducido en el Algoritmo 4.2, está basado en la latencia que los jugadores tienen respecto al *SM*, ya que los juegos MMOFPS requieren de latencias muy bajas, tal y como hemos presentado en anteriores capítulos. De hecho, muchos trabajos de la literatura remarcan la importancia de la latencia en la calidad de servicio de los MMOFPS [CHHL05, GH04, KLXH04, RMO07]. Por lo tanto, este mecanismo basado en latencias comprueba la latencia que todos los jugadores de la *CE* tienen respecto al *SM*, escogiendo como *SZ* aquel jugador con latencia menor y siendo *SRZ*, el segundo jugador con latencia menor. Seleccionar como *SZ* al jugador con latencia más baja respecto del *SM* es una garantía para la mayoría

de jugadores de la Zona, de que el *SZ* tendrá un rendimiento similar al *SM*, por lo que se refiere a latencia. Más adelante se evaluará la incidencia del algoritmo de búsqueda de *SZ* y *SRZ* en la calidad de servicio del sistema.

Evaluación del estado de sobrecarga

El estado de sobrecarga del sistema se obtiene evaluando el número de jugadores concurrentes que el *SM* gestiona. Este parámetro ha sido escogido teniendo en cuenta el análisis del Capítulo 3, en el que se ha mostrado como el número de jugadores concurrentes está directamente relacionado con el consumo de CPU y de ancho de banda. Otro hecho, que reafirma nuestros análisis empíricos, son los que hemos extraído de las aportaciones de varios autores de la literatura en las que se señala como factor más determinante de la carga de los sistemas MMOG, el número de jugadores concurrentes [GH04, RBD04, YC06].

Tolerancia a fallos

La implementación de mecanismos de tolerancia a fallos se basa en el rol del *Servidor Replicado de la Zona* (*SRZ*). El *SRZ* tiene el rol de reemplazar al *SZ*, en caso que éste se desconecte del juego. Por esta razón, tanto el *SZ* como el resto de jugadores de la Zona envían el estado del juego al *SRZ*. De este modo, el *SRZ* tiene el estado del juego constantemente actualizado, preparado en cualquier momento para sustituir de forma transparente al *SZ* en caso de fallo. Se considerará que el *SZ* ha fallado, si después de un segundo, ni jugadores ni *SRZ* reciben notificación alguna por parte del *SZ*, momento a partir del cual el *SRZ* se convierte en el nuevo *SZ*. Se ha fijado un periodo de tiempo superior a las latencias medias de estos juegos, contemplando de este modo, posibles fallos de red entre otros problemas externos al *SZ*. Sin embargo, no se ha contemplado un intervalo de tiempo superior al segundo, puesto que al tratarse de juegos tipo MMOFPS en los que el nivel de interacción y actividad de los jugadores es muy elevado, ampliar este umbral perjudicaría seriamente la calidad de servicio ofrecida a los jugadores.

La sobrecarga que introduce en las comunicaciones de las zonas la inclusión del *SRZ* es despreciable, puesto que tal y como se ha demostrado en el capítulo

3, el ancho de banda excedente en una conexión ADSL convencional es muy superior al usado por este tipo de juegos.

Funcionalidad Cores Libres

La funcionalidad *coresLibres* usada en los Algoritmos 4.5 y 4.6 retorna el número de cores o núcleos libres que tienen el *SZ* o *SRZ*. En nuestro sistema se asume que cada núcleo libre ejecutará una sola partida del juego. De este modo, el máximo número de partidas que una Zona es capaz de gestionar es igual al número de cores libres que tiene el *SZ*, teniendo en cuenta que el *SZ* se reserva un núcleo para ejecutar su propio *SJ* en el que está involucrado como jugador, más allá de su papel como *SZ*. Estudios llevados a cabo por Ye y Cheng en [YC06] demuestran como un único procesador disponible es capaz de gestionar una partida tipo MMOFPS con calidad de servicio.

4.3. Resultados Experimentales

Con el fin de mostrar la eficiencia y correcto funcionamiento del sistema *OnDeGaS*, en sus versiones homogéneas y heterogéneas, se ha realizado un proceso de experimentación cuyos resultados se aportan en la presente sección. La idea principal de esta experimentación es mostrar como el sistema propuesto es capaz de escalar de forma dinámica en función de la demanda, al mismo tiempo que la latencia de todo el sistema es mantenida bajo los umbrales de calidad de servicio aceptados para este tipo de juegos.

La experimentación ha sido efectuada mediante simulación, usando las librerías *SimPy* [Wor02]. *SimPy* agrupa las librerías necesarias para desarrollar simulaciones de eventos discretos mediante la codificación con lenguaje Python. Las herramientas que *SimPy* pone a disposición de la comunidad científica han sido usadas en conjunción a un desarrollo e implementación de módulos propios, para la simulación de los nodos de la plataforma. Se han desarrollado cuatro roles distintos: Jugador (J_i), Servidor de Zona (SZ), Servidor Replicado de Zona (SRZ) y el Servidor Maestro (SM). Las librerías de *SimPy* permiten introducir todos los componentes temporales de una forma aleatoria, permitiendo simular, de este modo, el comportamiento real de los jugadores.

En las simulaciones hemos prefijado la condición de sobrecarga del sistema central en los 2.000 jugadores concurrentes. De modo que cuando el *SM* gestione una cantidad igual a 2.000 jugadores se considerará que todo el sistema está en una estado de sobrecarga. Se ha establecido este valor teniendo en cuenta que según [KLXH04], los servidores actuales son capaces de gestionar de forma concurrente entre 2.000 y 6.000 jugadores. Sin embargo, para las pruebas que vamos a realizar, en las que queremos demostrar que el sistema *OnDeGaS* es escalable, el límite fijado no influirá para nada en los resultados obtenidos.

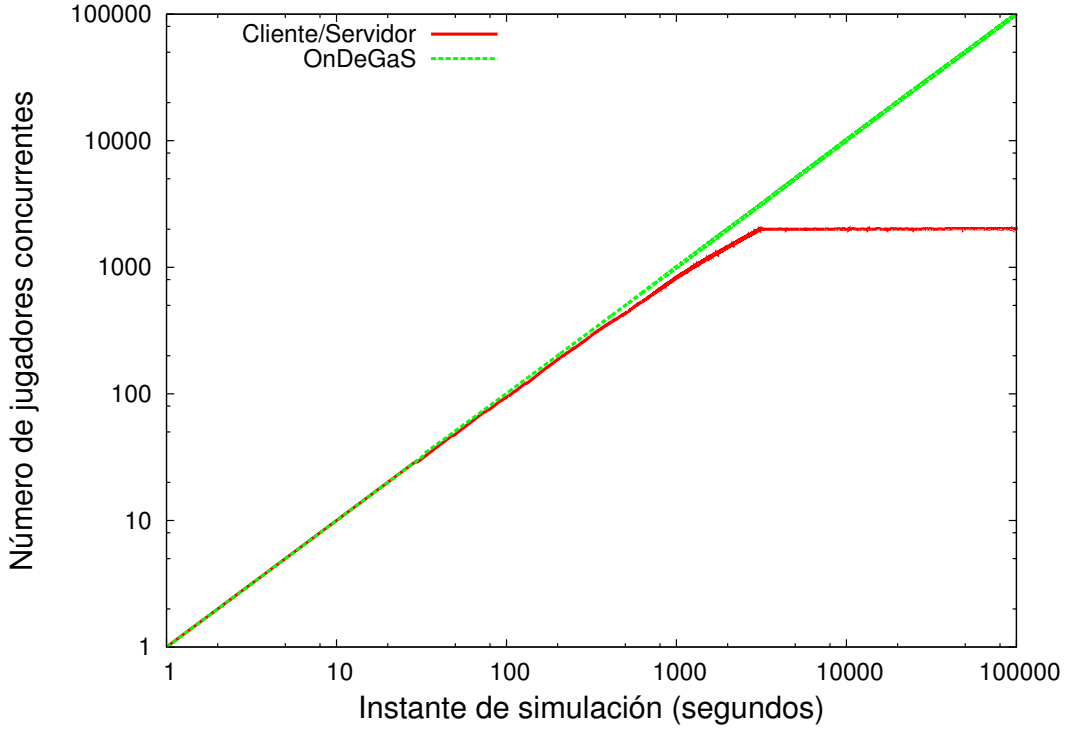


Figura 4.4: Comparativa entre los jugadores concurrentes gestionados por una propuesta Cliente-Servidor y *OnDeGaS*.

Cada una de las simulaciones efectuadas ha representado la conexión secuencial de 100.000 jugadores al *SM*. La frecuencia de conexión de jugadores ha sido establecida en 1 jugador/segundo, con el fin de someter al *SM* a una situación constante de estrés y evaluar, de este modo, que el área distribuida se expande añadiendo nuevas zonas, bajo la demanda de los jugadores. La Figura

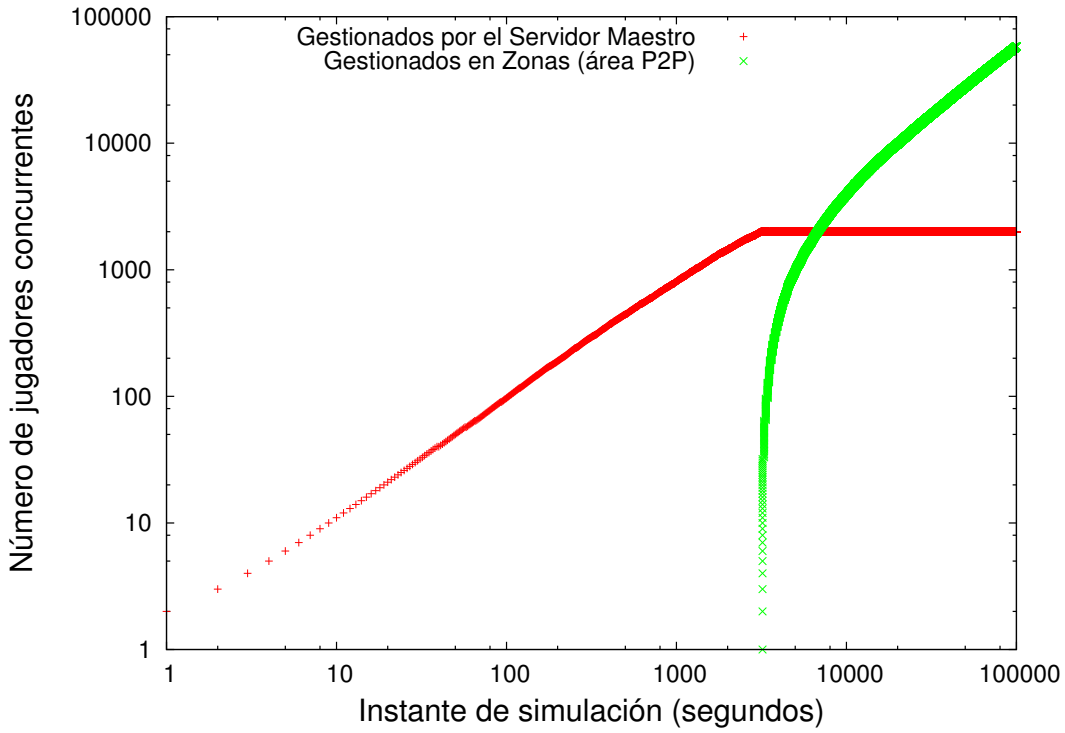


Figura 4.5: Comparativa entre los jugadores concurrentes del área central versus el área distribuida del sistema *OnDeGaS*.

4.4 representa, haciendo uso de una escala logarítmica, el volumen de jugadores concurrentes que son capaces de gestionar dos arquitecturas distintas: (a) una alternativa basada únicamente en una área central gestionada por un servidor maestro (o piscina de servidores), representada por la línea roja continua y (b), el sistema propuesto *OnDeGaS* representado por una línea discontinua de color verde. Se muestra como la solución Cliente-Servidor no es capaz de aceptar nuevos jugadores cuando ha hecho uso de todos sus recursos de cómputo, es decir, cuando éste ha llegado a una situación de sobrecarga; establecido en la simulación en 2.000 jugadores concurrentes. Por contra, cuando esta situación sucede en el sistema *OnDeGaS*, éste es capaz de escalar a medida que nuevas conexiones de jugadores se llevan a cabo, gracias a la explotación del área distribuida, gestionando entre el área central y la distribuida todo el volumen de jugadores que pretenden entrar en el sistema. Este comportamiento se refleja en la Figura 4.5, donde se diferencian los jugadores gestionados por el área

Central y por el área Distribuida del sistema *OnDeGaS*; reflejando los beneficios que aporta nuestra propuesta desde el punto de vista de la escalabilidad del sistema.

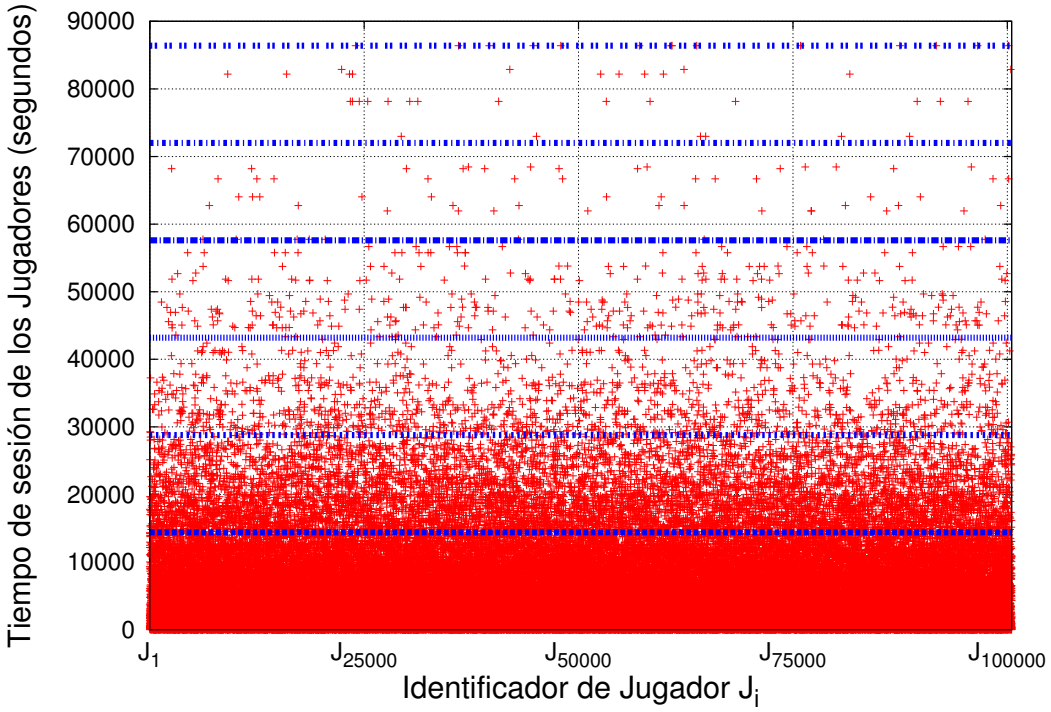


Figura 4.6: Tiempo de conexión de los 100.000 jugadores de la simulación representados mediante función de distribución *Weibull*.

Otro punto importante de los procesos de simulación es cómo hemos representado los valores de latencia en todos y cada uno de los jugadores, partiendo de la base que el punto de referencia es el *SM* del juego. Con objeto de simular los valores de latencia se ha delimitado un espacio Euclidiano de dos dimensiones con los siguientes rangos: $(x = [-110, +110],\ y = [-110, 110])$. Esta metodología está basada en las coordenadas relativas, tal y como se comenta en [EZ01].

Otro de los puntos vitales de la simulación es determinar el tiempo de sesión de los jugadores, es decir, determinar el tiempo de conexión o tiempo que permanecerán jugando. Para simular este tiempo de permanencia en el sistema hemos usado la función de distribución *Weibull* escalada desde los 0 segundos

hasta las 24 horas. En la Figura 4.6 se muestra la representación gráfica del tiempo de permanencia del sistema de todos los jugadores de la simulación. Cada uno de los puntos simboliza la permanencia de un único jugador (si extrapolamos hasta el eje de las abscisas), mientras que cada una de las líneas horizontales, punteadas de color azul, delimitan franjas de cuatro horas. Cabe destacar que la mayoría de jugadores tienen un tiempo de permanencia en el sistema bajo, ya que están situados en la primera franja de 4 horas. Otra franja poblada, pero en una proporción más baja, es la que representa el intervalo de 4 a 8 horas. El resto de intervalos son despreciables debido a la poca representación que tienen.

Otro aspecto a destacar es la duración temporal media de las partidas o Servicios de Juego (SJ) del área distribuida. Este valor se ha simulado mediante una función de distribución normal con una mediana de 900 segundos, la cual es una medida estándar para los juegos tipo MMOFPS según la literatura [BAS04] y una desviación estándar de 160 segundos, que introduce fluctuación en la simulación.

Los parámetros comentados son comunes para ambas alternativas: homogénea y heterogénea. Sin embargo, para configurar el sistema heterogéneo tenemos que considerar cuántos cores tiene el ordenador de cada uno de los jugadores del sistema. Tomando como referencia el número de cores que hoy en día se utilizan en ordenadores domésticos, hemos considerado que cada jugador puede tener 2, 4 o 8 cores, siendo ω_2, ω_4 y ω_8 los porcentajes de jugadores con este número de cores existentes en el sistema. Hemos definido como $\omega_{max} = max(\omega_2, \omega_4, \omega_8)$ al máximo porcentaje de todos, siendo ω_i y ω_j los otros dos porcentajes que sumados a ω_{max} nos permite obtener el total (100 %). Una vez definidas estas variables, la heterogeneidad del sistema, indicada como (het_degree), la podemos definir acorde a la Ecuación (4.1).

$$het\_degree = 1 - \frac{\frac{(\omega_{max} - \omega i)}{\omega_{max}} + \frac{(\omega_{max} - \omega j)}{\omega_{max}}}{2} \tag{4.1}$$

Los valores de het_degree se extienden entre el 0 y el 1, siendo 0 un sistema homogéneo y 1 un sistema en el que $\omega_2 = \omega_4 = \omega_8$, es decir, que existen el mismo número de jugadores con 2, 4 y 8 cores, en definitiva un sistema con la

máxima heterogeneidad.

Teniendo en cuenta las definiciones y funcionalidades comentadas previamente, en la siguiente subsección se evaluará el rendimiento de las políticas homogénea y heterogénea. Los casos de estudio para la alternativa homogénea serán: influencia de los parámetros α y β sobre el sistema, concretamente sobre la escalabilidad y calidad de servicio del mismo, analizando el número de zonas creadas, el valor de latencias medias en el área distribuida y por último, la tolerancia a fallos. Para el caso heterogéneo analizaremos la escalabilidad del sistema haciendo uso del área distribuida, la latencia media del sistema, el tiempo de espera en las colas y el impacto que tiene el grado de heterogeneidad del sistema sobre la escalabilidad del mismo.

4.3.1. Evaluación del rendimiento de la alternativa homogénea

En este apartado vamos a evaluar la capacidad del sistema *OnDeGaS* para escalar acorde con la demanda, teniendo en cuenta distintos valores de los parámetros α y β usados en los algoritmos que definen la operación del sistema.

Definimos la escalabilidad del sistema *OnDeGaS*, como la capacidad de gestionar jugadores bajo demanda mientras que la calidad de servicio de todo el sistema, fundamentalmente la latencia, se mantiene dentro de los intervalos aptos para los juegos MMOFPS.

La capacidad del sistema *OnDeGaS* para admitir jugadores según la demanda se ha medido mediante el análisis del número de zonas creadas en el área distribuida, así como por el número de jugadores por Zona. Ambos aspectos, dependen de una forma directa de los valores asignados a los parámetros α y β que han sido presentados en el Algoritmo 4.1. Para mostrar el impacto que tienen ambos parámetros, hemos realizado simulaciones en las que fijamos el valor del parámetro α, variando el de β y viceversa.

La parte superior de la Figura 4.7 muestra la influencia del parámetro α, cuando β (tiempo de espera máximo en la *CE*) tiene un valor fijo de 120 segundos. Por otro lado, la parte inferior de la misma imagen muestra la influencia del parámetro β, cuando α (número máximo de jugadores en la *CE*) tiene un

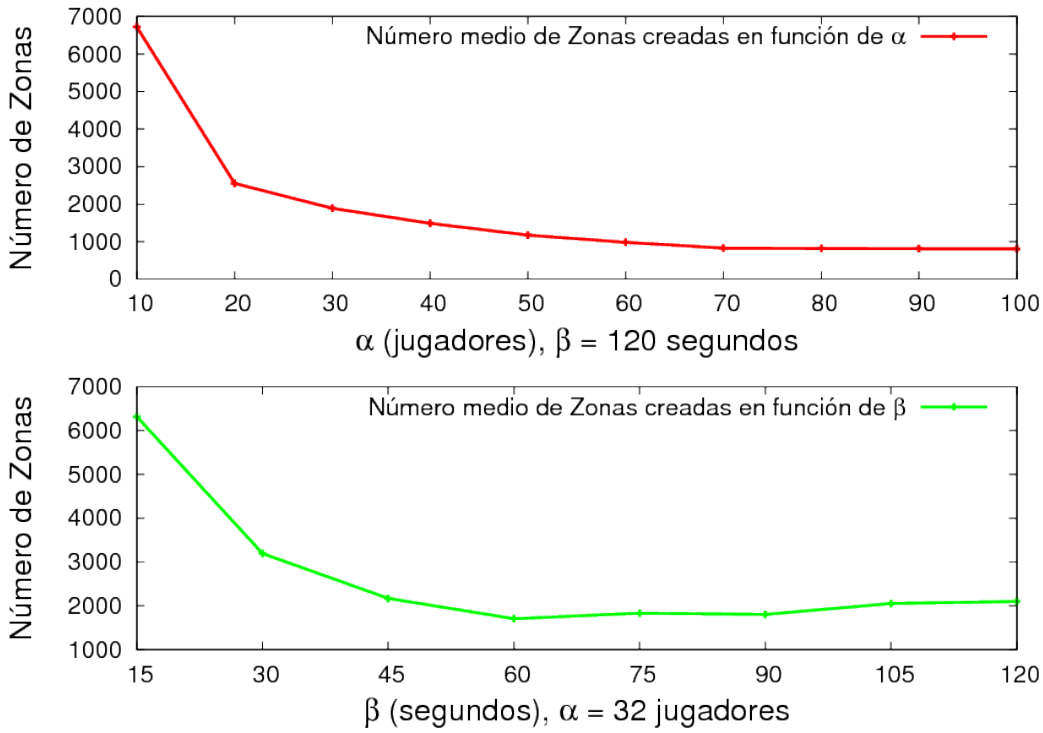


Figura 4.7: Número medio de zonas creadas en función de los valores de los parámetros α y β.

valor fijo de 32.

Los valores fijos usados para α y β han sido tomados teniendo en cuenta el comportamiento real que tienen los juegos comerciales tipo MMOFPS. Por un lado, un valor α de 32 jugadores por partida es un valor típicamente usado. Por otro lado, el tiempo de espera en la cola (β) es un valor que no encontraremos de forma explícita en los juegos comerciales tipo MMOFPS. Sin embargo, teniendo en cuenta que en casos de sobrecarga el tiempo de espera en los juegos comerciales sería indefinido, el tiempo de espera máximo fijado en 2 minutos (120 segundos) no supone un valor que afecte a la calidad de servicio del juego en una situación de sobrecarga.

Acorde con los resultados mostrados en la Figura 4.7, en general se observa que α tiene mayor influencia que β, puesto que mientras β impone el tiempo máximo de espera en la *CE*, α impone el máximo número de jugadores que podrán ser encolados en la *CE*, siendo éste el factor más influyente en el número

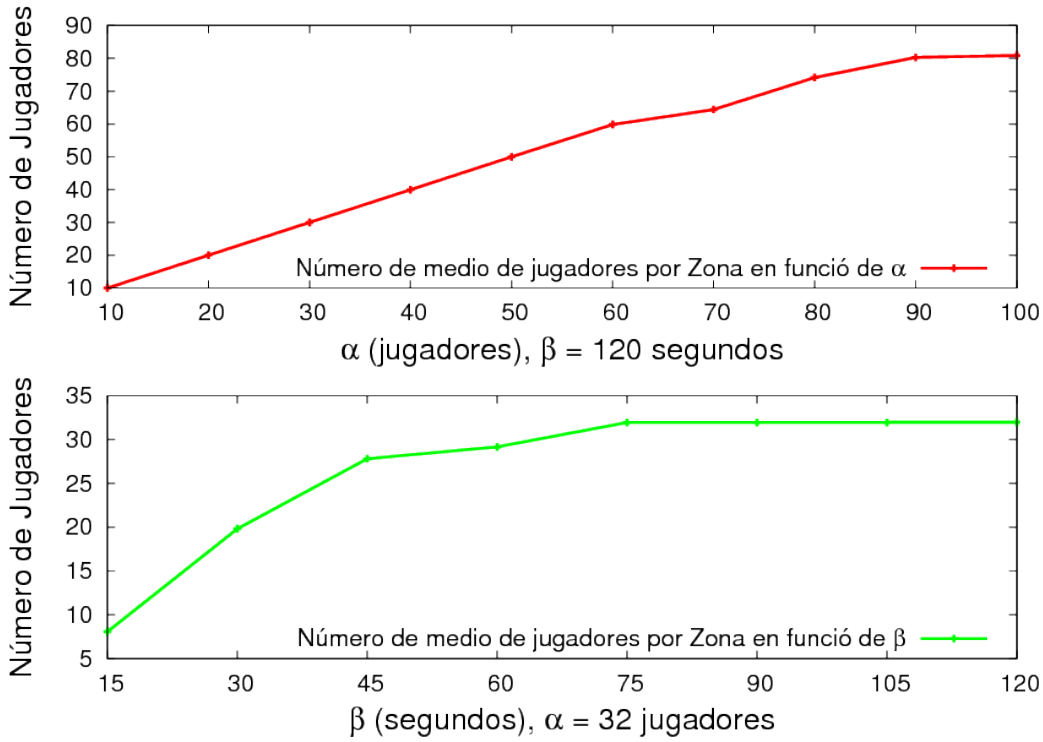


Figura 4.8: Número medio de jugadores por Zona en función de los valores de los parámetros α y β.

final de zonas creadas.

En la Figura 4.7 inferior, donde el valor de α ha sido fijado en 32 jugadores, se observa que el tiempo de espera máximo β tiene una influencia directa sobre el número de zonas creadas cuando éste toma valores bajos comprendidos entre 15 y 45 segundos. En cambio, cuando β va tomando valores superiores a los 45 segundos, su efecto es prácticamente despreciable, puesto que en estos casos son los valores que va tomando el parámetro α los que determinan el número de zonas que se crearán en el sistema *OnDeGaS*.

La Figura 4.8 muestra el número medio de jugadores que hay en cada una de las zonas creadas en función de los parámetros α y β. A medida que ambos parámetros se van incrementando, el número medio de jugadores por Zona se incrementará del mismo modo. Cuando los valores de α varían de los 10 hasta los 60 jugadores, la media de jugadores por Zona es exactamente el valor de α, lo cual denota la linealidad entre ambas variables. A partir de los

60 jugadores, la linealidad mostrada en el caso anterior se pierde, ya que el número medio de jugadores va creciendo de forma gradual, pero sin llegar a los valores de α. En cambio, cuando α se encuentra entre 90 y 100 jugadores existe un estancamiento de la tendencia observada para valores de α inferiores. Este comportamiento es debido al valor fijo de β igual a 120 segundos, ya que en caso de incrementar este valor, la tendencia de jugadores por Zona seguiría la tendencia lineal mostrada en las primeras series de la gráfica. Cuando α es el parámetro variable, el número medio de jugadores por Zona se incrementa a medida que lo hace el valor de β. Para intervalos de β comprendidos entre los 15 y los 45 segundos, existe un crecimiento progresivo, aunque sin llegar al valor de 32 jugadores, valor fijado para el parámetro α en este caso. Cuando β toma valores superiores a 75 segundos, el número medio de jugadores por Zona se acerca al valor establecido para α, es decir, a 32 jugadores.

Tal y como se muestra en las Figura 4.7 y 4.8 existe una relación inversamente proporcional entre los valores de α y β y el número de zonas creadas, así como el promedio de jugadores que éstas tienen, puesto que a medida que los valores de α y β se van incrementando, el número de zonas creadas decrece.

En conclusión, la máxima influencia que los parámetros α y β tienen sobre el número de zonas creadas y la media de jugadores por Zona sucede cuando ambos parámetros se encuentran en un intervalo de 10 a 60. A partir de estos valores, no es el parámetro variable, sino el fijo el que tiene una incidencia decisiva en el número de zonas y en la media de jugadores en ellas. Teniendo en cuenta que la mayoría de MMOFPS contemplan partidas de 32 jugadores, los resultados demuestran que el valor idóneo para α será justamente 32 jugadores. Extrapolando el valor adecuado para α, con los resultados del sistema *OnDeGaS*, podemos establecer el valor de β a los 120 segundos, como valor apropiado que permite obtener un buen rendimiento del sistema.

A continuación vamos a evaluar la calidad de servicio del sistema, representada por la habilidad en mantener la latencia dentro de cada Zona distribuida bajo unos umbrales aptos para los juegos tipo MMOFPS, así como la de asegurar un mínimo de tolerancia a fallos en el área distribuida.

Para estudiar la latencia media del sistema se han variado, de nuevo, los valores de los parámetros de α y β. La Figura 4.9 muestra la latencia media

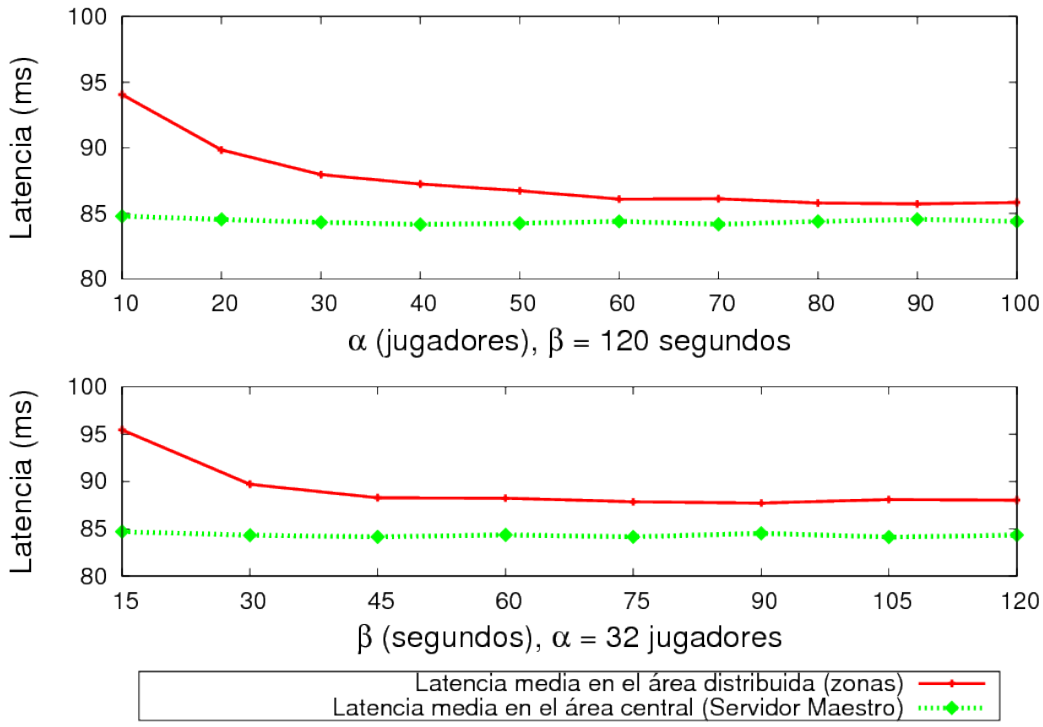


Figura 4.9: Latencia media del área central y del área distribuida en función de los valores de los parámetros α y β.

de las zonas (línea continua), así como la latencia existente en el área central gestionada por el *SM* (línea punteada).

Se puede observar en ambas gráficas, que a medida que el parámetro α se incrementa, la latencia media de las zonas disminuye, aproximándose al valor medio de la latencia existente en el área central. Este comportamiento se debe a la influencia que la función de búsqueda tiene en el sistema *OnDeGaS*, ya que si incrementamos el número de jugadores en la *CE*, la probabilidad de encontrar un *SZ* y un *SRZ* con latencia más baja es mayor, puesto que se incrementa el dominio de búsqueda. Tal y como hemos comentado previamente, existe una relación lineal entre el número medio de jugadores por Zona con los valores que toman los parámetros α y β. De este modo, a medida que ambos valores se incrementan, el valor medio de latencia de las zonas disminuye. Por tanto, para obtener valores medios de latencia próximos a los valores existentes en el área central, los valores α y β tienen que ser lo más grandes posible. Sin embargo,

cabe destacar que ciertamente, no es necesario ir más allá de los 32 jugadores o 120 segundos para α y β para obtener un rendimiento satisfactorio, por lo que respecta a valores medios de latencia para los juegos tipo MMOFPS.

Con el objetivo de demostrar que la media de latencia de las zonas mostradas en la Figura 4.9 no se aleja, en valores discretos, de los valores medios de las zonas, en la Figura 4.10 reflejamos la latencia media (representada por puntos rojos) de 500 zonas distintas seleccionadas al azar, con una escala comprendida entre el máximo y mínimo valor (≈ 70 y ≈ 110 milisegundos). Para facilitar la interpretación de la figura se han añadido tres líneas horizontales en la gráfica. Las líneas superior e inferior delimitan el intervalo de latencias medias con mayor frecuencia, comprendido entre los 85 y los 95 milisegundos, mientras que la línea central define el valor de latencia medio para todas las zonas. Teniendo en cuenta los valores mostrados en la Figura 4.10 podemos remarcar que la desviación estándar no es muy significativa, asegurando de este modo que la función de búsqueda implementada en los algoritmos del sistema permite la eliminación de valores extremos, por lo que respecta a los valores de latencia media de las zonas.

Teniendo en cuenta los resultados mostrados referentes a la escalabilidad del sistema, se ha demostrado que *OnDeGaS* es capaz de escalar bajo demanda, creando nuevas zonas cuando el sistema entra en una situación de sobrecarga. Adicionalmente, se ha mostrado que la escalabilidad lograda por *OnDeGaS* no afecta negativamente a la latencia media de todo el sistema, incluso en ciertas zonas, la latencia media es inferior a la que se experimenta en el área central. Cabe destacar que en todos los casos, la latencia asegurada es muy inferior al umbral mínimo de latencia que debe proporcionarse en los juegos MMOFPS.

Finalmente, la Figura 4.11 muestra los beneficios aportados en la tolerancia a fallos debido a la inclusión del rol de *SRZ* en las zonas, o dicho de otro modo, el margen temporal que el *SRZ* proporciona a las zonas en función de los valores de los parámetros α y β en aquellas Zonas en las que el *SZ* se ha desconectado antes que el *SRZ*. La línea punteada de color verde representa el tiempo de servicio medio, es decir, el tiempo que transcurre desde la creación de la Zona hasta que el *SZ* falla en las zonas que no disponen de un *SRZ*; mientras que la línea continua de color rojo representa lo mismo cuando la Zona contiene

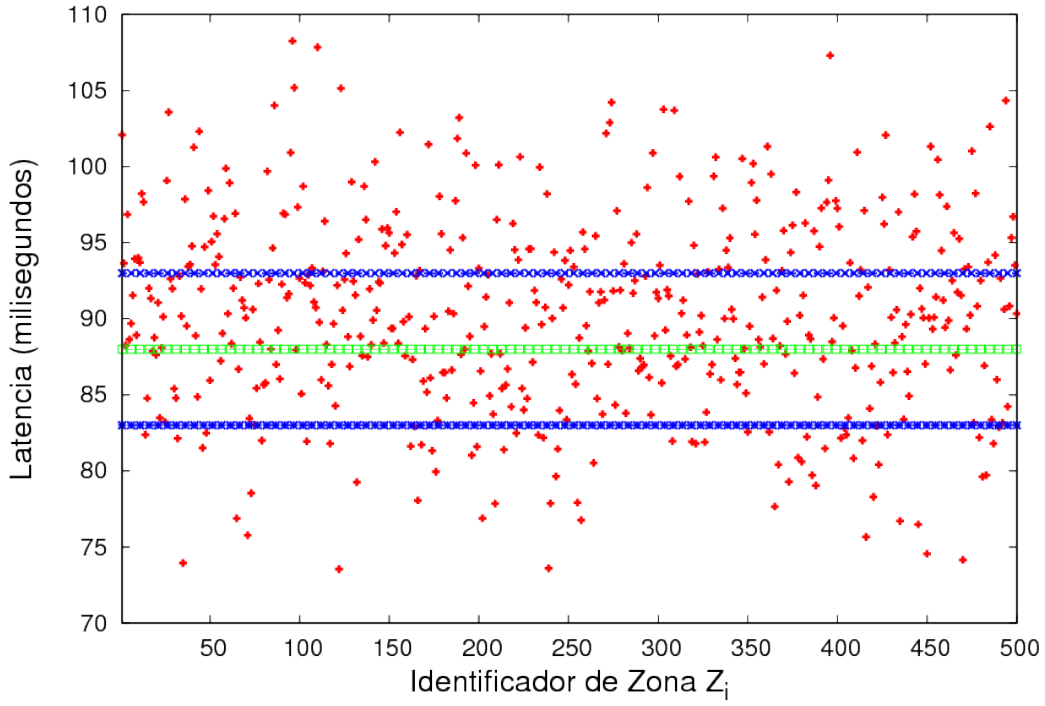


Figura 4.10: Valores medios de latencia de 500 zonas seleccionadas al azar.

un *SRZ*. Se puede ver que en este segundo caso, los valores que toman los parámetros α y β no influyen sobre el rendimiento global del mecanismo de tolerancia a fallos, ya que éste vendrá determinado por la duración temporal de juego que tienen los jugadores, que tal y como hemos comentado en el inicio de la Sección 4.3 está determinado por la función de distribución *Weibull* usada para representar este parámetro de la simulación (ver Figura 4.6). Por esta razón, el tiempo medio de disponibilidad del *SRZ*, así como el tiempo extra que aporta el *SRZ*, fluctúa entre muchos valores. Sin embargo, a pesar de la variación y alta fluctuación entre valores, la diferencia entre el tiempo de conexión del *SRZ* y el aporte extra generado por el *SRZ* es prácticamente constante. De media, para los casos en que α y β tienen unos valores de 32 jugadores y 120 segundos respectivamente, la inclusión de un *SRZ* en las zonas aporta un tiempo extra de disponibilidad de 17 minutos con una desviación estándar de $2minutos$, lo que representa un incremento notable de la esperanza del tiempo total de disponibilidad de los juegos, permitiendo de este modo,

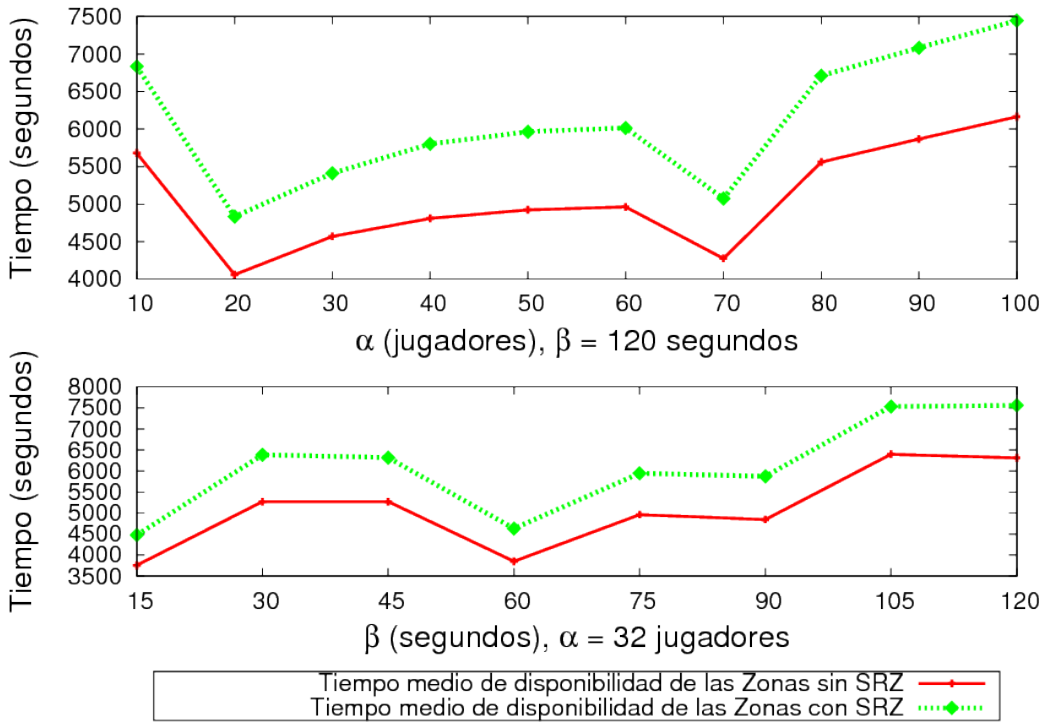


Figura 4.11: Estudio del impacto generado por la política de tolerancia a fallos.

extender el tiempo de ocio de los jugadores ubicados en las zonas.

4.3.2. Evaluación del rendimiento de la alternativa heterogénea

En este apartado vamos a evaluar la capacidad del sistema *OnDeGaS* para escalar acorde con la demanda, considerando la heterogeneidad de los nodos del sistema. En este caso, partiendo del conocimiento que la alternativa homogénea nos ha aportado, los parámetros α y β se han fijado en 32 jugadores y 120 segundos, respectivamente. Al igual que con la alternativa homogénea vamos a identificar la capacidad de escalar teniendo en cuenta el número de zonas que se crean en el área distribuida, así como la media de jugadores que existe en cada una de ellas. Finalmente, evaluaremos bajo distintas condiciones de heterogeneidad, la capacidad del sistema de explotar el área distribuida.

La Tabla 4.1 muestra los valores medios (X), así como la desviación estándar

(S), del número de zonas creadas, la latencia media en estas zonas y el tiempo de espera de los jugadores en la *CE* hasta que han sido distribuidos, bajo dos condiciones de heterogeneidad: (a) *h_d=0* (*het_degree=0*), cuando todos los jugadores disponen de dos cores ($\omega_2 = 100\,\%, \omega_4 = \omega_8 = 0\,\%$) y, (b) *h_d=1* (*het_degree=1*), cuando $\omega_2 = \omega_4 = \omega_8$. En cada caso hemos evaluado 1.000 simulaciones. Tal y como se observa en la columna del número de zonas creadas (segunda columna) de la Tabla 4.1, la distribución realizada por el sistema *OnDeGaS* es capaz de explotar los cores adicionales considerados en la alternativa heterogénea, ya que es capaz de crear un menor número de zonas cuando cada zona creada dispone de más cores (h_d=1), insertando en cada una de ellas un mayor número de *SJs*. En la práctica, gestionar el mismo volumen de jugadores en un menor número de zonas supone una disminución notable de la sobrecarga de comunicaciones que se genera debido a la gestión del conjunto de *SZ*s y *SRZ*s del área distribuida.

	Zonas (Núm.)		**Latencia (ms)**		**CE Tiempo (s)**	
	$h\_d = 0$	$h\_d = 1$	$h\_d = 0$	$h\_d = 1$	$h\_d = 0$	$h\_d = 1$
X	888,38	395,84	87,43	88,79	30,16	24,40
S	193,37	76,92	0,293	3,955	4,70	6,30

Tabla 4.1: Análisis del rendimiento del sistema *OnDeGaS* (alternativa heterogénea).

Referente a la calidad de servicio, la Tabla 4.1 muestra una latencia media similar en ambos casos (inferior a los dos milisegundos de diferencia). Sin embargo, la desviación estándar difiere ligeramente. Esto es debido a que al explotar los recursos ociosos de zonas existentes, los jugadores de la *CE* son enlazados con un *SZ predefinido*. Por tanto, el algoritmo de búsqueda basado en latencia no tiene en cuenta las latencias de los jugadores de la *CE*, hecho que provoca que algunas asignaciones mejoren el rendimiento de la latencia o por contra empeoren, hecho que se compensa tal y como muestra la latencia media del sistema, pero que afecta a la desviación estándar. Destacar que en ambos casos, los valores de latencia se mantienen por debajo del intervalo adecuado de latencias estipulado para juegos tipo MMOFPS en un máximo de 180 milisegundos.

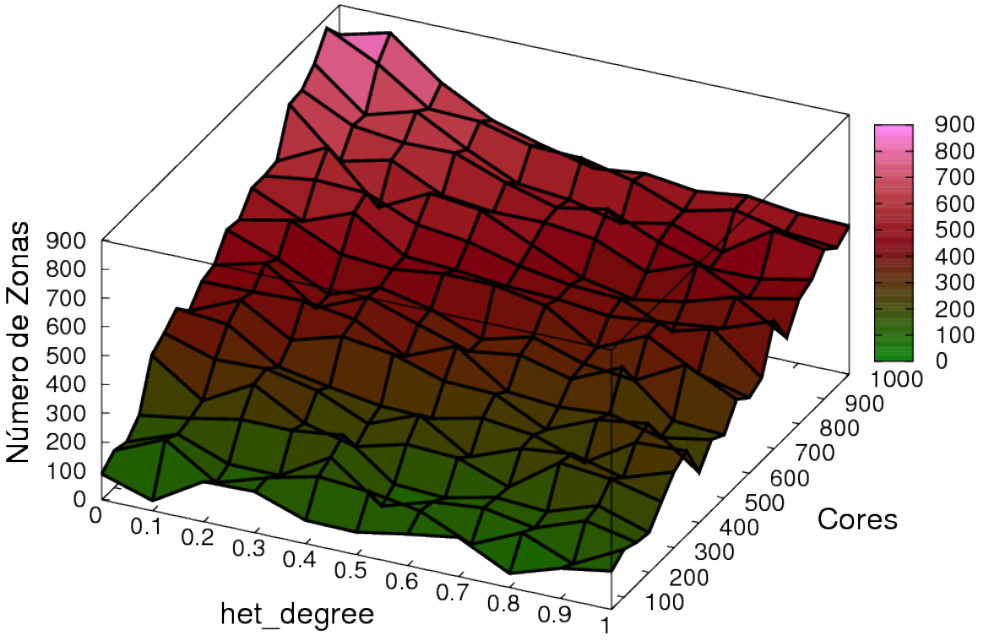


Figura 4.12: Número de zonas creadas en función del *het_degree* cuando $((\omega_{max} = \omega_2) \geq \omega_4 \geq \omega_8)$

Respecto a los tiempos de espera que pasan los jugadores encolados en la *CE*, desde su inserción hasta que son distribuidos o aceptados en alguna zona, los resultados revelan un impacto significativo en las medianas, dependiendo del número de cores disponibles en el sistema. Cada vez que se crea una nueva Zona, el algoritmo del sistema realiza la búsqueda de los *SZ* y *SRZ* apropiados; situación que se produce ≈ 889 veces en el caso $h\_d = 0$ y de ≈ 396 veces para $h\_d = 1$. Esta diferencia de 493 en el número de nuevas zonas, a priori debería suponer que el tiempo de espera para los jugadores encolados en la *CE* para $h\_d = 0$ debería ser el doble aproximadamente que para $h\_d = 1$; no obstante, los tiempos de espera son de 30,16 y 24,40, respectivamente. Por tanto, la diferencia de tiempos no es mayor porque a pesar que en el caso de $h\_d = 1$ se crean un menor número de zonas, existe un mayor número de cores, hecho que provoca que el algoritmo de distribución tenga que comprobar en que zonas ya creadas puede distribuir los jugadores encolados en la *CE*, invirtiendo en esta

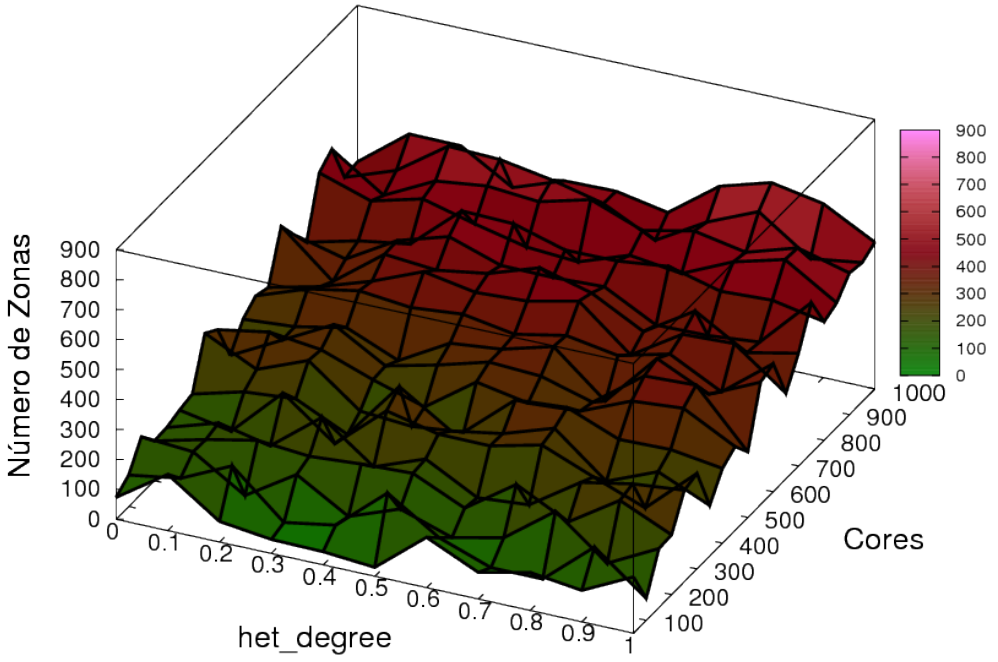


Figura 4.13: Número de zonas creadas en función del *het_degree* cuando ($\omega_2 \leq (\omega_{max} = \omega_4) \geq \omega_8$)

tarea unos segundos ($\approx 10s$).

En la Tabla 4.1 no se ha representado la diferencia en los mecanismos de tolerancia a fallos con respecto a la heterogeneidad, ya que tal y como hemos visto en la propuesta homogénea, es un rendimiento totalmente dependiente de cómo se modela el tiempo de sesión de los jugadores, y por tanto, totalmente independiente del grado de heterogeneidad.

Para expandir el estudio de la influencia del grado de heterogeneidad del sistema, hemos evaluado los efectos que ésta tiene en el número de zonas creadas. Las Figura 4.12, 4.13 y 4.14 reflejan las tendencias que comentamos en función del valor de *het_degree*, variando el número total de cores en el sistema, así como los porcentajes ω_2, ω_4 y ω_8. La Figura 4.12 corresponde a un sistema en el que la mayoría de máquinas de los jugadores tienen 2 cores ($\omega_{max} = \omega_2$) y los porcentajes se distribuyen de este modo: $\omega_2 \geq \omega_4 \geq \omega_8$. Del mismo modo, la Figura 4.13, muestra los resultados del sistema cuando la

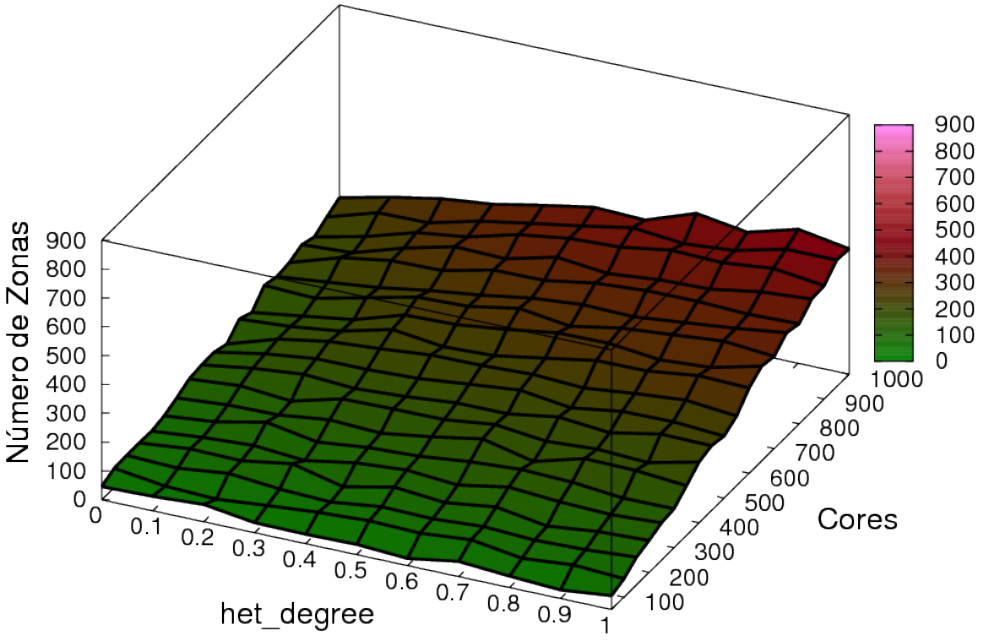


Figura 4.14: Número de zonas creadas en función del *het_degree* cuando ($\omega_2 \leq \omega_4 \leq (\omega_{max} = \omega_8)$)

relación es $\omega_2 \leq \omega_4 \geq \omega_8$, siendo $\omega_{max} = \omega_4$. Por último, la Figura 4.14 refleja los resultados del sistema cuando $\omega_{max} = \omega_8$ y el resto de porcentajes siguen la relación $\omega_2 \leq \omega_4 \leq \omega_8$. Tal y como se observa en dichas gráficas, el número total de cores y su distribución entre los jugadores determinará el número total de zonas a crear. Cuando $\omega_{max} = \omega_2$ (ver Figura 4.12), el número total de zonas creadas es, en la mayoría de las series del gráfico, mayor que cuando $\omega_{max} = \omega_8$ (ver Figura 4.14). De este modo, el sistema crea nuevas zonas sólo cuando los cores existentes están ocupados. Asimismo, también se puede percibir en los gráficos documentados, como a medida que se incrementan el número de cores del sistema, el número de zonas creadas también lo hace, hecho provocado por el incremento en el número de jugadores en el sistema y por tanto de la sobrecarga en el área central del mismo. La evolución en el grado de heterogeneidad (*het_degree*) muestra 3 comportamientos diferentes dependiendo de la gráfica. Cuando $\omega_{max} = \omega_2$ (ver Figura 4.12), el número

total de zonas creadas se incrementa a medida que *het_degree* disminuye, ya que predominan los jugadores con sólo 2 cores. Sin embargo, esta tendencia es totalmente opuesta en el caso de $\omega_{max} = \omega_8$ (ver Figura 4.14), ya que la mayoría de jugadores tienen 8 cores. En el caso $\omega_{max} = \omega_4$ (ver Figura 4.13), el rendimiento obtenido es justamente el intermedio, obteniendo una tendencia global estable a medida que varían los valores de *het_degree*.

A partir de los resultados experimentales mostrados podemos concluir que los mecanismos de distribución de carga del sistema *OnDeGaS* son capaces de gestionar los distintos niveles de heterogeneidad apropiadamente, gracias a la explotación de los recursos de cómputo del sistema.

Cabe destacar el hecho que, ambas alternativas (homogénea y heterogénea), ofrecen a todos los jugadores la posibilidad de jugar, ya bien sea en el área central o en la distribuida. Por último, remarcar que el tiempo necesario para realizar la transición de la *CE* a la Zona del área distribuida, como de ésta al área central gestionada por el *SM*, es del orden de segundos, siendo éste un tiempo de espera que se considera aceptable para los jugadores en el entorno de los juegos MMOFPS.

Capítulo 5

Aplicación del sistema híbrido CS/P2P a juegos MMORPG

En este capítulo se analiza cómo la propuesta de la arquitectura *OnDeGaS*, que combina un modelo centralizado Cliente-Servidor con uno distribuido P2P, puede ser también usada eficientemente para la ejecución de juegos tipo MMORPG.

Para ello se analizarán las características propias de los juegos MMORPG y sus diferencias con respecto a los MMOFPS, vistas en el capítulo anterior, en cuanto a su comportamiento y requisitos de ejecución se refiere.

A continuación se propondrán nuevos mecanismos concretos que permitan explotar el sistema *OnDeGaS* para juegos MMORPG. Finalmente se evaluarán mediante simulación la eficiencia y escalabilidad de nuestras propuestas.

5.1. Características de los juegos MMORPG

En el caso de los MMORPG, el juego principal (o historia del juego) se ejecuta en un mundo virtual común para todos los jugadores, que a diferencia de los juegos MMOFPS, es permanente, es decir, su ejecución es constante. En este caso, los jugadores no se agrupan en partidas independientes e impermeables, sino que todos son distribuidos a lo largo del mapa. Por tanto, el servidor del juego debe gestionar un gran volumen de jugadores comunicándose en un mismo escenario. Sin embargo, este tipo de juegos, aparte del juego que se

ejecuta en el mundo virtual principal, también tienen una serie de modos de juego que se ejecutan al margen del juego principal, a los cuales vamos a hacer referencia como *Juegos Auxiliares*[1]. En los juegos auxiliares se crean conjuntos de jugadores del orden de algunas decenas, para ejecutar instancias de un tipo de juego auxiliar, que son totalmente independientes entre el resto de juegos auxiliares y también de la ejecución del mundo virtual principal. Este modo de estructurar los juegos tipo MMORPG está representado gráficamente en la Figura 5.1, donde se ilustra el mundo virtual principal con todos los jugadores y un conjunto de juegos auxiliares, formados por unos cuantos jugadores del mundo virtual principal, pero que ejecutan una partida de forma independiente. Estos juegos auxiliares son de carácter temporal, es decir, los objetivos de estas instancias se realizan a corto plazo y los jugadores una vez logrado los objetivos de la instancia, retornan al mundo virtual principal. Vemos, por tanto, que los juegos auxiliares de los MMORPG guardan una gran similitud con las partidas de los juegos MMOFPS, puesto que constituyen partes independientes de duración limitada, facilitando de este modo una gestión aislada de los mismos.

Así pues, dadas las características de los juegos MMORPG, nuestra propuesta se basa en realizar su gestión combinando dos enfoques diferenciados:

1. Gestión del mundo virtual principal de forma centralizada.

2. Gestión de los juegos auxiliares de forma distribuida.

Para ello se propone el uso de la arquitectura *OnDeGaS*, propuesta en esta tesis, y el uso de algoritmos de gestión específicos para los juegos MMORPG.

[1]En el *World of Warcraft* son conocidos como las mazmorras, las arenas o los campos de batalla.

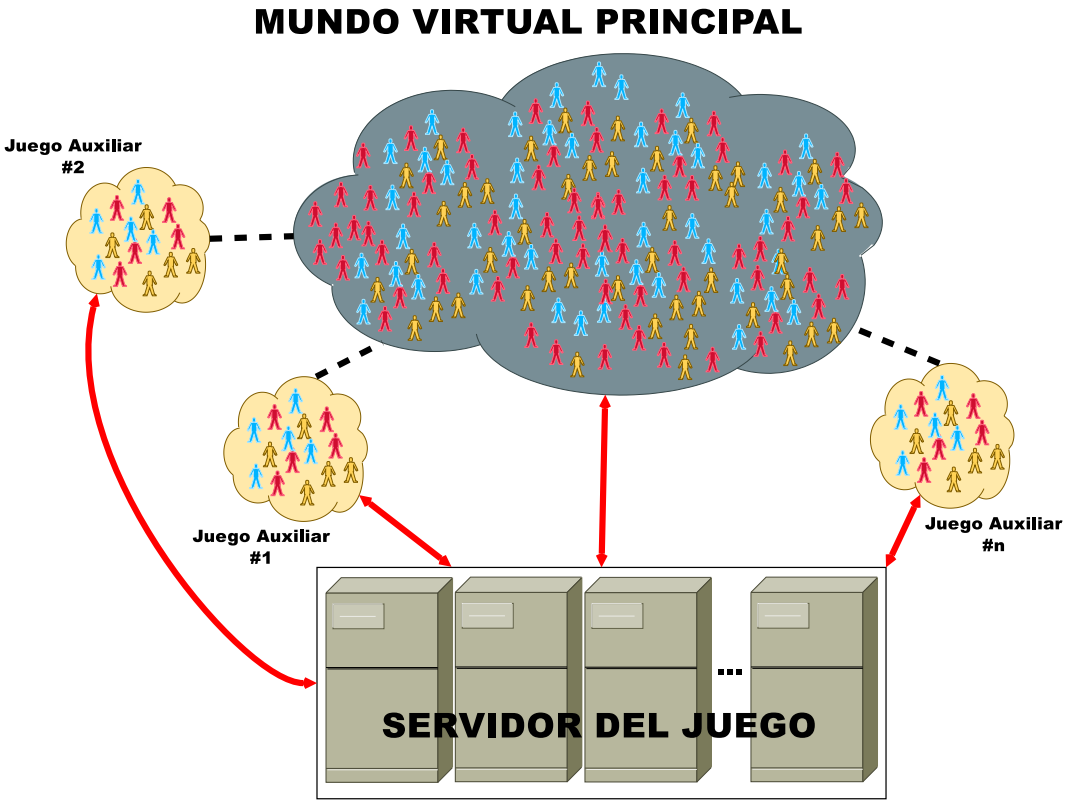


Figura 5.1: Modelo de gestión de los juegos tipo MMORPG.

La *Descripción del sistema*, la *Distribución de cómputo en el sistema* OnDeGaS *para MMORPG* y la *Evaluación de las propuestas* están bajo estados de revisión científica de una publicación de revista, por consiguiente, para encontrar estos capítulos se deberá esperar a su publicación en:

- **En Revisión I. Barri, C. Roig. F. Giné.**
- ***Distributing Game Instances in a Hybrid Client-Server/P2P System to Support MMORPG Playability.***
- **Journal of Parallel and Distributed Computing.**

Capítulo 6

Conclusiones y líneas abiertas

En este capítulo se describen las conclusiones obtenidas en el desarrollo del presente trabajo, junto con las principales aportaciones y publicaciones realizadas. Asimismo se explican las líneas abiertas que serán parte del desarrollo futuro de la investigación, y que alguno de sus puntos ya se está llevando a cabo.

6.1. Conclusiones

En los últimos años los videojuegos han estado en una evolución constante. Actualmente las aplicaciones tipo MMOG (*Massively Multiplayer Online Games*) se han apoderado del mercado de los videojuegos, convirtiéndose en una de las alternativas de ocio electrónico con más éxito. Siendo el paradigma Cliente-Servidor, el más utilizado para gestionar el cómputo de los MMOG, en esta tesis se ha propuesto un modelo híbrido, entre Cliente-Servidor y P2P (Peer-to-Peer), para solucionar los problemas de escalabilidad típicamente producidos en los entornos Cliente-Servidor. En el modelo híbrido, el servidor central sigue teniendo el control del juego, gestionando un número limitado de jugadores, mientras que la zona P2P constituye una alternativa emergente de bajo coste que proporciona servicio a aquellos jugadores que no pueden ser gestionados por el servidor debido a problemas de sobrecarga en el sistema central. De este modo, el modelo híbrido propuesto aprovechará la infrautilización de los ordenadores de los jugadores, integrándolos en una estructura de

juegos distribuida autogestionada gracias al uso de sus ciclos de procesador y otros recursos de cómputo ociosos a través de Internet.

Acorde con el objetivo descrito, el trabajo de tesis se ha estructurado en diferentes fases temporales. En primer lugar se realizó una investigación de los trabajos existentes en la literatura científica sobre la ejecución eficiente de aplicaciones MMOG tanto en entornos de cómputo distribuidos como centralizados. En esta fase se identificaron bajo qué criterios se diseñan los algoritmos o metodologías de balanceo de carga. De este modo, se comentó y clasificó la mayor parte de la información extraída de la literatura, resaltando las principales diferencias con respecto a nuestra propuesta. Una vez estudiado ésta, se procedió a la realización de un análisis empírico acerca de los requerimientos de recursos de cómputo que las categorias de juegos MMOFPS y MMORPG precisan para su ejecución.

A continuación se procedió a realizar una definición de una arquitectura híbrida entre Cliente-Servidor y P2P, denominada *OnDeGaS* (*On Demand Game Service*). Esta incipiente arquitectura permitió el desarrollo de las funcionalidades básicas necesarias para la ejecución eficiente de aplicaciones tipo MMOFPS y MMORPG dotando de escalabilidad al sistema al aumentar el número de jugadores.

Finalmente una extensa evaluación de la arquitectura propuesta, basada en simulaciones, se ha llevado a cabo para ambos tipos de juegos bajo distintos tipos de escenarios.

Para la realización del trabajo fue necesario desarrollar los siguientes puntos relacionados con la definición del modelo, las políticas de distribución de carga y la experimentación con las mismas. En cada uno de los puntos se citan además las aportaciones a que han dado lugar.

- **OnDeGaS, sistema híbrido CS/P2P.** Se propuso un nuevo entorno híbrido de gestión para juegos MMOG denominado *OnDeGaS* (*On Demand Game Service*). El sistema *OnDeGaS* está basado en los paradigmas Cliente-Servidor y P2P. La parte P2P consta de zonas de juego distribuidas gestionadas por los propios jugadores, que han sido seleccionados previo proceso de búsqueda basado en latencia. Por tanto, el sistema híbrido actúa mediante una zona central gestionada por un ser-

vidor estable, siendo el área P2P explotada en casos de sobrecarga del área central. Para la validación del sistema *OnDeGaS* se desarrollaron un conjunto de pruebas bajo un entorno de simulación y se aplicaron en primer lugar a juegos MMOFPS. El proceso de validación y análisis ha permitido verificar lo siguiente respecto el sistema *OnDeGaS*:

- El sistema es capaz de escalar en cada momento acorde con la demanda de los jugadores que se incorporan al mismo.
- En función del tipo de juego es posible determinar con precisión los valores apropiados de tiempo de espera en cola y número de jugadores por partida que aseguran un rendimiento adecuado.
- La latencia en el área P2P es lo suficientemente baja como para asegurar calidad de servicio para este tipo de juegos.
- La inclusión de un servidor replicado en las zonas del área P2P conlleva un efecto positivo en la tolerancia a fallos del sistema.

La descripción del sistema *OnDeGaS* y los resultados de la validación fueron publicados en:

[BGR10] I. Barri, F. Giné and C. Roig. *A Scalable Hybrid P2P System for MMOFPS*, Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2010). IEEE Computer Society, pages 341-347, 2010.

- **Distribución de juegos MMOFPS en el sistema OnDeGaS.** Partiendo del sistema *OnDeGaS*, se propusieron nuevos mecanismos de distribución de cómputo en juegos tipo MMOFPS basados en la distribución de partidas al área P2P cuando éstas no pueden ser ejecutadas en el servidor central por motivos de sobrecarga. Adicionalmente, se desarrolló un mecanismo de distribución que contempla la inherente heterogeneidad de los nodos del área P2P. Dicha heterogeneidad se centró en las diferencias existentes en el valor de latencia y número de núcleos del procesador de las máquinas de los jugadores. El análisis experimental realizado de las políticas de distribución de cómputo permitió verificar que:

- El método de distribución basado en partidas, como ente indivisible, respeta las características intrínsecas del juego. Esto proporciona un rendimiento que permitiría que el sistema pudiese ser explotado en un entorno comercial de juegos MMOFPS.
- Contemplar la heterogeneidad de los nodos permite un mayor aprovechamiento de recursos del área P2P y una explotación más eficiente del área distribuida.
- Tener en cuenta la heterogeneidad de los nodos tiene un efecto positivo en el tiempo de espera de los jugadores en cola.

Las políticas de distribución de partidas, así como las modificaciones del sistema *OnDeGaS* para trabajar teniendo en cuenta la heterogeneidad de los nodos y su evaluación fueron publicadas en:

[BRRG11] I. Barri, J. Rius, F. Giné and C. Roig. *Dealing with Heterogeneity for Mapping MMOFPS in Distributed Systems*, Euro-Par 2010 Parallel Processing Workshops Springer Berlin / Heidelberg, volume 6586, pages 51-61, 2011.

[BRGS11] I. Barri, C. Roig, F. Giné and F. Solsona *Mapping MMOFPS over Heterogeneous Distributed Systems*, The Journal of Supercomputing Springer Netherlands, volume 58, issue 3, pages 341-348, 2011.

- **Distribución de juegos MMORPG en el sistema OnDeGaS.** Los MMORPG, a diferencia de los MMOFPS, requieren un mundo virtual persistente, siendo la tolerancia a fallos mucho más trascendente puesto que los avances que el jugador va realizando están sujetos a un sistema fiable y siempre accesible. A diferencia de otros trabajos, en los que se equilibra la carga computacional mediante la distribución de jugadores individuales, en esta tesis se ha propuesto incorporar a *OnDeGaS* un mecanismo de distribución basado en juegos auxiliares puesto que están aislados del mundo virtual permanente y del resto de jugadores. Esto ha permitido acotar mucho más el problema y minimizar los costes de gestión.

A partir de las políticas de distribución de juegos auxiliares propuesta para MMORPG, se ha desarrollado un estudio comparativo, mediante simulación, para evaluar el rendimiento del sistema comparado con una política clásica basada en el balanceo de jugadores individuales. Dicho estudio ha permitido verificar los siguientes aspectos:

- Las políticas de distribución de cómputo basadas en la asignación de juegos auxiliares de los MMORPG, continúan dotando de escalabilidad al sistema *OnDeGaS*.
- La distribución basada en juegos auxiliares mantiene los estándares de calidad de servicio y tolerancia a fallos requeridos por los juegos MMORPG.
- El equilibrio de carga de los servidores del juego logrado con la política desarrollada es mayor que la que se puede alcanzar con una política de balanceo clásica de distribución de jugadores individuales.
- El coste de comunicaciones introducido por nuestra política es mucho menor, aportando, de esta manera una mejoría en la calidad de servicio.

La descripción del sistema propuesto, junto con los mecanismos de distribución de juegos auxiliares de los MMORPG, así como de las evaluaciones del sistema mediante la simulación y el análisis de los costes de comunicación formalizados analíticamente, fueron publicados en:

[BGR11] I. Barri, F. Giné and C. Roig. *A Hybrid P2P System to Support MMORPG Playability*. Proceedings of the 2011 High Performance Computing and Communications (HPCC '11) IEEE Computer Society, pages 569-574, 2011.

En Revisión I. Barri, C. Roig. F. Giné. *Distributing Game Instances in a Hybrid Client-Server/P2P System to Support MMORPG Playability*. Journal of Parallel and Distributed Computing.

6.2. Líneas abiertas

En base a la experiencia obtenida en el desarrollo de esta tesis, han ido surgiendo nuevas líneas de investigación que pueden contribuir a completar el nivel de refinamiento y la amplitud del problema abordado en un entorno de ejecución eficiente de juegos MMOG en arquitecturas híbridas Cliente-Servidor/P2P con calidad de servicio Estas líneas son:

1. La persistencia de los datos del juego es una característica que se da por sentada en los juegos MMORPG que son ejecutados en entornos Cliente-Servidor. Sin embargo, esto no es así cuando se trata de entornos distribuidos en los que los propios jugadores tienen que gestionar partes del juego. Con el fin de acercar el modelo distribuido al centralizado, por lo que respecta a la persistencia de los datos, se podría proponer un nuevo modelo de gestión y almacenaje de la información distribuido, de modo que siempre estuviera accesible, aún cuando se quiera almacenar o acceder desde una red P2P.

2. Otro punto relacionado con la persistencia de los datos es la tolerancia a fallos de los nodos de la red P2P que ejercen algún tipo de tarea de gestión del juego. Por tanto, con el fin de dotar de mayor estabilidad al sistema híbrido sería muy interesante mejorar aún más, los mecanismos de tolerancia a fallos aportados en esta tesis. Para ello, es necesario mejorar los algoritmos de búsqueda, con el propósito de seleccionar los nodos más fiables para asignarles roles de responsabilidad en la red distribuida. A modo de ejemplo, unos de los posibles refinamientos que se podrían aplicar en este campo, sería el de dotar al sistema de un módulo de predicción, que sumado al histórico de sesiones de los jugadores, permitiera lograr asignaciones más fiables.

3. Con los años, los recursos computacionales van aumentado sus capacidades. Sería interesante contemplar la heterogeneidad del sistema más allá de los valores de latencia y número de núcleos libres de los jugadores. De este modo se podrían añadir otros recursos como la memoria RAM o el ancho de banda, con el fin de refinar aún más las búsquedas

de servidores para las zonas distribuidas, hecho que permitirá mejorar el rendimiento del sistema híbrido.

4. En sistemas en los que se involucra la interacción de miles de usuarios al mismo tiempo, es necesario asegurar un orden y grado de confiabilidad. Por tanto, la implementación de sistemas de detección y eliminación de jugadores maliciosos, es uno de los puntos que requiere más esfuerzo en el diseño de MMOGs. Por consiguiente, es recomendable aplicar mecanismos de prevención y supresión de tramposos.

5. Lograr que los propios jugadores cedan sus recursos de cómputo ociosos, no es una tarea sencilla, puesto que estos jugadores esperarán algún tipo de recompensa o ventaja. Por tanto, desarrollar mecanismos de contabilidad e incentivación para motivar la cesión y/o compartición de recursos de cómputo es fundamental. Para ello es necesario establecer criterios que primen de algún modo aquellos jugadores con un mayor grado de participación en el entorno.

Bibliografía

[ABS06] Jeffrey Pang Ashwin Bharambe and Srinivasan Seshan. Colyseus: A Distributed Architecture for Online Multiplayer Games. In *Proc. Symposium on Networked Systems Design and Implementation (NSDI)*, 2006.

[Act] Activision. Enlace a la web principal de Call of Duty.

[AS08] Dewan T. Ahmed and Shervin Shirmohammadi. A Dynamic Area of Interest Management and Collaboration Model for P2P MMOGs. In *DS-RT '08: Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*. IEEE Computer Society, 2008.

[AT06] Marios Assiotis and Velin Tzanov. A Distributed Architecture for MMORPG. In *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '06, 2006.

[BA08] Eliya Buyukkaya and Maha Abdallah. Data Management in Voronoi based P2P Gaming. In *Data Management in Voronoi based P2P Gaming*, pages 1050–1053, 2008.

[BAS04] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. *SIGCOMM Comput. Commun. Rev.*, 2004.

[BF93] Steve Benford and Lennart Fahlén. A Spatial Model of Interaction in Large Virtual Environments. In *Proceedings of the Third Conference on European Conference on Computer-Supported Cooperative Work*. Kluwer Academic Publishers, 1993.

[BGR10] Ignasi Barri, Francesc Giné, and Concepció Roig. A Scalable Hybrid P2P System for MMOFPS. In *Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, PDP '10, pages 341–347. IEEE Computer Society, 2010.

[BGR11] Ignasi Barri, Francesc Giné, and Concepció Roig. A Hybrid P2P System to Support MMORPG Playability. In *Proceedings of the 2011 IEEE International Conference on High Performance Computing and Communications*, HPCC '11, pages 569–574. IEEE Computer Society, 2011.

[BKV06] Jean-Sébastien Boulanger, Jörg Kienzle, and Clark Verbrugge. Comparing Interest Management Algorithms for Massively Multiplayer Games. In *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames. ACM, 2006.

[Bli] Blizzard. Enlace a la web principal de World of Warcraft.

[BLL07] Nathaniel E. Baughman, Marc Liberatore, and Brian Neil Levine. Cheat-Proof Playout for Centralized and Peer-to-Peer Gaming. *IEEE/ACM Transactions on Networking*, 15:1–13, 2007.

[BRGS11] I. Barri, C. Roig, F. Giné, and F. Solsona. Mapping MMOFPS over Heterogeneous Distributed Systems. *The Journal of Supercomputing*, 58:341–348, 2011.

[BRRG11] Ignasi Barri, Josep Rius, Concepció Roig, and Francesc Giné. Dealing with Heterogeneity for Mapping MMOFPS in Distri-

buted Systems. In *Euro-Par 2010 Parallel Processing Workshops*, volume 6586 of *Lecture Notes in Computer Science*, pages 51–61. Springer Berlin / Heidelberg, 2011.

[CDKR02] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. SCRIBE: A Large-Scale and Decentralized Application-Level Multicast Infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)*, 20:1489–1499, 2002.

[CFFS05] Chris Chambers, Wu-chang Feng, Wu-chi Feng, and Debanjan Saha. Mitigating Information Exposure to Cheaters in Real-Time Strategy Games. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV. ACM, 2005.

[Cha10] Tom Chatfield. *Why Games are the 21st Century's Most Serious Business*. Virgin Books, 2010.

[CHHL05] Kuan-Ta Chen, Polly Huang, Chun-Ying Huang, and Chin-Laung Lei. Game Traffic Analysis: an MMORPG Perspective. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '05, pages 19–24. ACM, 2005.

[CHJ08] Mo-Che Chan, Shun-Yun Hu, and Jehn-Ruey Jiang. An Efficient and Secure Event Signature (EASES) Protocol for Peer-to-Peer Massively Multiplayer Online Games. *Computer Networks*, 52:1838–1845, 2008.

[CK05] Fang Chen and Vana Kalogeraki. Adaptive Real-Time Update Dissemination in Distributed Virtual Simulation Environments. In *Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, ISORC. IEEE Computer Society, 2005.

[CM04] Bei Di Chen and Muthucumaru Maheswaran. A Fair Synchronization Protocol with Cheat Proofing for Decentralized Online Multiplayer Games. In *Proceedings of the Network Computing and Applications, Third IEEE International Symposium*, NCA, pages 372–375. IEEE Computer Society, 2004.

[Cora] Atomic Blue Corporation. Enlace a la web principal de estadísticas de PlaneShift.

[Corb] Atomic Blue Corporation. Enlace a la web principal de PlaneShift.

[CRJ+04] Fabio Reis Cecin, Rodrigo Real, Rafael de Oliveira Jannone, Claudio Fernando Resin Geyer, Marcio Garcia Martins, and Jorge Luis Victoria Barbosa. FreeMMG: A Scalable and Cheat-Resistant Distribution Model for Internet Games. In *Proceedings of the 8th IEEE International Symposium on Distributed Simulation and Real-Time Applications*. IEEE Computer Society, 2004.

[CWD+05] Jin Chen, Baohua Wu, Margaret Delap, Björn Knutsson, Honghui Lu, and Cristiana Amza. Locality Aware Dynamic Load Management for Massively Multiplayer Games. In *Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '05, 2005.

[CYB+07] Luther Chan, James Yong, Jiaqiang Bai, Ben Leong, and Raymond Tan. Hydra: A Massively-Multiplayer Peer-to-Peer Architecture for the Game Developer. In *NetGames: Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games*. ACM, 2007.

[DC90] Stephen E. Deering and David R. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transactions on Computer Systems*, 8:85–110, 1990.

[DNBB00] Christophe Diot, Brian Neil, Levine Bryan, and Kassem Doug Balensiefen. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Network*, 2000.

[DTHK05] Scott Douglas, Egemen Tanin, Aaron Harwood, and Shanika Karunasekera. Enabling Massively Multi-Player Online Gaming Applications on a P2P Architecture. In *Proceedings of the IEEE International Conference on Information and Automation*. IEEE, 2005.

[DVVDBV+05] Bart De Vleeschauwer, Bruno Van Den Bossche, Tom Verdickt, Filip De Turck, Bart Dhoedt, and Piet Demeester. Dynamic Microcell Assignment for Massively Multiplayer Online Gaming. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and System Support for Games*, 2005.

[DZ03] Ta Nguyen Binh Duong and Suiping Zhou. A Dynamic Load Sharing Algorithm for Massively Multiplayer Online Games. In *Networks, 2003. The 11th IEEE International Conference on*, ICON, 2003.

[ESRM03] A. El-Sayed, V. Roca, and L. Mathy. A Survey of Proposals for an Alternative Group Communication Service. *Network, IEEE*, 17(1):46–51, 2003.

[EZ01] T. S. Eugene and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *INFOCOM*, 2001.

[FCFW02] Wu-chang Feng, Francis Chang, Wu-chi Feng, and Jonathan Walpole. Provisioning On-line Games: A Traffic Analysis of a Busy Counter-Strike Server. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*, IMW '02, pages 151–156. ACM, 2002.

[Froa] Frogster. Enlace a la web principal de Runes of Magic.

[Frob] Frozesand. Enlace a la web principal de Urban Terror.

[FTT07] Lu Fan, Hamish Taylor, and Phil Trinder. Mediator: A Design Framework for P2P MMOGs. In *Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames. ACM, 2007.

[FTT08] Lu Fan, Hamish Taylor, and Phil Trinder. MAMBO: Membership-Aware Multicast with Bushiness Optimisation. *Short papers of the 2nd International Conference on Distributed Event-Based Systems*, 2008.

[FTT10] Lu Fan, Phil Trinder, and Hamish Taylor. Design Issues for Peer-to-Peer Massively Multiplayer Online Games. *International Journal of Advanced Media and Communications*, 4, 2010.

[Fun06] Yeung Siu Fung. Hack-Proof Synchronization Protocol for Multi-Player Online Games. In *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames. ACM, 2006.

[FWW02] Stefan Fiedler, Michael Wallner, and Michael Weber. A Communication Architecture for Massive Multiplayer Games. In *Proceedings of the 1st Workshop on Network and System Support for Games*, NetGames. ACM, 2002.

[Gama] GameForge. Enlace a la web principal de Ikariam.

[Gamb] Gameforge. Enlace a la web principal de Ogame.

[GDZL04] Chris Gauthier Dickey, Daniel Zappala, and Virginia Lo. A Fully Distributed Architecture for Massively Multiplayer Online Games. In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames. ACM, 2004.

[GH04] Meng Ye Gao Huang. Modeling System Performance in MMORPG. *IEEE Global Telecommunications Conference Workshops*, pages 512–518, 2004.

[GJA03] Minaxi Gupta, Paul Judge, and Mostafa Ammar. A Reputation System for Peer-to-Peer Networks. In *Proceedings of the 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '03. ACM, 2003.

[GLZ05] Chris GauthierDickey, Virginia Lo, and Daniel Zappala. Using n-trees for scalable event ordering in peer-to-peer Games. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV. ACM, 2005.

[GS06] Marcin Gorawski and Karol Stachurski. A Secure Event Agreement (SEA) protocol for Peer-to-Peer Games. In *Proceedings of the First International Conference on Availability, Reliability and Security*. IEEE Computer Society, 2006.

[GZLM04] Chris GauthierDickey, Daniel Zappala, Virginia Lo, and James Marr. Low Latency and Cheat-Proof Event Ordering for Peer-to-Peer Games. In *Proceedings of the 14th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV. ACM, 2004.

[HBH06] Thorsten Hampel, Thomas Bopp, and Robert Hinn. A Peer-to-Peer Architecture for Massive Multiplayer Online Games. In *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames. ACM, 2006.

[HCC06] Shun-Yun Hu, Jui-Fa Chen, and Tsu-Han Chen. Von: A Scalable Peer-to-Peer Network for Virtual Environments. *IEEE Network*, 2006.

[HCJ08] Shun-Yun Hu, Shao-Chen Chang, and Jehn-Ruey Jiang. Voronoi State Management for Peer-to-Peer Massively Multiplayer Online Games. In *Consumer Communications and Networking Conference, CCNC 5th IEEE*, pages 1134–1138, 2008.

[HHJ08] Guan-Yu Huang, Shun-Yun Hu, and Jehn-Ruey Jiang. Scalable Reputation Management with Trustworthy User Selection for P2P MMOGs. *International Journal of Advanced Media and Communications*, 2:380–401, 2008.

[HL04] Shun-Yun Hu and Guan-Ming Liao. Scalable Peer-to-Peer Networked Virtual Environment. In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames. ACM, 2004.

[HM02] J. C. Herz and M. R. Macedonia. *Computer Games and the Military: Two views.* Defense Horizons, 2002.

[IHK04] Takuji Iimura, Hiroaki Hazeyama, and Youki Kadobayashi. Zoned Federation of Game Servers: A Peer-to-Peer Approach to Scalable Multi-Player Online Games. In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames, 2004.

[IYM+06] Takato Izaiku, Shinya Yamamoto, Yoshihiro Murata, Naoki Shibata, Keiichi Yasumoto, and Minoru Ito. Cheat Detection for MMORPG on P2P Environments. In *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames. ACM, 2006.

[Jay02] R. Jayakanthan. *Application of Computer Games in the Field of Education.* The Electronic Library, 2002.

[JZ08] Jared Jardine and Daniel Zappala. A Hybrid Architecture for Massively Multiplayer Online Games. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames. ACM, 2008.

[KHC03] J. Kim, E. Hong, and Y. Choi. Measurement and Analysis of a Massively Multiplayer Online Role Playing Game Traffic. 2003.

[Kir04] J. Kirriemuir. *Video gaming, education and digital learning technologies*. D-Lib, 2004.

[KLXH04] B. Knutsson, Honghui Lu, Wei Xu, and B. Hopkins. Peer-to-Peer Support for Massively Multiplayer Games. In *INFOCOM. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, 2004.

[KSGM03] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The Eigentrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03. ACM, 2003.

[KTCB05] Patric Kabus, Wesley W. Terpstra, Mariano Cilia, and Alejandro P. Buchmann. Addressing Cheating in Distributed MMOGs. In *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames. ACM, 2005.

[LC] Yeng-Ting Lee and Kuan-Ta Chen. Is Server Consolidation Beneficial to MMORPG? A Case Study of World of Warcraft. *IEEE International Conference on*.

[LL08] Huey-Ing Liu and Yun-Ting Lo. Dacap - A Distributed Anti-Cheating Peer to Peer Architecture for Massive Multiplayer On-line Role Playing Game. In *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, 2008.

[LLW07] Shanshan Liu, Jinlong Li, and Xufa Wang. Local Reputation for P2P MMOG Design. In *Proceedings of the Eighth International Conference on Parallel and Distributed Computing,*

Applications and Technologies, PDCAT '07. IEEE Computer Society, 2007.

[LPBC07] Peter Laurens, Richard F. Paige, Phillip J. Brooke, and Howard Chivers. A Novel Approach to the Detection of Cheating in Multiplayer Online Games. In *Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems*. IEEE Computer Society, 2007.

[LPM06] Fengyun Lu, Simon Parkin, and Graham Morgan. Load Balancing for Massively Multiplayer Online Games. In *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '06, 2006.

[LS06] Hsiu-Hui Lee and Chin-Hua Sun. Load-Balancing for Peer-to-Peer Networked Virtual Environment. In *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames. ACM, 2006.

[Mel04] Carnegie Mellon. *Client/Server Software Architectures-An Overview*. Carnegie Mellon University, 2004.

[MKMA04] N. Matsumoto, Y. Kawahara, H. Morikawa, and T. Aoyama. A Scalable and Low Delay Communication Scheme for Networked Virtual Environments. In *Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004. IEEE*, pages 529 – 535, 2004.

[Mun10] Banco Mundial. Indicadores de desarrollo mundial, Producto Interior Bruto (PIB). 2010.

[MZP+94] Michael R. Macedonia, Michael J. Zyda, David R. Pratt, Paul T. Barham, and Steven Zeswitz. Npsnet: A Network Software Architecture For Large Scale Virtual Environments, 1994.

[NCS] NCSOFT. Enlace a la web principal de Lineage.

[RBD04] Sean Rooney, David Bauer, and Rudy Deydier. A Federated Peer-to-Peer Network Game Architecture. *IEEE Communications Magazine*, 42(5):114–122, 2004.

[RD01a] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale Peer-to-Peer Systems. *Proceegins Middleware IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, 2001.

[RD01b] Antony Rowstron and Peter Druschel. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. *SIGOPS Oper. Syst. Rev.*, 35:188–201, 2001.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. *SIGCOMM Computer Communication Review*, 31:161–172, 2001.

[RHKS01] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Application-Level Multicast Using Content-Addressable Networks. In *Networked Group Communications*, pages 14–29, 2001.

[RM06] A. E. Rhalibi and M. Merabti. Interest Management and Scalability Issues in P2P MMOG. In *Consumer Communications and Networking Conference, (CCNC) 3rd IEEE*, volume 2, pages 1188–1192, 2006.

[RMO07] Silvia Rueda, Pedro Morillo, and Juan M. Orduña. A Peer-to-Peer Platform for Simulating Distributed Virtual Environments. In *Proceedings of the 13th International Conference on Parallel and Distributed Systems*, volume 2, pages 1–8. IEEE Computer Society, 2007.

[SLM04] Kier Storey, Fengyun Lu, and Graham Morgan. Determining Collisions between Moving Spheres for Distributed Virtual En-

vironments. In *Proceedings of the Computer Graphics International*, pages 140–147. IEEE Computer Society, 2004.

[Sni04] M. Snider. *Hollywood at the controls: Video Games feed off each other*. Usa Today 1D-2D, 2004.

[Ste] Steam. Enlace a la web principal de Counter Strike.

[Ste04] Constance A. Steinkuehler. Learning in Massively Multiplayer Online Games. In *Proceedings of the 6th International Conference on Learning Sciences*, ICLS '04, pages 521–528. International Society of the Learning Sciences, 2004.

[SZA08] Gayatri Swamynathan, Ben Y. Zhao, and Kevin C. Almeroth. Exploring the Feasibility of Proactive Reputations: Research Articles. *Concurrency Computation: Practice Experience*, 20:155–166, 2008.

[TTT+01] Lin Tong, Lin Tong, Lin Tong, Randy H. Katz, and Lin Tong. Bayeux: An Architecture for Scalable and Fault-Tolerant Wide-Area Data Dissemination. In *NOSSDAV 01: Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, page 11–20. ACM, 2001.

[Ubi03] Ubisoft. *PlanetSide*. Sony Online Entertainment, 2003.

[Vak02] S. Vaknin. *TrendSiters: Games people play*. Electronic Book Web, 2002.

[Wor02] IBM Developers Works. Charming Python: SimPy Simplifies Complex Models Simulate Discrete Simultaneous Events for Fun and Profit. 2002.

[YC06] M. Ye and L. Cheng. System-Performance Modeling for Massively Multiplayer Online Role-Playing Games. *IBM Systems Journal*, 45:45–58, 2006.

[YGM03] Beverly Yang and Hector Garcia-Molina. Designing a Super-Peer Network. *International Conference on Data Engineering*, 0:49, 2003.

[YKH04] Tatsuhiro Yonekura, Yoshihiro Kawano, and Dai Hanawa. Peer-to-peer networked field-type virtual environment by using atoz. In *Proceedings of the 2004 International Conference on Cyberworlds*, CW. IEEE Computer Society, 2004.

[YMYI05] Shinya Yamamoto, Yoshihiro Murata, Keiichi Yasumoto, and Minoru Ito. A Distributed Event Delivery Method with Load Balancing for MMORPG. In *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames. ACM, 2005.

[YV05] Anthony Peiqun Yu and Son T. Vuong. MOPAR: A Mobile Peer-to-Peer Overlay Architecture for Interest Management of Massively Multiplayer Online Games. In *Online Games, NOSSDAV'05, ACM*. ACM Press, 2005.

[ZA07] M. Zghaibeh and K. Anagnostakis. On the Impact of P2P Incentive Mechanisms on User Behavior. In *in NetEcon+IBC*, 2007.

[ZH03] Rongmei Zhang and Y. Charlie Hu. Borg: A Hybrid Protocol for Scalable Application-Level Multicast in Peer-to-Peer Networks. In *Proceedings of the 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV, pages 172–179. ACM, 2003.

[ZKJ01] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location. Technical report, Berkeley, CA, USA, 2001.