UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

CARLOS EDUARDO BENEVIDES BEZERRA

<PEGAR DO PLANODOC NO NOTE>

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Cláudio Fernando Resin Geyer
Orientador

Porto Alegre, janeiro de 2009

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Bezerra, Carlos Eduardo Benevides

<PEGAR DO PLANODOC NO NOTE> / Carlos Eduardo Benevides Bezerra. – Porto Alegre: PPGC da UFRGS, 2009.

67 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2009. Orientador: Cláudio Fernando Resin Geyer.

1. Jogos online maciçamente multijogador. 2. MMOG. 3. Simulação interativa distribuída. I. Geyer, Cláudio Fernando Resin. II. Título.

AGRADECIMENTOS

<TO-DO>

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ADI	*Área de Interesse*
API	*Application Programming Interface*
CVE	*Collaborative Virtual Environment*
CPU	*Central Processing Unit*
DHT	*Distributed Hash Table*
DIS	*Distributed Interactive Simulation*
FPS	*First-Person Shooter*
IP	*Internet Protocol*
J2SE	*Java 2 Standard Edition*
MMG	*Massively Multiplayer Game*
MMOFPS	*Massively Multiplayer Online First-Person Shooter*
MMORPG	*Massively Multiplayer Online Role-Playing Game*
MMORTS	*Massively Multiplayer Online Real-Time Strategy*
NVE	*Networked Virtual Environment*
RTP	*Real-time Transport Protocol*
RTS	*Real-Time Strategy*
TCP	*Transmission Control Protocol*
TSS	*Trailing State Synchronization*
UDP	*Unreliable Datagram Protocol*
URL	*Uniform Resource Locator*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

RESUMO

<lalalalalalalalá ... fazer resumo lalalalalalá....>

A popularização das tecnologias de acesso à Internet por "banda larga" suportam a crescente proliferação de aplicações do tipo "par-a-par" (peer-to-peer), onde o usuário doméstico, tipicamente consumidor de informação, passa também a atuar como provedor. De forma simultânea, há uma popularização crescente dos jogos em rede, especialmente dos "jogos maciçamente multijogador" (MMG ou *massively multiplayer game*) onde milhares de jogadores interagem, em tempo real, em mundos virtuais de estado persistente. Os MMGs disponíveis atualmente, como *EverQuest* e *Ultima Online*, são implementados como sistemas centralizados, que realizam toda a simulação do jogo no "lado servidor". Este modelo propicia controle de acesso ao jogo pelo servidor, além de ser muito resistente a jogadores trapaceiros. Porém, a abordagem cliente-servidor não é suficientemente escalável, especialmente para pequenas empresas ou projetos de pesquisa que não podem pagar os altos custos de processamento e comunicação dos servidores de MMGs centralizados. Este trabalho propõe o FreeMMG, um modelo híbrido, cliente-servidor e par-a-par, de suporte a jogos maciçamente multijogador de estratégia em tempo real (MMORTS ou *massively multiplayer online real-time strategy*). O servidor FreeMMG é escalável pois delega a maior parte da tarefa de simulação do jogo para uma rede par-a-par, formada pelos clientes. É demonstrado que o FreeMMG é resistente a certos tipos de trapaças, pois cada segmento da simulação distribuída é replicado em vários clientes. Como protótipo do modelo, foi implementado o jogo *FreeMMG Wizards*, que foi utilizado para gerar testes de escalabilidade com até 300 clientes simulados e conectados simultaneamente no mesmo servidor. Os resultados de escalabilidade obtidos são promissores, pois mostram que o tráfego gerado em uma rede FreeMMG, entre servidor e clientes, é significativamente menor se comparado com uma alternativa puramente cliente-servidor, especialmente se for considerado o suporte a jogos maciçamente multijogador de estratégia em tempo real.

Palavras-chave: Jogos online maciçamente multijogador, MMOG, simulação interativa distribuída.

FreeMMG: A Client-Server and Peer-to-Peer Support Architecture for Massively Distributed Games

ABSTRACT

The increasing adoption of "broadband" Internet access technology fosters the increasing development of networked "peer-to-peer" applications, where the end-user, usually the consumer of information, begins to act also as a producer of information. Simultaneously, there is a growing popularity of networked games, especially "massively multiplayer games" (MMGs) where thousands of players interact, in real time, in persistent-state virtual worlds. State-of-the-art massively multiplayer games such as EverQuest and Ultima Online are currently implemented as centralized, client-server systems, which run the entire game simulation on the "server-side". Although this approach allows the development of commercially viable MMG services, with game access control and player cheating prevention, the processing and communications costs associated with running a MMG server are often too high for small companies or research projects. This dissertation proposes FreeMMG, a mixed peer-to-peer and client-server approach to the distribution aspect of MMGs. It is argued that the FreeMMG model supports scalable, cheat-resistant, massively multiplayer real-time strategy games (MMORTS games) using a lightweight server that delegates the bulk of the game simulation to a peer-to-peer network of game clients. It is shown that FreeMMG is resistant to certain kinds of cheating attempts because each segment of the distributed simulation is replicated by several clients. A working prototype game called FreeMMG Wizards is presented, together with scalability test results featuring up to 300 simulated game clients connected to a FreeMMG server. The obtained results are promising, by showing that the generated server traffic in a FreeMMG network, between the server and the clients, is substantially lower if compared with purely client-server alternatives, especially if the support for massively multiplayer real-time strategy games is considered.

Keywords: massively multiplayer games, distributed interactive simulation, peer-to-peer systems, real-time strategy games.

1 INTRODUÇÃO

2 CONTEXTO, ESTADO DA ARTE E MOTIVAÇÃO

3 MODELO BASE

4 A^3: UM ALGORITMO DE OTIMIZAÇÃO POR ÁREA DE INTERESSE

Traditionally, a central server is used to provide support to MMGs (massively multiplayer games), where the number of participants is in the order of tens of thousands. Much work has been done trying to create a fully peer-to-peer model to support this kind of application, in order to minimize the maintenance cost of its infra-structure, but critical questions remain. Examples of the problems relative to peer-to-peer MMG support systems are: vulnerability to cheating, overload of the upload links of the peers and difficulty to maintain consistency of the simulation among the participants. In this work, it is proposed the utilization of geographically distributed lower-cost nodes, working as a distributed server to the game. The distribution model and some related works are also presented. To address the communication cost imposed to the servers, we specify a novell refinement to the area of interest technique, significantly reducing the necessary bandwidth. Simulations have been made with ns-2, comparing different area of interest algorithms. The results show that our approach achieves the least bandwidth utilization, with a 33.10% maximum traffic reduction and 33.58% average traffic reduction, when compared to other area of interest algorithms.

Atualmente, jogos eletrônicos têm se tornado bastante populares, especialmente os jogos maciçamente multijogador, onde há um número de participantes simultâneos da ordem de dezenas de milhares (CECIN et al., 2004). Como exemplos, podemos citar World of Warcraft (BLIZZARD, 2004), Lineage II (NCSOFT, 2003) e Guild Wars (ARENANET, 2005).

Usualmente, o suporte de rede para este tipo de aplicação consiste em um servidor central com recursos - capacidade de processamento e largura de banda para comunicação com os jogadores - super-dimensionados, ao qual se conectam as máquinas clientes. Cada jogador interage através de um destes clientes, que envia suas ações para o servidor, que as processa, verificando que alterações no jogo elas causam, e difunde o resultado para todos os clientes envolvidos. Em virtude do número de participantes simultâneos que este tipo de jogo costuma ter, percebe-se que tais tarefas demandam por uma quantidade de recursos significativa, no que tange a poder de processamento e, principalmente, largura de banda disponível para que sejam recebidas as ações dos jogadores e enviadas as atualizações de estado.

Nos últimos anos, têm-se pesquisado alternativas à abordagem com servidor centralizado. Uma delas é a distribuição, entre os próprios participantes, tanto da simulação do jogo quanto da responsabilidade de atualizarem-se entre si quando realizam ações. A comunicação entre eles ocorre par-a-par, formando uma rede descentralizada (SCHIELE et al., 2007). Esta abordagem seria o ideal, não fossem alguns problemas que lhe são

inerentes. Por exemplo, como os jogadores participam do processamento da simulação, é necessário que eles entrem em acordo no que diz respeito ao estado da partida, sob pena de haver inconsistências caso isto não seja feito.

Outra questão se refere ao número de envios que cada participante tem que executar. No modelo cliente-servidor, basta que cada um envie suas ações para o servidor, que se encarrega de simular e difundir o novo estado para os outros jogadores. No caso do modelo par-a-par, cada par envolvido torna-se responsável por processar suas ações e enviar as atualizações de estado para os outros participantes. O problema disto reside no fato de que não se pode garantir que todos os jogadores possuam conexões de rede com largura de banda suficiente. Por fim, sem um servidor central, que poderia atuar como árbitro, o jogo torna-se dependente da simulação que os próprios jogadores executam, que pode ser desvirtuada de forma a chegar a um resultado inválido, que beneficie indevidamente determinado jogador ou mesmo que invalide a sessão de jogo.

Além do modelo par-a-par, existe também a alternativa de utilizar um servidor distribuído, em que diversos nodos conectados entre si dividem a tarefa de simular o jogo, como também de enviar as atualizações de estado aos jogadores (ASSIOTIS; TZANOV, 2006). Tal abordagem possibilita o uso de computadores de menor custo para comporem o sistema distribuído servidor, barateando a infra-estrutura de suporte. Questões como consistência e vulnerabilidade a trapaça podem ser abstraídas, restringindo o conjunto de nodos servidores a computadores comprovadamente confiáveis, o que é plausível, levando em conta que o número de nodos servidores deverá ser algumas ordens de grandeza menor do que o número de jogadores. Além disso, não é necessário exigir que cada jogador envie atualizações de estado para todos os outros jogadores. Com menores exigências de largura de banda e processamento das máquinas clientes, o jogo torna-se acessível para um maior público.

No entanto, para evitar que o custo de manutenção do sistema distribuído servidor como um todo não se aproxime do custo de manutenção de um servidor central, é necessário realizar algumas otimizações com o intuito de reduzir a largura de banda necessária para cada um dos nodos. O presente trabalho propõe uma técnica para reduzir o consumo de largura de banda causado pelo tráfego do jogo entre os servidores e os clientes, diminuindo a quantidade de recursos necessários, através de um refinamento da técnica de gerenciamento de interesse (BOULANGER; KIENZLE; VERBRUGGE, 2006) dos jogadores. O princípio básico desta técnica é que cada participante do jogo receba apenas atualizações de jogadores cujo estado lhes seja relevante. Foram realizadas simulações comparando a proposta deste trabalho com técnicas convencionais, obtendo resultados significativos.

O artigo está dividido da seguinte maneira: na seção 4.1 são citados alguns trabalhos relacionados onde buscou-se distribuir o servidor do jogo; na seção 4.2, são apresentadas as definições de alguns conceitos utilizados ao longo do texto; na seção 4.3, é descrito o modelo de distribuição proposto; na seção 4.4 é apresentada a otimização proposta para reduzir o tráfego sem comprometer a qualidade do jogo; nas seções 4.6 e 4.7 é descrita a simulação realizada para validar a técnica proposta e os resultados obtidos, respectivamente e, na seção 4.8, são apresentadas as conclusões a que se chegou neste trabalho.

4.1 Related Work

Como já foi dito, alguns trabalhos já foram feitos nos últimos anos visando distribuir o suporte a jogos maciçamente multijogador. Uma das abordagens é o modelo par-a-par, que tem algumas dificuldades, no que se refere a consistência do estado do jogo nos diferentes pares participantes, vulnerabilidade a trapaça e uso eficiente de largura de banda. Alguns autores propõem abordagens cujo objetivo é minimizar estes problemas. Um destes trabalhos (SCHIELE et al., 2007) propõe a divisão do ambiente virtual simulado no jogo em regiões, e dentro de cada região é escolhido um par que será eleito coordenador daquela região. Sua função será a de gerenciar o interesse dos jogadores, verificando para quais pares cada atualização realmente precisa ser enviada. Dessa forma, reduz-se o uso de largura de banda de envio dos pares. No entanto, o uso de largura de banda de envio de cada participante ainda tende a ser significativamente superior àquele necessário quando utilizado o modelo cliente-servidor, pois neste é necessário apenas que cada jogador envie suas ações para um único destino. No modelo par-a-par, cada jogador deve atualizar, normalmente, mais de um outro jogador. Além disso, é necessário que o par escolhido para gerenciar o interesse naquela região seja confiável.

Outro trabalho voltado para o modelo par-a-par (IIMURA; HAZEYAMA; KADOBAYASHI, 2004) tem uma abordagem semelhante à de (SCHIELE et al., 2007), mas sugere que, para cada região do ambiente virtual, seja criada uma “federação de servidores", formada por pares escolhidos entre os participantes. A simulação torna-se mais confiável, já que diferentes nodos irão gerenciar aquele lugar no mundo do jogo e precisarão estar em acordo para que a simulação prossiga. Porém, o risco dos nodos escolhidos para gerenciarem aquela região cometerem trapaça de conluio (YAN; RANDELL, 2005) não é eliminado. Além disso, o próprio acordo entre os nodos servidores, que provê maior confiabilidade na simulação, implica em grande quantidade de tráfego entre os nodos participantes, além de potencialmente atrasar cada passo da simulação.

Um grande problema das arquiteturas par-a-par, no que diz respeito à utilização de gerenciamento de interesse é que cada par é responsável por parte da simulação e por decidir para quem sua atualização de estado interessaria. Assumindo que haja apenas jogadores confiáveis, a técnica de gerenciamento de interesse pode ser útil. No entanto, supondo que determinado jogador seja malicioso, ele pode não enviar atualizações de seu próprio estado para algum outro jogador, que ficaria prejudicado no jogo. Sendo assim, o modelo de servidor distribuído é considerado mais adequado para utilização de técnicas de gerenciamento de interesse dos jogadores.

Um exemplo deste tipo de modelo é descrito em (ASSIOTIS; TZANOV, 2006), onde é proposta uma arquitetura distribuída para jogos maciçamente multijogador. Também é baseada na divisão do ambiente virtual do jogo em regiões, porém a cada uma destas estaria associado um nodo servidor. O jogador que estivesse situado em determinado lugar no mundo virtual deveria conectar-se ao servidor responsável por aquela região. Desta forma, cada servidor agruparia diferentes jogadores, baseado em sua localidade no ambiente do jogo. Para alcançar consistência entre os diferentes nodos servidores efetuando a simulação, é utilizado o conceito de travas. Quando um determinado nodo servidor precisa alterar o estado de uma entidade qualquer da partida, primeiro precisa obter acesso exclusivo àquela entidade. Para isso, ele negocia com os outros nodos servidores que possam também querer fazer alguma alteração, para somente então efetuar a mudança. Quando termina, o acesso é liberado, e os outros servidores são avisados através de mensagens.

A primeira grande restrição no trabalho de (ASSIOTIS; TZANOV, 2006), no entanto, é a premissa de que os nodos servidores estão conectados através de uma rede de alta ve-

locidade e baixa latência, o que não pode ser assumido quando se trata de nodos de mais baixo custo geograficamente distribuídos. Outro problema é que a questão da escalabilidade é tratada através da pura e simples expansão do ambiente virtual, supondo que os jogadores se espalharão por ele. Por último, sugere-se resolver o problema de haver um grande número de jogadores no mesmo lugar através de sucessivos reparticionamentos recursivos das regiões, de forma a dividir os jogadores entre diferentes servidores. No entanto, existe um limite para o reparticionamento do ambiente virtual, e é deixado de lado o que fazer quando é atingido este limite.

No que se refere a gerenciamento de interesse, trabalhos como (MORSE et al., 1996; RAK; VAN HOOK, 1996; ZOU; AMMAR; DIOT, 2001; MORGAN; LU; STOREY, 2005) e (MINSON; THEODOROPOULOS, 2005) podem ser citados. Em (RAK; VAN HOOK, 1996), é proposto um esquema de gerenciamento de interesse baseado em um ambiente virtual divido em células de uma grade. A cada célula está associado um grupo de multicast. Cada participante da simulação se subscreve então no grupo de multicast da célula onde ele se encontra, assim como de células vizinhas se estiverem ao alcance de sua visão. Cada participante envia então suas atualizações de estado ao grupo de multicast da região onde se encontra.

Em (ZOU; AMMAR; DIOT, 2001), também são considerados esquemas de gerenciamento de interesse baseados em grupos de multicast. É feita uma comparação entre a formação de grupos baseado em célula, onde cada um está associado a uma célula de uma grade que compõe o ambiente virtual, e agrupamento baseado em objetos, onde para cada objeto existe um grupo multicast associado. Verificou-se que há uma compensação (tradeoff) entre o custo das mensagens de controle dos grupos de multicast e o custo das mensagens de atualização de estado propriamente ditas.

Um esquema de gerenciamento de interesse que utiliza um middleware orientado a mensagens é apresentado em (MORGAN; LU; STOREY, 2005). Dentre outros aspectos, este esquema faz uma predição do que será o interesse de determinado participante no futuro, baseado na posição e vetor velocidade do mesmo no ambiente virtual. Dessa forma, cada um começa a receber atualizações de estado de entidades que não estão ainda ao alcance de sua visão, mas que provavelmente estarão em um futuro próximo, tornando seus estados disponíveis assim que elas estiverem dentro do campo de visão.

Em (MORSE et al., 1996), é feito um apanhado de sistemas que utilizam a técnica de gerenciamento de interesse, salientando quais critérios são utilizados por cada um. É apresentada então uma taxonomia de tais sistemas, classificando-os de acordo com: modelo de comunicação, foco da filtragem e domínios de responsabilidade. O modelo pode ser *unicast*, *multicast* ou *broadcast*. O foco da filtragem refere-se a que tipo de características são observadas de cada objeto para realizar esta filtragem: podem ser *intrínsecas*, como o valor de atributos do objeto (e.g. coordenadas exatas de sua localização), ou *intrínsecas*, como a qual grupo multicast ele está associado. Por fim, o domínio de responsabilidade atribuída a um gerenciador de interesse, que verifica para quem cada estado é relevante, pode ser *dinâmico* ou *estático*. Por exemplo, se cada gerenciador é designado para controlar uma área fixa do ambiente virtual, seu domínio de responsabilidade é estático, mas se ele controla uma área que possa aumentar ou diminuir de tamanho, seu domínio de responsabilidade é dinâmico.

Levando em consideração que o modelo de comunicação multicast não é amplamente suportado na Internet (EL-SAYED, 2004), tanto por razões técnicas quanto comerciais, neste trabalho optou-se por seguir o modelo unicast, considerando que cada broadcast consiste na verdadem em um conjunto de sucessivas transmissões unicast, uma para cada

destino. Além disso, utiliza-se filtragem intrínseca e domínios de responsabilidade dinâmicos, para que haja maior precisão e, consequentemente, uma maior redução no tráfego de atualizações de estado (MORSE et al., 1996).

4.2 Definitions

Será utilizado o termo cliente para referir-se ao computador utilizado por cada jogador para conectar-se a um dos servidores do jogo, assim como o termo servidor fará referência a cada nodo integrante do sistema distribuído que estará servindo o jogo. É necessário descrever o modelo de suporte a jogos maciçamente jogador sobre o qual pretende-se utilizar o algoritmo de gerenciamente de interesse proposto. Ao longo do texto, serão utilizados alguns termos que precisam antes ser definidos:

Avatar é a representação do jogador no ambiente virtual. É através dele que o jogador interage com o mundo do jogo e com outros jogadores. Exemplos de avatar são os personagens controlados pelo jogador em jogos MMORPG, como World of Warcraft.

Entidades são as peças constituintes do mundo virtual. Exemplos de entidades são os próprios avatares dos jogadores, assim como avatares controlados por inteligência artificial do servidor - monstros dos MMORPGs, por exemplo - e dos objetos inanimados presentes no ambiente, tais como portas, armas e itens em geral com que os avatares possam interagir.

Estado é o conjunto de propriedades que podem ser observadas nas diferentes entidades do jogo. O estado global do mundo simulado é constituído dos estados individuais das diferentes entidades nele presentes.

Os jogadores interagem com o mundo do jogo através de **ações**. Uma ação é um comando do jogador como, por exemplo, mover seu avatar para determinada localização no mundo virtual, atacar outro jogador, tomar para si algum objeto disponível no ambiente e assim por diante. Em geral, ações modificam o estado de uma ou mais entidades presentes no jogo.

Região é um partição do ambiente virtual, sob responsabilidade de um único servidor. Dessa forma, jogadores cujos avatares estejam localizados na mesma região terão sua interação beneficiada, pois suas máquinas estarão conectadas ao mesmo servidor.

A **fronteira** entre duas regiões é a divisa entre as áreas que essas regiões ocupam. Quando um avatar está localizado próximo a uma fronteira, o servidor responsável pela região além desta fronteria é avisado a respeito da presença daquele avatar pelo servidor onde ele se encontra.

4.3 Distribution Model

Este trabalho baseia-se em um ambiente virtual particionado em regiões, cada uma gerenciada por um servidor. As regiões são contíguas, explorando a localidade dos avatares dos jogadores. Dessa forma, avatares próximos no jogo provavelmente estarão localizados na mesma região e, por conseguinte, os clientes dos jogadores a eles associados tenderão a estar conectados ao mesmo servidor, fazendo com que sua interação seja mais rápida (Fig. 4.1). Em situações em que jogadores interagindo entre si estivessem conectados a diferentes servidores implicaria em maior tráfego, pois seria necessário algum tipo de negociação entre os servidores aos quais os diferentes jogadores estão conectados, para que os estados da simulação em ambos fossem idênticos. Além disso seria necessário que cada mensagem entre estes clientes desse mais saltos, passando por mais de um

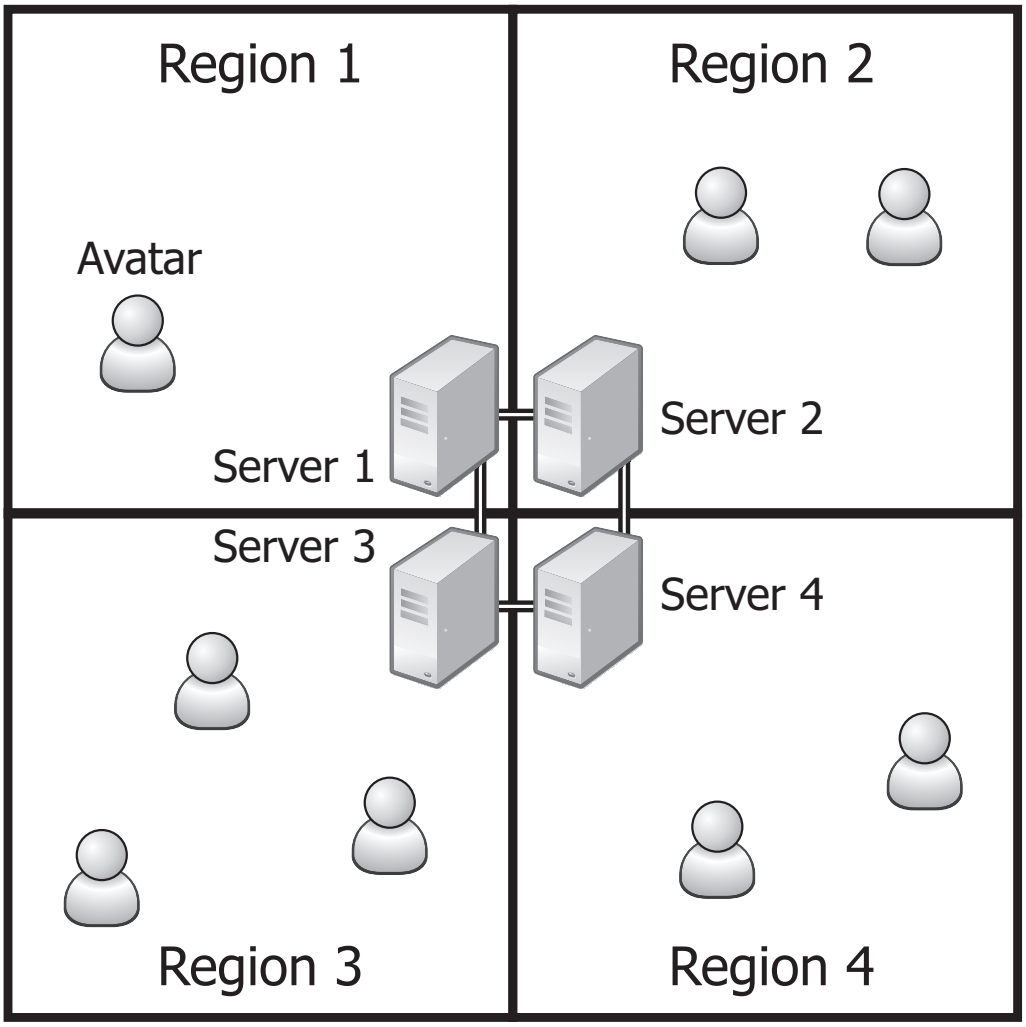

Figura 4.1: Distribution model

intermediário.

Uma questão que diz respeito a esse modelo de particionamento do ambiente virtual está relacionada às fronteiras entre as regiões. Se um avatar de um cliente conectado a um servidor está próximo à fronteira de uma região com outra, que está associada a um outro servidor, será necessário haver troca de informações entre os servidores. Essas informações consistirão em atualizações dos estados das entidades que estão interagindo entre si apesar de estarem situadas em regiões diferentes. Por exemplo, seja S_A o servidor responsável pela região R_A onde está situado o avatar do cliente C_A e S_B o servidor responsável por outra região, R_B, onde está situado o avatar do cliente C_B. Quando o avatar de C_A aproxima-se da fronteira com R_B, S_A envia para S_B uma mensagem alertando a respeito da presença daquele avatar próximo da fronteira. Se o avatar de C_B aproximar-se da fronteira com R_A, S_B também avisa S_A a respeito, e começa a enviar atualizações de estado de C_B para S_A, que então encaminha para C_A, e vice-versa.

No que diz respeito à simulação das ações executadas por jogadores cujos avatares estão situados em regiões diferentes, deve-se decidir como será feita a simulação. Como o foco deste trabalho não é a simulação em si, mas sim a otimização do uso de largura de banda através de um novo algoritmo de gerenciamento de interesse, decidiu-se que a simulação será realizada pelo servidor ao qual o cliente daquele jogador está conectado.

Dessa forma, se o jogador cujo avatar está em R_i executar uma ação próximo à fronteira, envolvendo entidades em R_j, será o servidor S_i quem decidirá o resultado destas ações, repassando a S_j apenas o novo estado já calculado.

Desta maneira, os detalhes deste mecanismo não irão implicar em mudanças relevantes para o gerenciamento de interesse. Quando um jogador J com o avatar próximo à fronteira de determinada região executa ações cujo resultado precisa ser difundido para jogadores com os avatares em outras regiões, o servidor S responsável por J simula suas ações, calcula o estado resultante e simplesmente o envia para o servidor vizinho, como se estivesse enviando para seus próprios clientes. Isso acontece da mesma forma que aconteceria se os outros jogadores também estivessem conectados a S. Analogamente, quando S receber o estado resultante de uma ação de um jogador que está na região vizinha à sua, difunde-o para os jogadores a ele conectados como se um jogador dentro de sua própria região tivesse executado a ação.

4.4 Area of Interest Algorithms

Para que os diferentes jogadores interajam entre si e com as diversas entidades presentes no ambiente do jogo de maneira adequada, é necessário que disponham de réplicas locais destas entidades, cujo estado deve ser o mesmo para todos. A maneira mais simples de fazer isso seria difundir o estado de todas as entidades para todos os jogadores, mas isso geraria uma quantidade alta de tráfego, a depender do número de jogadores participando. Para economizar largura de banda, tanto dos jogadores, quanto dos servidores que os intermediam, é utilizada uma técnica conhecida como gerenciamento de interesse. Esta técnica reduz o número de atualizações que determinado jogador irá receber - e enviar, no caso de uma arquitetura par-a-par.

Em resumo, o gerenciamento de interesse funciona da seguinte forma: para cada mudança de estado de cada entidade, é calculado para quem ela será relevante. Por exemplo, se um avatar situa-se a quilômetros de distância de outro, sem nenhum tipo de vínculo (como grupo, guilda etc.) entre eles, é irrelevante para cada um deles o estado mais recente do outro. Assim, não é necessário que eles troquem suas informações de estado. Este princípio, de localidade, é utilizado como critério principal no gerenciamento de interesse.

Os algoritmos descritos nas próximas seções baseiam-se, entre outras coisas, na distância euclidiana entre cada avatar e todas as outras entidades presentes no ambiente virtual. Isso poderia gerar um problema de escalabilidade, porém está sendo suposta uma arquitetura distribuída, onde tal processamento poderá e deverá ser paralelizado. No modelo de distribuição definido anteriormente, cada servidor controla uma região do mapa. Por conseguinte, cada um deles gerencia apenas um subconjunto das entidades do jogo, verificando somente as distâncias entre cada par delas, além das entidades que estiverem em uma região vizinha, próximos à sua fronteira.

Nas sessões seguintes, serão descritos algumas versões desta técnica, tais como gerenciamento de interesse baseado em área circular e em ângulo de visão do avatar. Na seção 4.5 é introduzida a abordagem de atenuação da freqüência de atualizações, onde será descrita em detalhes o algoritmo proposto.

4.4.1 Área circular

A forma mais simples de executar gerenciamento de interesse consiste em definir uma área em forma de círculo, cujo centro é definido pelas coordenadas da localização

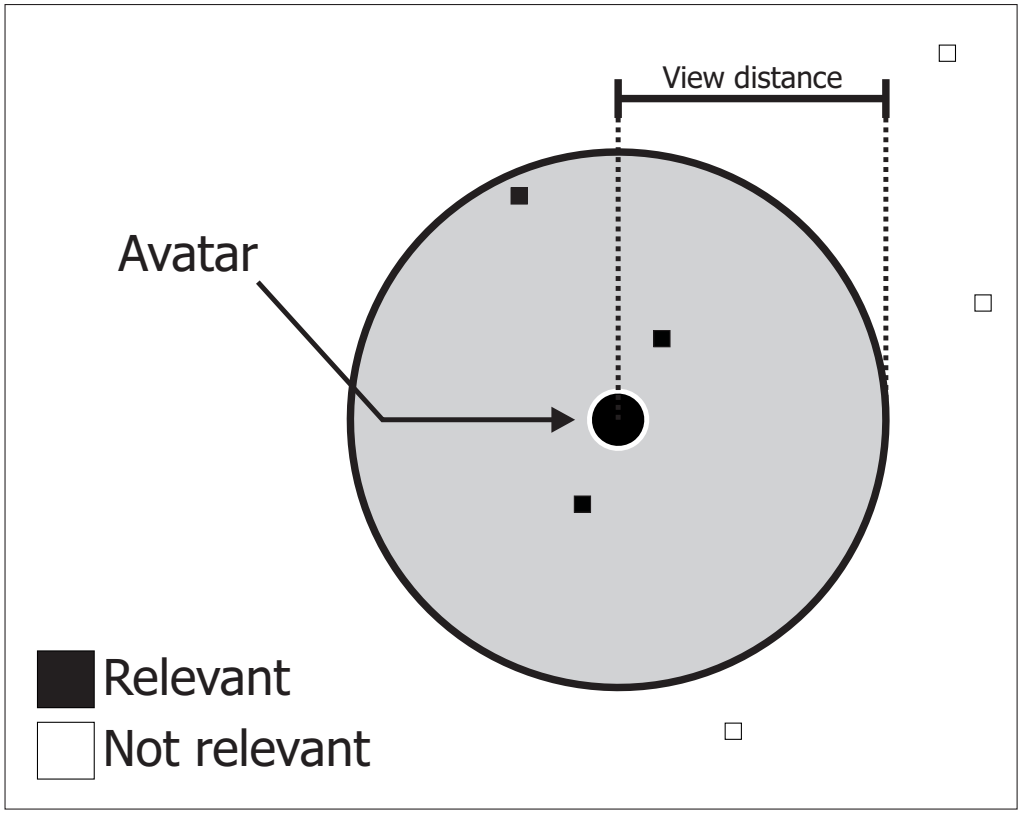

Figura 4.2: Circular area of interest

do avatar no ambiente virtual. Após isso, é calculada a distância euclidiana entre cada avatar e cada uma das outras entidades presentes no mundo do jogo. Seja o avatar A_i, cuja área de interesse é um círculo de raio rad_i. Se o avatar A_j estiver a uma distância menor que rad_i de A_i, então suas atualizações de estado serão relevantes. A_i não receberá atualizações de estado de entidades que estejam a uma distância maior. A figura 4.2 ilustra este tipo de área de interesse.

4.4.2 Ângulo de visão

Outra maneira, um pouco mais refinada, de gerenciar o interesse dos avatares consiste em levar em conta o que o jogador pode visualizar, ou seja, seu ângulo de visão. A área dentro de onde esse jogador perceberá mudanças relevantes pode ser definida como um setor de círculo. É similar à área em formato de círculo definida anteriormente, porém leva em consideração que o jogador só pode visualizar objetos que estão situados à frente de seu avatar.

Uma questão a ser considerada, no entanto, é que o jogador não irá receber atualizações de estado de entidades imediatamente atrás de seu avatar, podendo comprometer o jogo. Se o avatar girar 180° em torno de seu próprio eixo rapidamente, pode ser que não veja determinada entidade que deveria estar ali, necessitando de certo tempo para receber o estado dela. Isto acontece porque, apesar desta entidade ter estado próximo do avatar, ele não recebeu suas informações ainda pois antes ela estava atrás dele, fora do seu campo de visão. Na figura 4.3 é ilustrado como seria uma área de interesse que levaria em consideração o campo de visão do jogador.

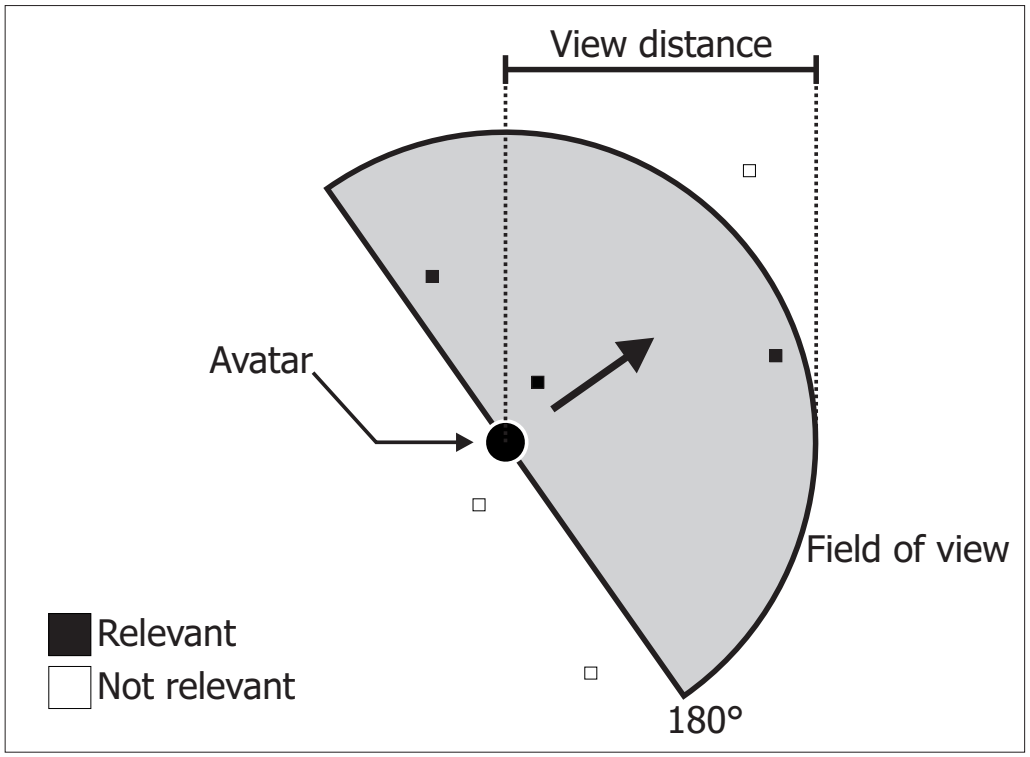

Figura 4.3: Field of view based area of interest

4.5 Graded Area of Interest

O princípio por trás da abordagem proposta aqui baseia-se no fato de que, quanto mais distante uma entidade se situar do avatar no ambiente virtual, menor será a exigência por rapidez nas suas atualizações, para aquele avatar. Sendo assim, pode-se receber atualizações de estado de entidades que estão mais distantes com maior intervalo entre elas. Por outro lado, se uma entidade está muito próxima, é desejável que o jogador disponha de seu estado mais recente assim que possível, para poder visualizar quaisquer mudanças rapidamente.

Para atingir este objetivo, é necessário definir alguns parâmetros:

Relevância - valor real entre 0 e 1, inclusive, que determina o quanto o estado de determinada entidade é relevante para um avatar.

Freqüência de atualização - quantidade de atualizações que cada avatar recebe de cada uma das entidades do ambiente virtual por unidade de tempo.

Intervalo normal de atualização - menor intervalo de tempo entre a chegada de duas atualizações de estado consecutivas de uma mesma entidade em um cliente, ou seja, quando a relevância daquela entidade para o avatar daquele cliente é 1. Assim sendo, o intervalo normal determina a freqüência máxima de atualização.

Alcance de visão - determina a que distância as entidades podem estar do avatar, no máximo, para que o jogador possa visualizá-las.

Distância crítica - é o raio do círculo, em torno do avatar, onde todas as entidades têm relevância igual a 1.

Para se enviar o estado de uma entidade para determinado cliente, verifica-se primeiro quando foi o último envio. O próximo instante de envio é então escalonado para ocorrer após um determinado intervalo de tempo. Se a relevância daquele estado for 1, será utilizado o intervalo normal de atualização. Se for menor que 1, divide-se o intervalo normal

pela relevância. Por exemplo, seja um jogo em que o intervalo normal de atualização seja de 200 ms. Se o avatar A_i, que acabou de enviar uma atualização de estado para A_j, está a uma distância de A_j tal que sua relevância é 0.5, o próximo envio será depois de um intervalo de 200/0.5, ou seja, 400 ms. Apesar deste intervalo ainda ser uma fração de segundo, representa uma diminuição da freqüência de atualização do estado de A_i em 50%. Como estão a uma distância maior um do outro, e o intervalo foi aumentado de apenas 200 ms, esta variação deverá ser imperceptível para o jogador que controla A_j.

É importante perceber que a atenuação da freqüência de atualização das entidades é compatível com outras técnicas mais complexas de gerenciamento de interesse. Em (BOULANGER; KIENZLE; VERBRUGGE, 2006), são descritos diversos algoritmos de gerenciamento de interesse que poderiam ser ainda melhorados se fosse agregada a idéia de diferentes intervalos de envio baseado na relevância destas atualizações. Geralmente o estado de cada entidade é classificado em um de apenas dois extremos: é relevante ou não é relevante, ignorando-se que há uma vasta gama de valores intermediários. A questão está em como definir esse valor de relevância para cada estado. Nas seções seguintes, serão apresentados dois exemplos de algoritmos que definem o método de se obter esse valor, assim como o tipo de área utilizado. Na seção 4.5.2, é especificado o A^3, que é um algoritmo original de gerenciamento de interesse, que, dentre outros princípios, emprega a atenuação da freqüência de atualização. As simulações e seus resultados são apresentados nas seções 4.6 e 4.7, respectivamente.

4.5.1 Círculo com atenuação

Um exemplo simples de utilização de intervalos variados de atualização baseado em relevância seria utilizando a área de interesse em formato de círculo. Para obter-se a relevância de uma entidade em relação a um avatar, pode-se fazer com que seu valor seja 1 quando a entidade estiver na mesma posição do avatar e ir diminuindo gradualmente à medida em que se afasta, até chegar a 0, quando para de diminuir não importa o quanto mais se afaste. Essa é uma maneira que, apesar de simples, demonstrou uma significante redução no tráfego entre clientes e servidores. Na figura 4.4, é ilustrado como seria a área de interesse com atenuação gradual da freqüência de atualização das entidades para um determinado avatar.

4.5.2 Algoritmo A^3

O algoritmo de gerenciamento de interesse proposto neste artigo, denominado A^3 (ângulo de visão com área próxima e atenuação de freqüência de atualização), leva em conta três fatores principais:

- Ângulo de visão do avatar, para determinar quais entidades o jogador tem que ser capaz de perceber imediatamente, por estarem à sua frente, até a distância que seu alcance de visão permita;
- Área próxima, cujo objetivo é melhorar a qualidade do jogo no espaço mais perto do avatar. Seu raio é a distância crítica, definido anteriormente;
- Atenuação da freqüência de atualizações.

A área de interesse resultante então toma a forma de um setor de círculo, cuja origem é o centro de outro círculo, menor. Este círculo menor é a área próxima do avatar do jogador, que receberá atualizações de estado com intervalo normal de entidades que nela

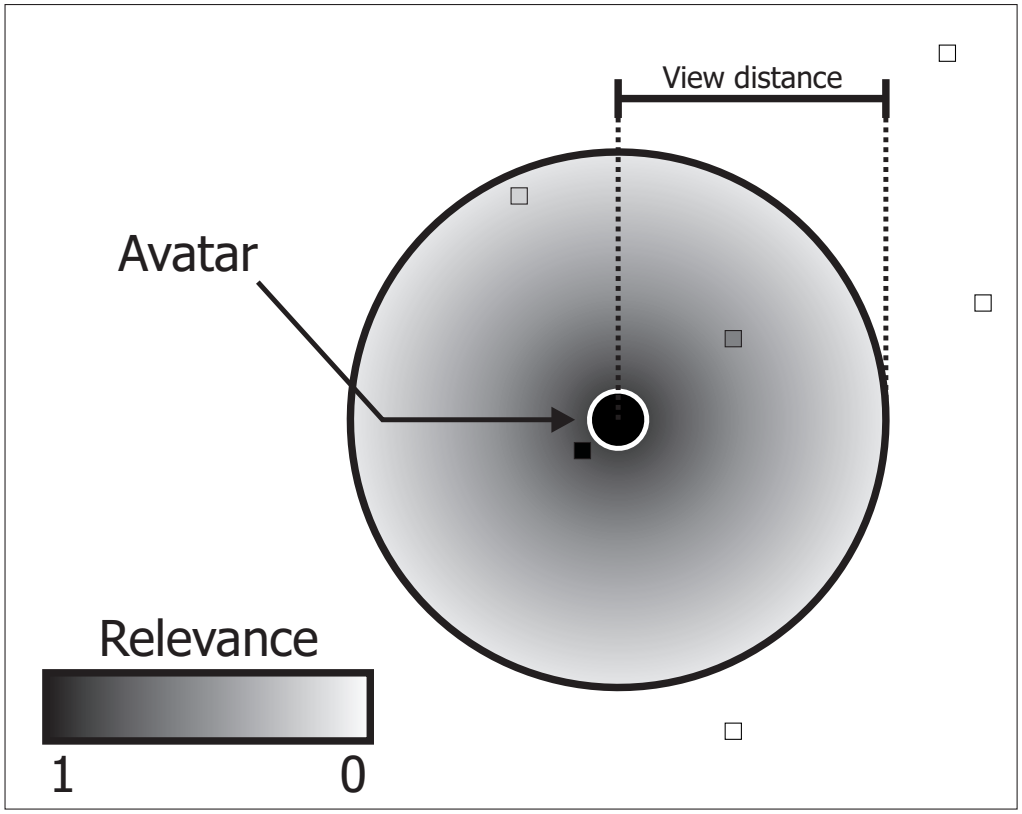

Figura 4.4: Circular area of interest with update frequency atenuation

estiverem. Dessa forma, tem-se o estado mais atualizado possível do jogo na região próxima ao avatar. Isso favorece a interação com entidades que estejam perto dele. Mesmo que alguma delas esteja momentaneamente fora do campo de visão do jogador, ela estará disponível caso ele gire seu avatar repentinamente na direção oposta à que está voltado. Na figura 4.5, é ilustrada a área de interesse que acaba de ser definida.

Quanto às entidades que estiverem fora da área próxima, mas ainda dentro do ângulo de visão, será calculada sua relevância. Propõe-se que a relevância de cada entidade diminua gradualmente de acordo com a distância entre ela e o avatar do jogador em questão. Quanto mais longe, menos freqüentes serão as atualizações de estado. Isso é possível porque mesmo que o intervalo de atualização seja duplicado, ainda será de uma fração de segundo, o que será dificilmente perceptível por um jogador cujo avatar está situado a uma grande distância da entidade em questão. Além disso, pequenos atrasos entre a chegada das atualizações de estado podem ser facilmente mascarados através de técnicas de interpolação, como dead-reckoning (SMED; KAUKORANTA; HAKONEN, 2002). O algoritmo 1 define o funcionamento deste gerenciamento de interesse.

4.6 Simulation

Para efetuar a simulação do algoritmo proposto, foi necessário primeiro criar um modelo de ambiente virtual a simular, com diversos avatares presentes, pois o algoritmo é baseado nas informações de localização e ângulo de visão. O ambiente consiste em um espaço bidimensional, que corresponde à região gerenciada por um dos servidores. Nela, há diversos avatares presentes, cujo número varia de uma simulação para outra. Cada avatar escolhe aleatoriamente um ponto de destino no ambiente e segue até lá. Ao chegar

Algoritmo 1 Calculate relevance of entity E to avatar A

$dist \leftarrow distance(A, E)$
se $dist \leq distância\_crítica$ **então**
  $relevance \leftarrow 1$
senão
  se A can see E in its field of view **então**
    $relevance \leftarrow 1 - \frac{dist - distância\_crítica}{alcance\_da\_visão - distância\_crítica}$
    se $relevance < 0$ **então**
      $relevance \leftarrow 0$
    fim se
  senão
    $relevance \leftarrow 0$
  fim se
fim se

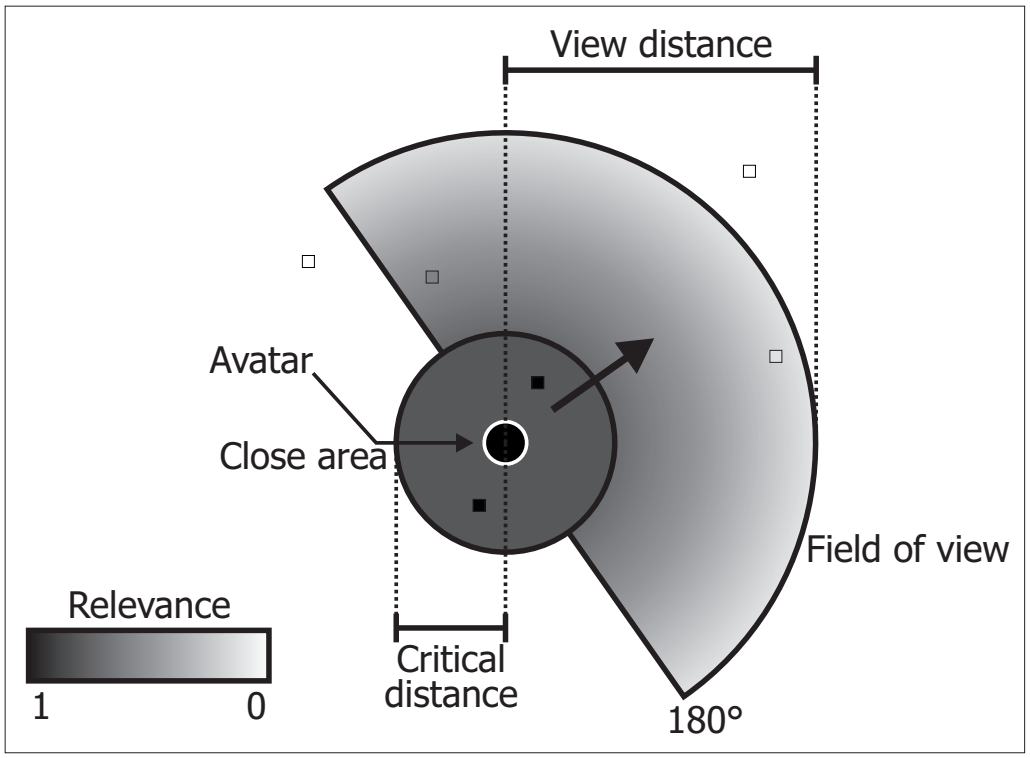

Figura 4.5: A^3 area of interest

no destino, permanece parado por um tempo aleatório, que pode ser zero, e então escolhe uma nova localização para se dirigir.

Foi utilizado o simulador de rede ns-2 (MCCANNE; FLOYD et al., 2006). Este simulador permite criar código específico da aplicação que será simulada. No caso, foi simulado um servidor, que deveria enviar atualizações de estado para um cliente, responsável por um dos avatares na região. Baseado na localização dos outros avatares e no algoritmo de gerenciamento de interesse escolhido, o servidor decidia quais outros avatares tinham um estado relevante para o cliente em questão. Com isso, obtém-se a ocupação de largura de banda de envio necessária para um único cliente. Não se julgou necessário simular simultaneamente todos os clientes conectados àquele servidor, pois todos os

avatares têm o mesmo perfil. Para encontrar a carga total no servidor, basta multiplicar a banda de envio necessária para um cliente pelo número de clientes presentes na região.

Outra questão é que o consumo de largura de banda de envio do servidor é muito maior que o de recepção - se ele recebe n ações, cada uma oriunda de um dos n clientes, é necessário, no pior caso, enviar $O(n^2)$ atualizações de estado, pois cada jogador precisaria do estado de todos os outros. Assim sendo, foi necessário apenas medir a banda de transmissão utilizada.

Em trabalhos como (YU et al., 2007), (KIM et al., 2005) e (SVOBODA; KARNER; RUPP, 2007), é analisado o tráfego de rede gerado por jogos em larga escala. Baseado nestes trabalhos, e adotando uma postura conservadora, foram decididos os seguinte parâmetros para serem utilizados na simulação:

- Intervalo normal de atualização: 250 ms;
- Tamanho do pacote de atualização de estado de uma única entidade: 100 bytes;
- Duração de cada sessão de jogo simulada: 20 min;
- Área do ambiente virtual: 750 x 750 unidades de área;
- Alcance da visão: 120 unidades de comprimento;
- Distância crítica: 40 unidades de comprimento;
- Ângulo de visão: 180°.

Foram executadas diversas simulações, com o objetivo de comparar os algoritmos de gerenciamento de interesse apresentados. O número de avatares presentes no ambiente foi uma das variáves analisadas, para verificar a escalabilidade. Os algoritmos comparados foram os baseados em círculo, círculo com atenuação, ângulo de visão e o algoritmo proposto, A^3. Para demonstrar o quanto cada um destes reduz o tráfego, foram feitas simulações também em que não é empregado nenhum tipo de gerenciamento de interesse, e o servidor envia para o cliente atualizações de estado de todas as outras entidades do jogo.

4.7 Results

Os resultados foram coletados da seguinte maneira: para encontrar a largura de banda utilizada em média para envio, foram somados todos os pacotes de cada sessão e dividido pelo tempo que foi simulado; para determinar a largura de banda máxima utilizada, foi verificado, segundo a segundo, quantos bytes foram enviados e foi selecionado o máximo.

Nas tabelas 4.1 e 4.2, são apresentados os dados coletados de largura de banda máxima e média, respectivamente, utilizada com os quatro algoritmos simuladas - área em círculo (C), área em círculo com atenuação (C & A), área do campo de visão (FoV) e área do campo de visão mais área próxima mais atenuação (A^3) - além de mostrar quanto seria a largura de banda utilizada se nenhuma técnica fosse empregada (None). Os valores estão em bytes/s. Nas figuras 4.6 e 4.7 são mostrados os gráficos correspondentes.

Apenas usando diferentes freqüências de atualização no gerenciamento de interesse baseado em círculo, reduziu-se em 41.59% a largura de banda de envio utilizada em média pelo servidor por cliente. A utilização máxima de largura de banda também foi reduzida,

Tabela 4.1: Largura de banda máxima utilizada

Avatars	None	C	C & A	FoV	A^3
25	9400	8500	5700	7100	4700
50	19300	17000	10300	12300	8100
75	29100	23600	16600	17800	11300
100	38800	32500	20500	23000	15500
125	48600	37400	24300	29500	19700
150	58300	47400	29900	32900	22700
175	67700	56100	34300	32400	21500
200	77600	62300	37500	41200	28900

Tabela 4.2: Largura de banda utilizada em média

Avatars	None	C	C & A	FoV	A^3
25	9221	4715	2759	2534	1700
50	18826	9350	5442	4949	3303
75	28432	13963	8315	7619	5137
100	38037	19324	11029	9928	6739
125	47642	23138	13871	12434	8290
150	57247	29031	16432	15085	10062
175	66853	34697	19661	23060	14250
200	76458	38600	23450	21491	14413

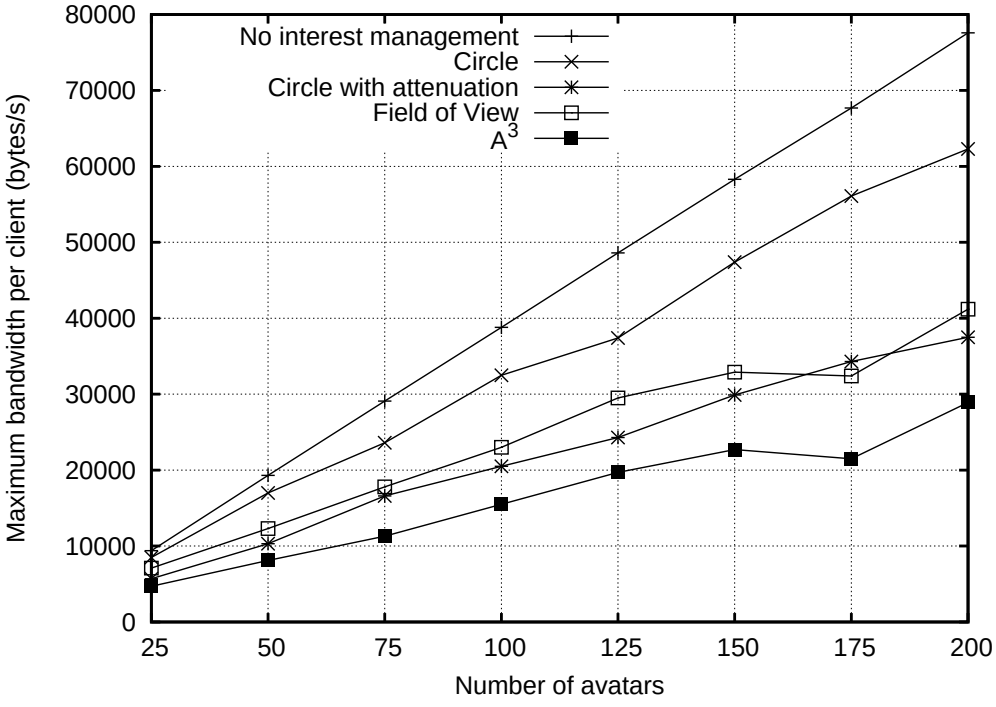

Figura 4.6: Simulation Results: maximum bandwidth usage

em 36.19%. Estes valores representam a média de redução de uso de largura de banda para os diferentes números de clientes.

No que diz respeito ao algoritmo proposto A^3, obteve-se uma redução de uso médio de largura de banda de envio de 63.51% e 33.58%, comparado respectivamente com o

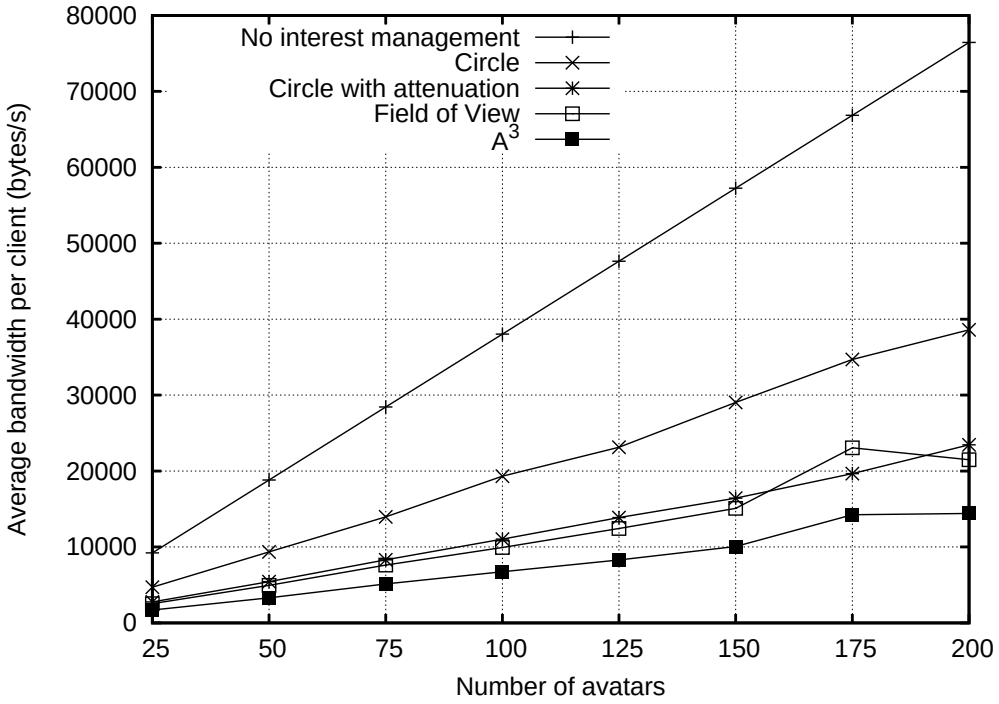

Figura 4.7: Simulation Results: average bandwidth usage

algoritmo de área de interesse circular e baseado em ângulo de visão. Reduziu-se também o pico de utilização em 52.03% e 33.10%, comparado com os mesmos algoritmos. Na tabela 4.3, são mostrados os percentuais médios de economia de largura de banda máxima e média com o algoritmo A^3, em relação aos outros algoritmos apresentados.

Tabela 4.3: Economia de largura de banda com o algoritmo A^3

Utilização	None	C	C & A	FoV
Máxima	60.10%	52.03%	24.81%	33.10%
Média	81.64%	63.51%	37.48%	33.58%

Observou-se também que os valores médio e máximo observados diferem, mesmo quando não é utilizado nenhum algoritmo de gerenciamento de interesse, ou seja, o cliente recebe atualizações de estado de todas as entidades presentes no jogo, com a freqüência normal. Além disso, com 200 avatares no ambiente, com estado de 100 bytes, cuja atualização é enviada a cada 250 ms, o servidor deveria alocar $199 \times 100 \times 4$ bytes/s para cada cliente, ou seja, 79600 bytes/s. No entanto, observou-se que a utilização máxima e média, com 200 avatares presentes e nenhum gerenciamento de interesse, foi de 77600 e 76458, respectivamente. Isso acontece porque o ns-2 é um simulador de eventos discreto, e o servidor simulado foi programado para checar o schedule de envios a cada 10 ms. Em conseqüência disto, cada atualização de estado pode ter tido seu intervalo aumentado em até 10 ms, o que explica os valores encontrados.

4.8 Conclusion

Foi apresentado um algoritmo de gerenciamento de interesse, o A^3, cuja idéia principal é adaptar a freqüência de atualização de estado das entidades do jogo de acordo com

sua relevância para o cliente que receberá as atualizações. O formato da área de interesse utilizada pelo algoritmo A^3 consiste em um setor de círculo, correspondente ao campo de visão do jogador, mais um círculo de raio menor, que corresponde à área próxima ao avatar daquele jogador. O objetivo deste círculo menor é o de manter o estado naquela região, que é considerada crítica, o mais atualizado possível. Somando-se essas características, chegamos a um algoritmo que obteve redução da utilização máxima da banda de envio do servidor de 52.03% e 33.10%, comparados com o gerenciamento de interesse baseado em círculo e em campo de visão, respectivamente, e de 63.51% e 33.58% de utilização média, comparados com os mesmos algoritmos.

5 BALANCEAMENTO DE CARGA

A principal característica dos jogos maciçamente multijogador é a grande quantidade de jogadores, chegando a ter dezenas ou centenas de milhares de participantes simultaneamente [TODO:ref]. Essa grande quantidade de jogadores interagindo entre si gera um tráfego na rede de suporte que tem crescimento quadrático em relação ao número de jogadores, no pior caso [TODO:referenciar/provar].

Quando se utiliza uma arquitetura cliente-servidor, é necessário que o servidor intermedie a comunicação entre cada par de jogadores – supondo que se pretende prover ao jogo garantias de consistência e resistência a trapaça. Obviamente, esse servidor terá uma grande carga de comunicação e, consequentemente, deverá ter recursos (largura de banda disponível) proporcional à demanda do jogo.

[TODO:largura de banda X CPU]

A questão posta aqui é que, quando se faz uso de um servidor distribuído, principalmente com recursos escassos, que é a proposta deste trabalho, é necessário otimizar o uso destes recursos, atribuindo a cada servidor uma carga que ele seja capaz de suportar. Dessa forma, não importando a qual servidor cada jogador estiver conectado, sua experiência de jogo será semelhante, no que diz respeito ao tempo de resposta para suas ações e o tempo que leva para ser notificado de ações de outros jogadores, assim como de mudanças de estado no ambiente virtual do jogo.

[aqui já fala do n ao quadrado, por cima]

Uma idéia inicial poderia ser a de distribuir os jogadores entre servidores, de maneira que o número de jogadores em cada servidor fosse proporcional à largura de banda daquele servidor. No entanto, essa distribuição não funcionaria, pelo fato de que a carga causada pelos jogadores depende também do quanto os jogadores estão interagindo entre si. Por exemplo, se os avatares de dois jogadores estiverem muito distantes um do outro, provavelmente não haverá interação entre eles e, portanto, o servidor precisará apenas atualizar cada um a respeito de suas próprias ações. No entanto, se estes avatares estiverem próximos, cada jogador deverá ser atualizado não apenas a respeito de suas próprias ações, como também das ações do outro jogador.

Percebe-se, então, que quando os avatares estão distantes uns dos outros, o tráfego cresce linearmente com o número de jogadores. Porém, se eles estão próximos uns dos outros, o tráfego cresce quadraticamente. Por fim, ambas as funções de crescimento do número de mensagens podem estar presentes no mesmo jogo se, em alguns lugares do ambiente virtual, os avatares estiverem próximos e, em outros lugares, eles estiverem distantes.

Essa característica de localidade, no que diz respeito à distribuição dos avatares no ambiente virtual, está presente na grande maioria dos jogos multijogador. Existem alguma exceções, como GuildWars [TODO:ref], em que apenas grupos com um número

limitado de jogadores podem iniciar uma partida. Este tipo de jogo é baseado no modelo de instâncias, onde todos os avatares dos jogadores se encontram em um espaço social, de interação limitada, menos dinâmica e, portanto, com tráfego de rede algumas ordens de grandeza menor [TODO:ref/prove]. Quando pretende-se iniciar uma partida "real", os jogadores requisitam ao servidor que seja criado um grupo de ação. Dessa forma, impede-se que um número teoricamente ilimitado de jogadores interajam entre si, sobrecarregando o servidor.

Normalmente, porém, os jogadores podem mover seus avatares livremente através do mundo do jogo. Isso torna possível a formação de pontos de interesse – também conhecidos como *hotspots* [TODO:ref] – ao redor dos quais os jogadores se concentram mais do que em outras regiões do ambiente virtual. Aliás, muitos jogos de RPG online maciçamente multijogador não só permitem como também estimulam, até certo ponto, a formação destes pontos de interesse. Nestes mundos dos MMORPGs, existem cidades inteiras, onde os jogadores se encontram para conversar, trocar mercadorias virtuais do jogo e/ou duelar, assim como existem também zonas desérticas, sem muitos atrativos para os jogadores, e onde o número de avatares presentes é relativamente pequeno, se comparado com outros lugares no jogo.

[TODO:screenshot(s)]

Por esta razão, não é suficiente apenas dividir os jogadores entre os servidores, mesmo que proporcionalmente aos recursos de cada um destes. Em primeiro lugar, em alguns casos o uso de largura de banda do servidor é quadrático ao número de jogadores, enquanto é linear em outros. Essa razão por si só já é suficiente para buscar outro critério para o balanceamento de carga. Além disso, surge outra questão importante: a existência de pontos de interesse. Esta última característica motiva à criação de um esquema de balanceamento de carga para jogos que impeça que a presença de hotspots degrade a qualidade do jogo além do tolerável.

5.1 Trabalhos relacionados

Como foi dito, quando existe um número considerável de avatares em um mesmo ponto de interesse, é gerado um tráfego proporcional ao quadrado do número de avatares ali presentes. Também foi mostrado que os servidores recebem as ações enviadas pelos jogadores, calculam seu resultado e o enviam para todos os jogadores interessados, que são, geralmente, aqueles cujos avatares estiverem próximos do avatar do primeiro jogador. Se esses jogadores forem divididos entre diferentes servidores, cada um destes precisará não apenas enviar o estado do mundo resultante das ações para os jogadores conectados a ele, como também deverá enviá-lo para o servidor ao qual os outros jogadores estão conectados. Este, por sua vez, encaminhará este resultado para seus jogadores.

Percebe-se, então, que cada estado deverá ser enviado duas vezes, para cada par de jogadores que se comunicam através de servidores diferentes (Figura 5.1). Esse overhead não apenas causa o desperdício de recursos dos servidores, como também aumenta o atraso para atualização de estado das réplicas do jogo nas máquinas dos jogadores. Isto faz com que o tempo entre o envio de uma ação por um jogador conectado a um servidor e o recebimento do estado resultante por outro jogador, conectado a outro servidor, seja maior, prejudicando a interação entre eles.

Assim sendo, jogadores que estão interagindo entre si devem, idealmente, estar conectados ao mesmo servidor. Contudo, é possível que todos os jogadores estejam ligados entre si através de relações transitivas de interação. Por exemplo, dois avatares, de dois

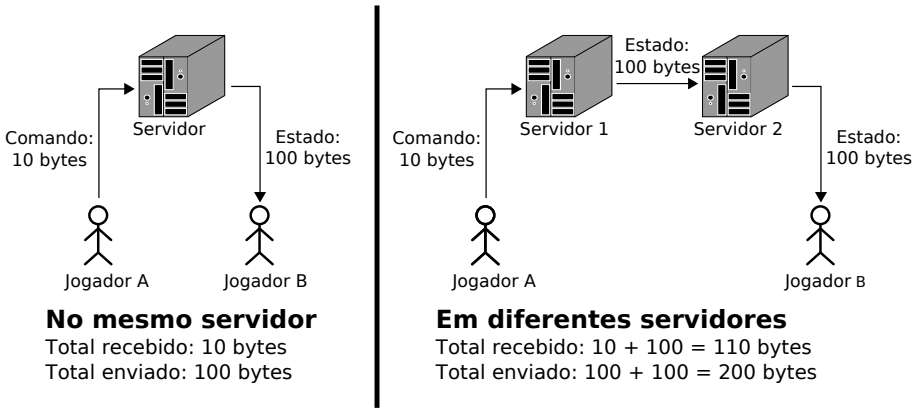

Figura 5.1: Overhead causado pela interação de jogadores em diferentes servidores

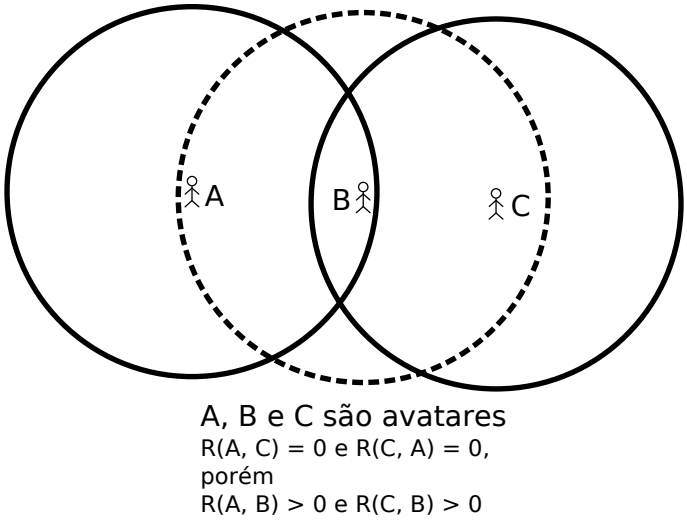

Figura 5.2: Dependência transitiva entre os avatares

jogadores diferentes, podem estar distantes, porém ambos interagindo com um terceiro avatar, entre os dois. A Figura 5.2 ilustra melhor este tipo de situação. Contudo, ainda assim será necessário dividí-los entre servidores. A questão é quanto a quais pares de jogadores estarão divididos em servidores diferentes. É necessário, portanto, decidir um critério para agrupar jogadores em um mesmo servidor.

Nas próximas seções, serão apresentados os trabalhos e princípios utilizados nas abordagens de outros autores. Embora apontem direções que se podem tomar para resolver o problema de distribuição e balanceamento de carga dinâmico de MMOGs, não o resolvem por completo, por razões que serão melhor detalhadas ao longo do texto. Com base em alguns desses princípios a seguir, e em pesquisa e implementação realizadas, será definido o esquema de balanceamento de carga aqui proposto.

5.1.1 Microcélulas e macrocélulas

Uma maneira de fazer uso da localidade dos jogadores é agrupá-los de acordo com a posição ocupada por seus avatares no ambiente virtual. A questão seria a de como formar estes grupos. Uma maneira de fazer isto seria dividindo o mundo do jogo em várias células conectadas entre si. Cada célula consistiria em uma parte do mundo, com conteúdo e características próprios, que seria delegada a um nodo servidor. A forma mais

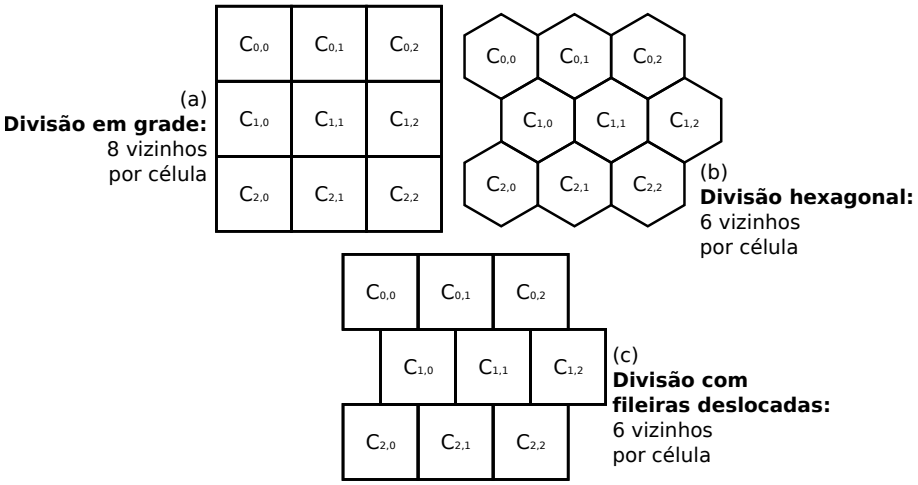

Figura 5.3: Diferentes tipos de divisão em células

simples de fazer isto é com uma grade de células de mesmo tamanho e formato.

Porém, o formato e a disposição destas células irá influenciar no tráfego gerado pelas mesmas entre os servidores. Por exemplo, um ambiente bidimensional poderia ser dividido em uma grade de células quadradas (Figura 5.3(a)). Neste caso, cada uma destas células teria oito vizinhos, em média – células nas bordas do mapa poderiam ter cinco ou três vizinhos apenas. Um servidor é considerado **vizinho** de outro quando administra uma célula que é vizinha a uma célula do outro servidor. Quanto mais servidores vizinhos, maior o tráfego entre servidores, e maior o overhead causado por esta comunicação. A distribuição ideal, então, seria utilizando células hexagonais, cada uma com seis vizinhos (Figura 5.3(b)). Estudos comprovam que esta é a divisão em células iguais que permite o menor número de vizinhos por célula [TODO:ref]. Outra possibilidade seria utilizando fileiras alternadas de células quadradas, onde cada fileira seria deslocada o equivalente à metade do comprimento de uma célula (Figura 5.3(c)).

Uma questão importante deste design é que o conceito de célula é transparente para os jogadores. Estes visualizam um mundo vasto, único e não fragmentado, mesmo que estejam cruzando repetidamente as fronteiras entre diferentes células. Assim, é possível para eles moverem-se livremente através do ambiente virtual, independente de como é feita a distribuição. Obviamente, isso exige que as células se comuniquem, de maneira a atualizarem-se mutuamente e notificarem-se a respeito de eventos que ocorram próximo à fronteira entre elas, assim como a respeito da migração de jogadores entre uma e outra.

Embora essa abordagem com células distribua a carga entre diversos servidores, não há garantias de que essa distribuição será uniforme, devido à grande mobilidade dos jogadores e à existência de pontos de interesse. Uma das idéias propostas na literatura (DE VLEESCHAUWER et al., 2005) também segue o princípio de dividir o ambiente virtual em células de tamanho e posição fixos, porém essas células são relativamente pequenas – ou **microcélulas** – e podem ser agrupadas, formando um espaço contínuo chamado de **macrocélula**. Cada macrocélula é então designada a um diferente servidor, que passa a administrar não apenas uma grande célula de tamanho e posição fixos, mas um conjunto variável de pequenas células. Estas microcélulas podem então ser transferidas dinamicamente entre diferentes macrocélulas, de maneira a manter a carga em cada um dos diferentes servidores abaixo do limite por ele suportado.

Obviamente, as microcélulas designadas ao mesmo nodo servidor não gerarão tráfego adicional para sincronizarem-se entre si, porém não se poderá conhecer previamente o

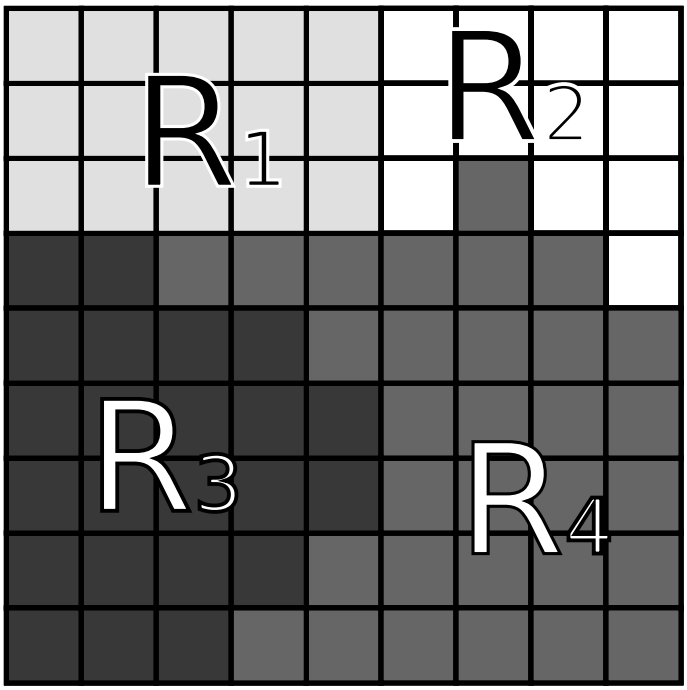

Figura 5.4: Microcélulas agrupadas em quatro macrocélulas (R_1, R_2, R_3 e R_4)

overhead de sincronização entre diferentes macrocélulas, pois o número de vizinhos que cada macrocélula terá é imprevisível, assim como o seu formato. Contudo, demonstrou-se (DE VLEESCHAUWER et al., 2005) que esse overhead é compensado pela melhor distribuição da carga do jogo entre os servidores. A Figura 5.4 ilustra a divisão de um ambiente virtual bidimensional em microcélulas e o agrupamento destas em macrocélulas dinâmicas, que podem se adaptar à distribuição de avatares.

5.1.2 Balanceamento com informações locais

No trabalho de (LEE; LEE, 2003), também é proposto um esquema dinâmico de balanceamento de carga para os servidores de um sistema multi-servidor de ambiente virtual, levando em conta que os usuários podem estar distribuídos através deste mundo de maneira não uniforme. De acordo com o esquema proposto pelos autores, um servidor sobrecarregado inicia o processo selecionando um conjunto de outros servidores para fazerem parte da redistribuição de carga. O conjunto de servidores selecionados dependerá do nível de sobrecarga do servidor inicial, assim como da quantidade de recursos ociosos dos outros servidores. Após a formação desse conjunto, seus elementos repartirão as porções do ambiente virtual que lhes pertencem utilizando um algoritmo de particionamento de grafo, de forma que os servidores envolvidos tenham carga final de trabalho semelhante.

O principal aspecto da solução proposta pelos autores foi a utilização de informações locais (do servidor que iniciou o processo de balanceamento e de seus vizinhos), ao invés de informações globais (todos os servidores do sistema se envolveriam no balanceamento). A primeira apresenta pouco overhead, mas pode não resolver o problema de maneira eficiente em poucos passos, já que servidores sobrecarregados tendem a estar adjacentes. Já a abordagem global é capaz de dividir a carga de trabalho da forma mais equilibrada possível, mas sua complexidade cresce rapidamente com o aumento do número de servidores envolvidos. A solução apontada no trabalho então é de o balan-

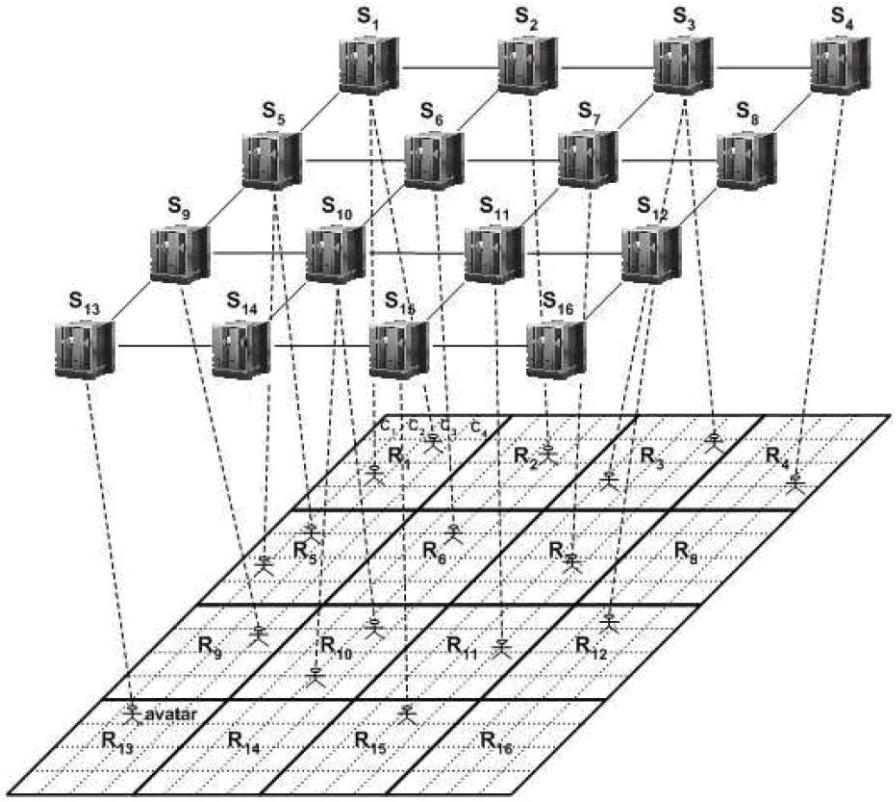

Figura 5.5: Um modelo multi-servidor para ambiente virtual distribuído

ceamento de carga envolver apenas um subconjunto de servidores, sendo que sua cardinalidade varia de acordo com a necessidade (se os vizinhos do servidor que disparou o balanceamento de carga estiverem também sobrecarregados, são selecionados mais servidores). Dessa forma, tem-se um pouco mais de informação do que a abordagem local, mas sem o problema da complexidade inerente à abordagem global.

Para atacar o problema, os autores também subdividem o ambiente virtual em **células** – semelhantes às microcélulas – retangulares, sendo que o número de servidores é muito menor que o número de células. As células são agrupadas em **regiões** – ou macrocélulas – e cada região é gerenciada por um servidor. Cada servidor mantém atualizadas as informações de estado dos usuários e lida com as interações entre avatares na região a ele dedicada. Cada usuário envia e recebe atualizações de estado através do servidor que gerencia a região na qual ele está jogando. Duas células são ditas adjacentes (ou vizinhas) se elas compartilharem uma fronteira. Analogamente, duas regiões, e seus respectivos servidores, são ditos adjacentes se existir um par de células adjacentes, cada uma das quais pertencendo a uma das duas regiões. A Figura 5.5 ilustra o modelo de um sistema multi-servidor consistindo de 16 servidores. O ambiente virtual é dividido em 256 células, que são agrupadas em 16 regiões.

Foi definida a carga de trabalho de uma célula como o número de avatares presentes naquela célula. Os autores assumiram que todos avatares atualizam seus estados na mesma frequência, de forma que a carga de processamento (computação e comunicação) que uma célula impõe a um servidor é proporcional ao número de usuários naquela célula. A carga de trabalho de uma região e seu servidor designado é definida como a soma das cargas de trabalho individuais das células que compõem aquela região. Cada servidor periodicamente avalia sua carga de trabalho e troca informações de carga com os servido-

res vizinhos. Assumiu-se, também, que estes servidores estão conectados através de uma rede de alta velocidade. Dessa forma, o overhead de trocar informações de sobrecarga entre vizinhos é limitada e considerada negligenciável, se comparada com outros custos da distribuição de carga. Pelo mesmo motivo, também assumiu-se como negligenciável o overhead de comunicação entre servidores quando jogadores em diferentes regiões estão interagindo.

5.1.2.1 Seleção de grupo local para balancear a carga

Um servidor dispara o balanceamento quando a carga atribuída a ele excede sua capacidade. Este servidor seleciona um conjunto de outros servidores para se envolverem com a distribuição. Primeiro, o servidor que iniciou escolhe o menos carregado dentre seus vizinhos e envia um pedido de que ele participe do balanceamento de carga. O vizinho escolhido rejeita o pedido se ele já está envolvido em outro grupo de balanceamento; caso contrário, ele responde ao servidor iniciador com a informação de carga de seus próprios vizinhos. Se o servidor vizinho que está participando não for capaz de absorver a carga de trabalho excedente do servidor iniciador, a seleção é executada novamente entre os servidores vizinhos de não apenas o servidor sobrecarregado, como também os vizinhos do vizinho escolhido na primeira fase. A seleção continua até que a carga de trabalho excedente do primeiro servidor possa ser absorvida – isto é, a carga de trabalho de todos os servidores selecionados torna-se menor que um limite pré-definido. Os autores definiram este limite como sendo 90% da capacidade total do conjunto de servidores. O critério utilizado para esta escolha foi o de evitar o imediato reinício da balanceamento de carga.

Sejam SELECIONADOS e CANDIDATOS dois conjuntos de servidores, ambos inicialmente vazios. Seja P a capacidade de cada servidor. O procedimento para determinar quem serão os servidores envolvidos é o seguinte:

1. O servidor que disparou o balanceamento, S_i, é inserido em SELECIONADOS, que são os servidores envolvidos na distribuição de carga. Os servidores vizinhos do iniciador são adicionados em CANDIDATOS, que são os servidores que podem vir a participar da seleção.

2. De CANDIDATOS, é selecionado o servidor com menor carga de trabalho, S_v; então, S_i envia um pedido a ele para participar na distribuição de carga

   (a) Se o servidor S_v não está envolvido em outra distribuição de carga, ele responde ao servidor S_i com a carga de trabalho de seus vizinhos. Quando S_i recebe esta resposta, ele insere S_v em SELECIONADOS e seus vizinhos são inseridos em CANDIDATOS, se eles já não estiverem dentro de SELECIONADOS ou de CANDIDATOS.

   (b) Se S_v já está participando de outra distribuição de carga, ele rejeita o pedido e é removido do conjunto CANDIDATOS.

3. O passo 2 é repetido até que a carga de trabalho média dos servidores selecionados se torne menor que um limite: $0,9 \times P$.

Para exemplificar o funcionamento do algoritmo, pode-se observar a Figura 5.6. Todos os servidores têm a mesma capacidade, podendo cada um comportar 100 usuários. Primeiro, o servidor iniciador, S_6, é inserido em SELECIONADOS e seus vizinhos (S_2, S_5, S_7 e S_{10}) são adicionados a CANDIDATOS (Figura 5.6(a)). Então, S_7, que tem a

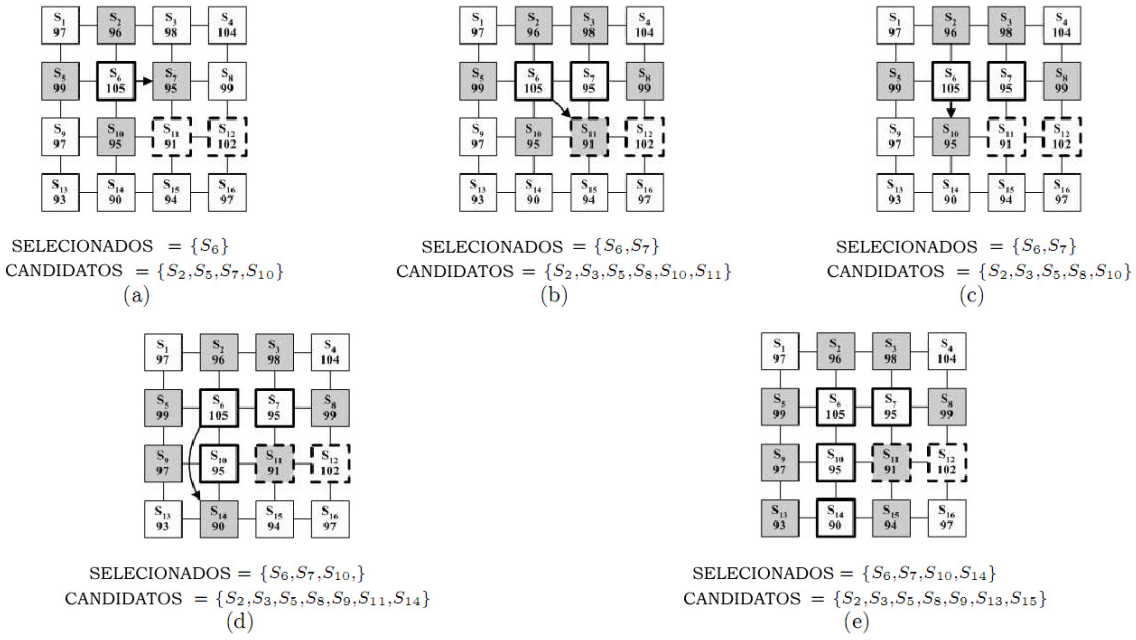

Figura 5.6: Seleção do grupo de servidores para balanceamento local

menor carga de trabalho dentre os servidore em CANDIDATOS, é selecionado e convidado a participar da distribuição de carga. Quando S_7 responde a S_6 com a informação de carga de seus vizinhos (S_3, S_6, S_8 e S_{11}), S_7 é inserido em SELECIONADOS e seus vizinhos, exceto S_6, são adicionados a CANDIDATOS (Figura 5.6(b)). Agora, S_{11}, que tem a menor carga de trabalho dentre os servidores em CANDIDATOS, é selecionado e convidado a participar da distribuição de carga. Porém, S_{11} rejeita o convite, pois já está envolvido em outra distribuição, iniciada por S_{12}. Assim, S_{11} é removido de CANDIDATOS e S_{10} é selecionado porque tem agora a menor carga de trabalho dentre os servidores em CANDIDATOS (Figura 5.6(c)). Até que a carga de trabalho média dos servidores em SELECIONADOS se torne menor que $0,9 \times P$, ou seja, 90, o procedimento acima continua (Figura 5.6(d) e Figura 5.6(e)).

5.1.2.2 Reparticionamento das regiões

Uma vez que o servidor iniciador seleciona um conjunto de servidores para se envolverem na distribuição de carga, ele reparte as regiões a ele dedicadas com os servidores envolvidos. É sugerido pelos autores, embora não sejam dados detalhes, o uso de alguma técnica de particionamento de grafos para repartir essas regiões. Cada célula seria representada por um vértice adjacente àqueles que correspondessem às células vizinhas. O peso de cada vértice é então ajustado como a carga de trabalho da célula que representa. Eles são agrupados em partições, cada uma das quais representa uma região, e o número de partições deve ser igual ao número de elementos do conjunto SELECIONADOS. Cada uma das partições formadas após a execução do algoritmo de particionamento de grafo deverá ter um conjunto de vértices tal que o peso das partições seja semelhante – ou seja, cada servidor terá aproximadamente a mesma carga de trabalho.

Com o término do reparticionamento das regiões, o servidor que iniciou todo o processo de balanceamento de carga dissemina o novo particionamento, que acabou de ser calculado, para os outros servidores envolvidos. Isso inclue a informação de que células devem migrar para quais regiões/servidores. Após receber estas informações, cada servi-

dor inicia o processo de emigração de células e usuários. Primeiro, envia-se o estado dos avatares e das células que serão transferidos para o servidor que os receberá. A informação a ser enviada varia de acordo com as características da aplicação. Por exemplo, pode incluir apenas a localização dos avatares, ou pode conter informações mais detalhadas, como qual é o modelo visual que representa aquele jogador. Depois de enviar esses dados, o servidor notifica cada usuário que migrou, que passa então a comunicar-se apenas com seu novo servidor.

5.1.3 Uso de grafos em distribuição de tarefas

Um problema clássico de alocação de tarefas em sistemas distribuídos é o da dependência entre tarefas [TODO:refsDEdistGRAPH]. Tarefas dependentes entre si fazem com que os processadores nos quais elas estão sendo executadas tenham que se comunicar para que o processamento possa continuar. Isso gera dois problemas principais: em primeiro lugar, o processamento como um todo é atrasado por causa do tempo de espera de cada transmissão e recebimento de mensagens – supondo que a comunicação entre os processos seja por meio de mensagens, como é o caso do MPI [TODO:ref] – e, em segundo lugar, banda de comunicação é ocupada para o envio e recebimento destas mensagens.

Para resolver este problema, são utilizados grafos, da seguinte maneira: o conjunto de tarefas a realizar é mapeado em um grafo com pesos. Cada vértice representa uma tarefa e cada aresta representa a comunicação entre as tarefas. O peso de cada vértice representa o custo de processamento e o peso da aresta representa a carga de comunicação. Para fazer a distribuição, é feito um particionamento do grafo gerado. Cada partição terá um conjunto de tarefas que serão executadas no mesmo nodo e, conseqüentemente, o atraso de comunicação entre estas tarefas será pequeno [TODO:refbli]. No entanto, as arestas do grafo que ligam partições diferentes representam comunicação entre diferentes nodos. A Figura 5.7 ilustra essa situação.

Os algoritmos de particionamento do grafo de tarefas, geralmente, buscam atingir dois objetivos: gerar partições de peso aproximadamente igual, ou seja, cada nodo terá uma carga computacional semelhante à dos outros; e minimizar o corte de aresta, ou seja, fazer com que a dependência entre os conjuntos de tarefas seja a menor possível, reduzindo a comunicação entre os nodos de processamento. Por corte de aresta, entende-se o somatório dos pesos de todas as arestas que ligam vértices que estão em partições diferentes (Figura [TODO:reffig]).

O problema de particionamento de grafos é NP-completo [TODO:ref]. Contudo, já foi feita bastante pesquisa nessa área, resultando em heurísticas que obtêm boas soluções, muitas vezes se aproximando de valores ótimos [TODO:ref]. Uma das heurísticas mais conhecidas é a de Kernighan e Lin (KERNIGHAN; LIN, 1970), que tem como objetivo, partindo de duas partições iniciais quaisquer, chegar a duas partições com peso semelhante e corte de aresta reduzido. O algoritmo de Fiduccia e Mattheyses (FIDUCCIA; MATTHEYSES, 1982) incrementa a solução de Kernighan e Lin, generalizando-a para hipergrafos, além de permitir que cada partição tenha uma fração diferente do peso total do hipergrafo.

Existem outros algoritmos mais recentes, como os baseados em divisão espectral [TODO:ref-33-e-20-karypisirreggraphs]. Estes são computacionalmente caros, por envolver complexos cálculos de álgebra linear, precisando do autovetor associado ao menor autovalor da matrix laplaciana associada ao grafo [TODO:ref]. No entanto, o particionamento resultante é considerado excelente para uma vasta gama de problemas [TODO:ref]. Existe também a abordagem em vários níveis [TODO:citemultilevel], através da simpli-

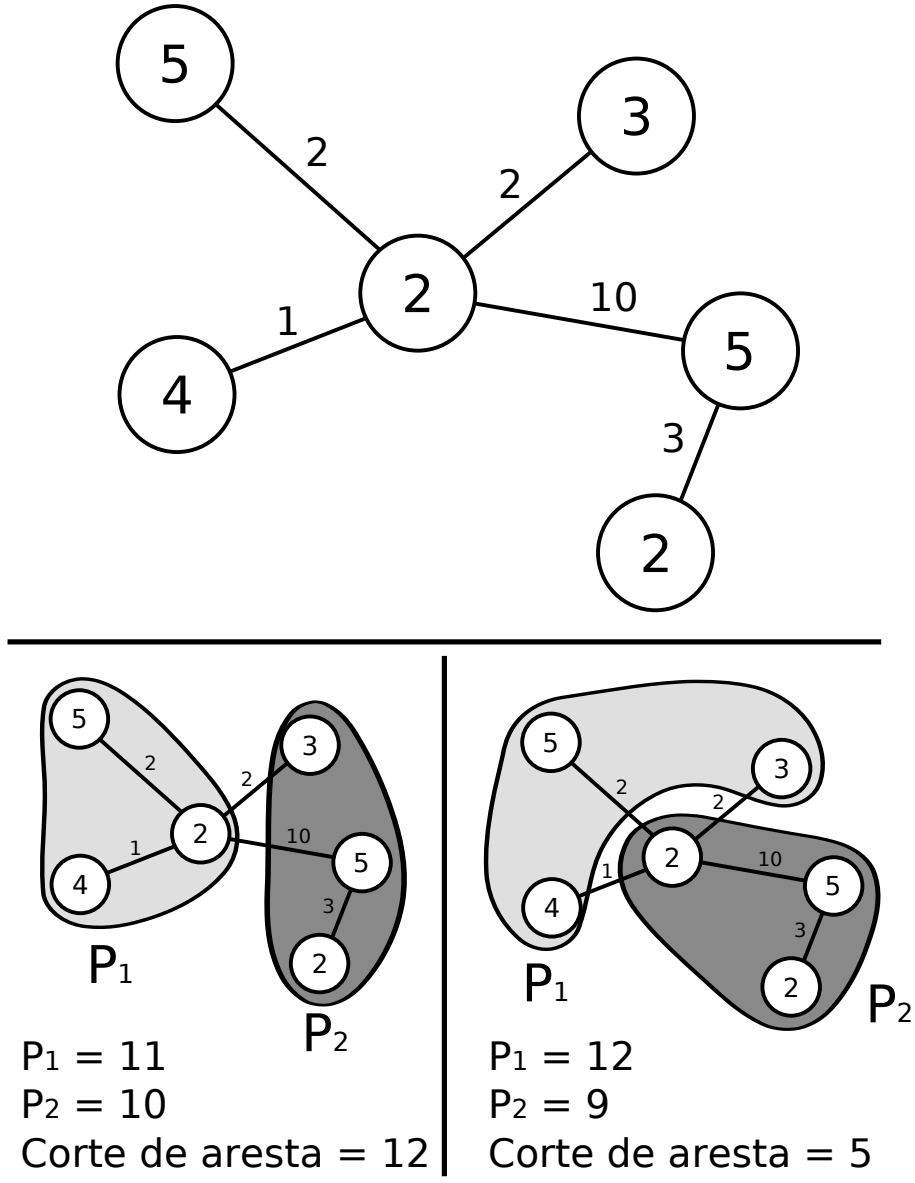

Figura 5.7: Divisão de tarefas utilizando grafos

fação do grafo por meio de contração de arestas. Com o grafo mais simples, é aplicado algum algoritmo de particionamente e então ele se desdobra, já particionado, até o grafo original. Eventualmente, refinamentos no particionamento podem ser necessários. Esta última técnica é especialmente útil em grafos com grande número de vértices, na ordem de centenas a milhares, ou mais. Na figura [TODO:ref] é ilustrado esse particionamento em vários níveis.

[TODO:figura do multilevel]

Contudo, MMOGs são geralmente aplicações de tempo-real, além de a carga imposta sobre os servidores mudar constantemente em tempo de execução. Por esse motivo, o algoritmo de particionamento utilizado deve ser rápido, tornando inviável o uso da divisão espectral. É proposto neste trabalho utilizar um algoritmo guloso, mais simples, baseado em [TODO:aOUTRArefdekarypis], que pode não atingir soluções tão boas quanto as da divisão espectral, porém é consideravelmente mais rápido. Além disso, o custo para encontrar uma solução muito próxima do ótimo não se justifica no contexto de MMOGs, pelo fato dos pesos mudarem constantemente, precisando de um novo particionamento em relativamente pouco tempo. Já quanto ao particionamento em vários níveis, seu uso não foi considerado necessário pois o número de vértices do grafo que representará o ambiente virtual é relativamente pequeno.

Nas próximas seções, será apresentada a solução proposta neste trabalho para balanceamento de carga, começando pelas definições e parâmetros utilizados, assim como será descrito como o problema foi mapeado para grafos e quais os objetivos e critérios do esquema proposto.

5.2 Esquema proposto

Na seção anterior foram apresentados trabalhos existentes no que se refere a balanceamento de carga em MMOGs que utilizam vários servidores para prover o suporte de rede. O esquema de balanceamento de carga proposto neste trabalho tem como base alguns dos princípos já existentes na literatura. Um deles é o da divisão do ambiente virtual em microcélulas, para posterior agrupamento em macrocélulas. Isto permite tratar de maneira relativamente simples a questão da dinâmica da movimentação dos avatares através do mundo do jogo, através da formação dos conjuntos de microcélulas, cujos elementos podem ser transferidos dinamicamente de acordo com a necessidade.

Além disso, também será utilizada a idéia de fazer o balanceamento baseado apenas em informações locais – cada servidor, quando precisar diminuir a carga atribuída a ele, seleciona somente alguns outros servidores para participar de um rebalanceamento de carga local. Dessa forma, pode-se reduzir consideravelmente a complexidade do balanceamento, pois não será necessário que todos os servidores do jogo troquem mensagens entre si cada vez que qualquer um deles estiver desbalanceado.

Os trabalhos relacionados que foram descritos na seção anterior buscam resolver o problema do balanceamento de carga dinâmico. No entanto, carecem de diversas melhorias para que sejam mais coerentes com as necessidades dos MMOGs. Por exemplo, em momento algum foi considerado que, geralmente, o tráfego gerado pelos jogadores não é simplesmente linear, mas quadrático para cada aglomerado de jogadores. Tal equívoco pode gerar diferenças consideráveis entre a carga real de cada servidor e a carga estimada pelo algoritmo de balanceamento. Uma célula com 100 avatares esparsamente distribuídos no ambiente virtual teria um peso muito menor que uma célula onde outros 100 avatares estivessem todos próximos uns dos outros e interagindo entre si.

Outra questão que foi deixada de lado refere-se ao overhead, tanto no atraso para o envio de mensagens, quanto no uso de largura de banda dos servidores, quando jogadores estão interagindo cada um conectado a um servidor diferente. Em (LEE; LEE, 2003), é considerado que os servidores estão todos em uma mesma rede local, de alta velocidade e baixo atraso, e que esse overhead é desprezível. No entanto, ao se considerar um sistema servidor geograficamente distribuído, não se pode partir desse pressuposto. Esse overhead deve ser levado em conta para qualquer que seja o algoritmo de balanceamento a ser utilizado nesse sistema.

Mais um ponto importante que não foi devidamente considerado por outros trabalhos existentes foi que o critério principal a ser levado em consideração em um balanceamento de carga de servidores de MMOGs é o da largura de banda, e nem tanto o poder de processamento. Diversos jogos eletrônicos incluem simulações de ambientes virtuais, de várias centenas ou milhares de entidades, como é o caso do jogo Age of Empires [TODO:ref], que são efetuadas sem problemas nos computadores pessoais de hoje em dia. No entanto, se esse jogo fosse multijogador e cada uma dessas mesmas entidades fosse controlada por um jogador conectado em rede aos outros jogadores, muito provavelmente seria gerado um tráfego que dificilmente seria suportado por uma conexão doméstica[TODO:ref].

Além disso, a largura de banda de envio, ou *upload*, deve ser levada em conta, muito mais do que a de recebimento, ou *download*. Isso ocorre por duas razões: em primeiro lugar, o uso da banda de recebimento por cada nodo servidor cresce linearmente com o número de jogadores a ele conectados, enquanto que o uso da banda de envio pode ter um crescimento quadrático, saturando-a muito mais rapidamente; segundo, as conexões domésticas – pretende-se obter recursos para formar o sistema servidor dos próprios jogadores, pelo menos em parte – geralmente possuem uma banda de envio pequena em comparação com a banda de recebimento.

Há questões de menor importância, mas que também devem ser levadas em consideração e que não foram consideradas em outros trabalhos. Uma delas é que dificilmente o sistema servidor será homogêneo – considerando-se que é baseado em recursos voluntários. Portanto, não se pode assumir que os servidores tenham a mesma quantidade de recursos. Outro problema, específico do algoritmo proposto por (LEE; LEE, 2003), o critério de alocar para cada servidor uma carga equivalente a 90% da sua capacidade, com o fim de evitar rebalanceamentos constantes, é fraco. Não há garantias de que o sistema servidor terá 11,11% a mais de capacidade total do que o necessário. Muito pelo contrário, ele deve se adaptar a situações de sobrecarga generalizada. Além disso, o critério de parada é fraco pois o algoritmo não termina se o sistema todo estiver com carga acima de 90% da sua capacidade, ou se o servidor sobrecarregado não tiver vizinhos disponíveis para balanceamento.

Porém, uma idéia importante que foi sugerida é a do uso de grafos para representar o ambiente virtual, e usar algoritmos de particionamento de grafo para realizar o balanceamento de carga. Na seção seguinte, será feita uma breve introdução a respeito desse princípio, que é a base do esquema de balanceamento de carga aqui proposto.

5.2.1 Definições e mapeamento para grafo

Como já foi dito, será utilizada a idéia de mapear o ambiente virtual em um grafo, que será então particionado para dividir a carga do jogo entre os diferentes servidores. Para isso, é necessário primeiramente definir o que são vértices, arestas, pesos e partições no grafo que representa o mundo do jogo, além de outros conceitos que serão utilizados ao longo do texto.

- **Servidor**: aqui, servidor é definido como sendo um nodo pertencente ao sistema distribuído que servirá o jogo. A cada servidor, pode ser atribuída uma única região;

- **Capacidade do servidor**: a capacidade do servidor, $p(S)$, é um valor numérico proporcional à largura de banda de envio do servidor, que será um dos critérios para o algoritmo de balanceamento de carga utilizado;

- **Fração de capacidade do servidor**: dado um conjunto de servidores $Servers$ = $\{S_1, S_2, ..., S_n\}$, a fração de capacidade de um servidor S, $frac_p(S)$, em relação àquele conjunto, é igual à sua capacidade dividida pela soma das capacidades dos servidores do conjunto $Servers$. Tem-se:

$$frac_p(S) = \frac{p(S)}{\sum_{i=1}^{n} p(S_i)}$$

- **Capacidade do sistema**: a capacidade total do sistema, P_{total}, é igual a soma da capacidade dos n servidores que o compõem:

$$P_{total} = \sum_{i=1}^{n} p(Si)$$

- **Célula**: semelhante às microcélulas do modelo de (DE VLEESCHAUWER et al., 2005), considera-se aqui o ambiente dividido em células pequenas, com tamanho e posições fixas. Se duas células compartilham uma fronteira, elas são ditas adjacentes, ou vizinhas;

- **Região**: as células se agrupam, formando o que será chamado de regiões. Geralmente essas regiões são contíguas, embora em alguns casos o subgrafo que as representa pode ser desconexo, resultando na presença de células isoladas umas das outras. Cada região é atribuída a um servidor, e apenas um, sendo $s(R)$ o servidor associado à região R. Poderá ser lido ao longo do texto "capacidade da região", o que se refere na verdade à capacidade do servidor associado àquela região, ou seja, $p(s(R))$;

- **Relevância**: a relevância de um avatar A_i para outro, A_j, determina a freqüência das atualizações do estado de A_i que o servidor deve enviar ao jogador controlando A_j (BEZERRA; CECIN; GEYER, 2008). Pode ser representada pela função $R(a, b)$, onde a e b são avatares quaisquer;

- **Carga de um avatar**: a cada avatar estarão associados diversas outras entidades (aqui, considera-se apenas outros avatares) do jogo, cada uma com uma freqüência de atualizações de estado que precisarão ser enviadas para o jogador que controla aquele avatar. Assim sendo, para cada avatar A, a sua carga individual – ou banda de envio que o servidor utilizará para enviar a seu jogador atualizações de estado – $w_a(A)$ dependerá de quais outras entidades lhe são relevantes, e o quanto. Seja $\{A_1, A_2, ..., A_t\}$ o conjunto de todos os avatares presentes no ambiente virtual, temos:

$$w_a(A) = \sum_{i=1}^{t} R(A, Ai)$$

Seja o servidor S aquele ao qual está conectado o jogador P que controla o avatar A. Será S a enviar para P atualizações de estado de qualquer outro avatar, independente de onde esteja. Se A estiver interagindo com um avatar A', cujo jogador estiver conectado a outro servidor, S', este deverá primeiro enviar o estado de A' para S, que então encaminhará para o jogador P.

- **Carga de uma célula**: aqui, carga total da célula (ou uso da banda de envio do servidor) será igual à soma das cargas individuais dos avatares naquela célula. Seja a célula C, onde estão presentes n avatares $\{A_1, A_2, ..., A_n\}$, sendo que em todo o ambiente virtual existem, no total, t avatares. A carga da célula, $w_c(C)$, é encontrada com o seguinte somatório:

$$w_c(C) = \sum_{i=1}^{n} w(Ai) = \sum_{i=1}^{n} \sum_{j=1}^{t} R(A_i, A_j)$$

- **Carga de uma região**: a carga da região equivale à soma das cargas individuais das células que a compõem. Seja a região R formada pelas células $\{C_1, C_2, ..., C_p\}$, a carga da região, $w_r(R)$, será definida por:

$$w_r(R) = \sum_{i=1}^{p} w_c(Ci)$$

- **Fração de carga da região**: dado um conjunto de regiões $Regions$ = $\{R_1, R_2, ..., R_n\}$, a fração de carga de uma região R, $frac_r(R)$, em relação àquele conjunto, é igual à sua carga dividida pela soma das cargas das regiões do conjunto $Regions$. Tem-se:

$$frac_r(R) = \frac{w_r(R)}{\sum_{i=1}^{n} w_r(R_i)}$$

- **Uso de recursos em uma região**: fração que indica o quanto da capacidade do servidor daquela região está sendo utilizada. É definido por:

$$u(s(R)) = \frac{w_r(R)}{p(s(R))}$$

- **Carga total do jogo**: a carga total do jogo, W_{total} independe de como será feita a distribuição e será usado como parâmetro para o particionamento do ambiente virtual. Equivale à soma da carga individual de todas as células. Seja $\{C_1, C_2, ..., C_w\}$ o conjunto de todas as células em que está subdivido o mundo do jogo, temos:

$$W_{total} = \sum_{i=1}^{w} w_c(Ci)$$

- **Uso total do sistema**: fração que indica o quanto de recursos do sistema como um todo está sendo utilizado. É definido por:

$$U_{total} = \frac{W_{total}}{P_{total}}$$

- **Interação entre células**: a interação entre duas células é igual à soma de todas as interações entre pares de avatares onde cada um está situado em uma dessas células. Para tornar mais claro, sejam C_i e C_j, $i \neq j$, duas células quaisquer. Sejam $AvSet_i$ e $AvSet_j$ os conjuntos de avatares presentes em C_i e C_j, respectivamente, com cardinalidades m e n. A interação entre essas células é dada por:

$$Int_c(C_i, C_j) = \sum_{i=1}^{m} \sum_{j=1}^{n} R(A_i, A_j),$$
$$\text{onde } A_i \in AvSet_i \text{ e } A_j \in AvSet_j.$$

- **Overhead** entre duas regiões: se houver apenas um servidor e uma região, compreendendo todo o ambiente virtual do jogo, o uso da banda de envio do servidor será proporcional à carga total do jogo. No entanto, devido à distribuição em diversos servidores, surge o problema de haver jogadores de diferentes regiões interagindo um com o outro muito próximos à fronteira entre as regiões (Figura [TODO:ref]). Por causa disso, cada atualização de estado dos avatares desses jogadores será enviada duas vezes. Para exemplificar, seja A_i o avatar do jogador P_i, conectado ao servidor S_i, e A_j o avatar do jogador P_j, conectado ao servidor S_j. Para que P_i interaja com P_j, é necessário que S_i envie o estado de A_i ao servidor S_j, que então encaminha ao computador de P_j. O mesmo ocorre no caminho inverso. O overhead entre as regiões R_i e R_j é igual, portanto, à soma das interações entre pares de células onde cada uma está em uma dessas regiões. Se R_i e R_j possuem respectivamente m e n células, temos que a interação – ou *overhead* – entre elas é dada por:

$$Int_r(R_i, R_j) = \sum_{i=1}^{m} \sum_{j=1}^{n} Int_c(C_i, C_j),$$
$$\text{onde } C_i \in R_i \text{ e } C_j \in R_j.$$

  O $Int_r(R_i, R_j)$ terá um valor numérico proporcional ao uso de largura de banda de $s(R_j)$ para envio de mensagens para $s(R_i)$.

- **Overhead Total**: o overhead total sobre o sistema servidor é calculado como a soma dos overheads entre cada par de regiões. Sendo assim, temos:

$$OverHead = \sum_{i} \sum_{j,j \neq i} Int_r(Ri, Rj)$$

[TODO:figura ilustrando a carga de uma célula/regiao]
[TODO:figura com o overhead]

Agora que os conceitos necessários para entender o esquema de balanceamento de carga proposto foram definidos, será descrito como será o mapeamento dos mesmo em um grafo com pesos, para posterior particionamento. Seja $GW = (V, E)$ um grafo que reprenta o mundo do jogo, onde V é o conjunto de vértices e E é o conjunto de arestas entre os vértices. A seguir são listados cada componente desse grafo e o que representam:

- **Vértice**: cada vértice do grafo representa uma célula no ambiente virtual;
- **Aresta**: cada aresta do grafo liga dois vértices que representam células adjacentes, ou seja, que compartilham uma fronteira;
- **Partição**: cada partição do grafo GW – um subconjunto de vértices do grafo GW, mais as arestas que ligam pares de vértices nessa mesma partição – representa uma região;
- **Peso do vértice**: o peso de cada vértice é igual à carga da célula que representa;
- **Peso da aresta**: o peso da aresta que liga dois vértices é igual à interação entre as células que os mesmos representam;
- **Peso da partição**: o peso da partição é igual à soma do peso de seus vértices, ou seja, o peso da região que representa;
- **Corte de aresta**: o corte de aresta em um particionamento é igual à soma dos pesos de todas as arestas que ligam vértices de diferentes partições. Este valor é igual à soma dos overheads entre todos os pares de regiões. Assim sendo, o corte de aresta do grafo GW é igual ao overhead total sobre o sistema servidor, sendo que reduzí-lo é um dos objetivos deste esquema de balanceamento de carga.

[TODO:figura do mapeamento de cell/region para vertex/partition/edge]

O objetivo do esquema de balanceamento aqui proposto é atribuir a cada servidor uma carga proporcional à sua capacidade, reduzindo tanto quanto possível o corte de aresta do grafo que representa o ambiente virtual e, portanto, o overhead intrínseco à distribuição do jogo em vários servidores. Embora este seja um problema NP-completo [TODO:ref, de novo?], serão utilizadas heurísticas eficientes para reduzir esse problema. Nas seções a seguir serão apresentados alguns algoritmos propostos neste trabalho.

5.2.2 Algoritmos propostos

Considera-se que uma divisão inicial do ambiente virtual já foi feita. Cada servidor deverá então verificar periodicamente se está sobrecarregado o e disparar o algoritmo. Embora o overhead resultante da distribuição do ambiente virtual faça parte da carga imposta aos servidores, não há como saber qual será esse overhead sem efetuar o re-particionamento primeiro. Por esta razão, a "carga" a ser distribuída não incluirá este sobrecusto.

Quando se verifica que uma região está desbalanceada, é selecionado um grupo local de regiões, de maneira parecida com o que foi mostrado na seção 5.1.2.1, porém com algumas mudanças (seção 5.2.2.1). Após essa seleção, será utilizado um algoritmo cujos parâmetros são apenas as cargas das células e suas interações (que são os pesos dos vértices e das arestas), pois estes são os dados de que se dispõe. Por último, com as cargas

das regiões já balanceadas, será utilizado o algoritmo de Kernighan e Lin para refinar o particionamento, reduzindo o corte de aresta, porém mantendo o balanceamento.

O esquema proposto se divide então em três fases:

1. Seleção do grupo local de regiões;
2. Balanceamento dessas regiões, atribuindo a cada uma delas uma carga proporcional à capacidade do seu servidor;
3. Refinamento do particionamento, reduzindo o overhead.

Uma decisão tomada para esse esquema de balanceamento é que uma região R é considerada sobrecarregada apenas quando o uso de seus recursos é maior que o uso total do sistema, considerando também uma certa tolerância, tol, para evitar constantes rebalanceamentos. Assim, o servidor de R inicia o balanceamento sempre que, e somente quando, $u(s(R)) > U_{total} \times tol$. Dessa forma, mesmo que o sistema como um todo esteja sobrecarregado, será mantida uma qualidade de jogo semelhante nas diferentes regiões, dividindo o excedente de carga entre todos os servidores de maneira justa. O que pode ser feito quando $U_{total} > 1$ é reduzir gradativamente a quantidade de informações enviadas a cada atualização de estado, deixando a aplicação extrapolar as lacunas com base em atualizações anteriores.

Outro aspecto importante é que cada servidor sempre tem uma região associada. O que pode ocorrer é de uma região ser vazia, ou seja, ela não possui células e o seu servidor não participa da execução do jogo. Isso é útil quando a capacidade total do sistema servidor é muito maior que a carga total do jogo, ou seja, $P_{total} \gg W_{total}$. Neste caso, a introdução de mais servidores apenas aumentaria o overhead de comunicação no sistema, sem melhorar sua qualidade – salvo quando provesse tolerância a falhas.

Os algoritmos foram desenvolvidos orientados a regiões, ao invés de servidores, para que fossem mais legíveis, levando em conta as constantes transferências de células no código. Além disso, torna-se mais fácil no futuro estender o modelo de balanceamento utilizado aqui para que mais de um servidor possa administrar a mesma região. Os algoritmos propostos são descritos a seguir, sendo que o da seção 5.2.2.1 é para a fase 1; os das seções 5.2.2.2 a 5.2.2.5 são opções para a fase 2; e o algoritmo da seção 5.2.2.6 é o refinamento da fase 3.

5.2.2.1 Seleção local de regiões

O algoritmo de seleção de regiões (Algoritmo 2) tem como objetivo formar um conjunto de regiões tal que o uso médio de recursos dos servidores dessas regiões esteja abaixo de um certo limite. Partindo da região do servidor que disparou o balanceamento, as regiões vizinhas com menor uso de recursos vão sendo adicionadas. Quando o uso médio for menor que 1, ou menor que U_{total} (linha 5), a seleção termina e a fase 2 começa, tendo como entrada o conjunto formado nesta fase. Essas duas condições se justificam porque existem duas possibilidades: $U_{total} \leq 1$ e $U_{total} > 1$.

No caso de $U_{total} \leq 1$, existe capacidade suficiente no sistema para que todos os servidores tenham um uso menor que 100%. Assim, são adicionadas regiões ao grupo até que todos os servidores envolvidos estejam usando uma quantidade de recursos menor ou igual à que possuem. No entanto, quando $U_{total} > 1$, não há como todos os servidores estarem usando menos que 100% de seus recursos ao mesmo tempo. Assim, considera-se suficiente que todos os servidores estejam igualmente sobrecarregados, e que algum tipo

de adaptação seja feita, o que provavelmente será uma redução nas informações enviadas aos jogadores a cada atualização de estado.

Se mesmo após serem selecionadas todas as regiões vizinhas, as vizinhas das vizinhas e assim por diante, o critério não for cumprido, regiões vazias – pertencentes a servidores ociosos até então – serão incluídas no grupo (linha 8), pois o overhead de interação entre regiões introduzido por elas se justifica pela necessidade de mais recursos.

Algoritmo 2 Seleção local de regiões

1: $grupo\_local \leftarrow \{R\}$
2: $carga\_local \leftarrow w_r(R)$
3: $capacidade\_local \leftarrow p(s(R))$
4: $uso\_médio \leftarrow \frac{carga\_local}{capacidade\_local}$
5: **enquanto** $uso\_médio > max(1, U_{total})$ **faça**
6: **se** há alguma região não selecionada vizinha a um dos elementos do $grupo\_local$ **então**
7: $R \leftarrow$ região vizinha a um dos elementos do $grupo\_local$ que não foi selecionada, com o menor $u(s(R))$
8: **senão se** há alguma região vazia **então**
9: $R \leftarrow$ região vazia com o maior $p(s(R))$
10: **senão**
11: pare. não há mais regiões a selecionar.
12: **fim se**
13: $carga\_local \leftarrow carga\_local + w_r(R)$
14: $capacidade\_local \leftarrow capacidade\_local + p(s(R))$
15: $uso\_médio \leftarrow \frac{carga\_local}{capacidade\_local}$
16: $grupo\_local \leftarrow grupo\_local \cup \{R\}$
17: **fim enquanto**
18: executar a fase 2 passando $grupo\_local$ como parâmetro

Após terem sido selecionadas as regiões que serão rebalanceadas, uma lista com as mesmas é passada como parâmetro ao algoritmo da fase 2 que as reparticionará.

5.2.2.2 ProGReGA

Uma opção para a fase 2 é o algoritmo de crescimento proporcional e guloso de região (**ProGReGA**, *proportional greedy region growing algorithm*), que busca atribuir as células mais carregadas às regiões gerenciadas pelos servidores mais poderosos. Os detalhes estão exibidos no Algoritmo 3.

Como foi dito, é passada como parâmetro uma lista das regiões cuja carga será rebalanceada. Isso torna possível o uso desse algoritmo tanto em âmbito local quanto global, pois a lista pode conter apenas algumas das regiões do mundo do jogo. A distribuição será feita com base nas informações desse conjunto de regiões, cujas carga e capacidade totais são calculas nas linhas 1 a 7. Para que sejam posteriormente redistribuídas, todas as células associadas a essas regiões são liberadas (linha 6).

Para prover um particionamento balanceado, proporcional e com um corte de aresta já reduzido na primeira fase do balanceamento, as regiões são ordenadas em ordem decrescente de capacidade de seus servidores (linha 8). O ProGReGA então percorre essa lista ordenada, buscando atribuir às regiões com mais capacidade as células mais pesadas, enquanto que encarrega as regiões com servidores mais fracos de células menos carregadas.

Algoritmo 3 ProGReGA

1: $carga\_a\_dividir \leftarrow 0$
2: $capacidade\_livre \leftarrow 0$
3: **para** cada região R na $lista\_de\_regiões$ **faça**
4: $carga\_a\_dividir \leftarrow carga\_a\_dividir + w_r(R)$
5: $capacidade\_livre \leftarrow capacidade\_livre + p(s(R))$
6: libere temporariamente todas as células de R
7: **fim para**
8: ordene a $lista\_de\_regiões$ em ordem decrescente de $p(s(R))$
9: **para** cada região R na $lista\_de\_regiões$ **faça**
10: $parcela\_da\_carga \leftarrow carga\_a\_dividir \times \frac{p(s(R))}{capacidade\_livre}$
11: **enquanto** $w_r(R) < parcela\_da\_carga$ **faça**
12: **se** existe alguma célula de R vizinha a uma célula livre **então**
13: $R \leftarrow R \cup \{$célula vizinha livre ligada pela aresta mais pesada$\}$
14: **senão se** existe alguma célula livre **então**
15: $R \leftarrow R \cup \{$célula livre mais pesada$\}$
16: **senão**
17: pare. não há mais células livres.
18: **fim se**
19: **fim enquanto**
20: **fim para**

Na linha 10, é calculado qual a parcela de carga que cabe a cada região, considerando a carga total que está sendo dividida e a capacidade total daquelas regiões. A fração da capacidade de servidor de cada região, em relação ao conjunto de todas as regiões envolvidas no balanceamento, $\frac{p(s(R))}{capacidade\_livre}$, deve ser a mesma fração de carga que deve ser atribuída a ela. Ainda que essa carga seja maior que a capacidade do servidor, resultando em uma sobrecarga, todos os servidores estarão igualmente sobrecarregados, satisfazendo o critério de balanceamento que foi definido. A condição para o término da alocação de novas células a uma região é que a sua carga seja maior ou igual a essa parcela.

A escolha de células para incluir na região busca fazer com que cada uma das arestas mais pesadas do grafo GW ligue vértices da mesma partição, reduzindo o corte de aresta e, assim, o overhead. A cada passo é dado preferência a escolher a célula livre que não apenas seja vizinha de uma célula já presente na região, mas também cuja aresta ligando-as seja a mais pesada possível. A Figura [TODO:reffig] mostra um exemplo dos passos do crescimento de uma região até atingir a sua parcela de carga no balanceamento.

[TODO:figura -> passos do crescimento de uma região]

No primeiro passo do ciclo iniciado na linha 16, quando a região ainda não tem célula nenhuma, o ProGReGA lhe atribui a célula livre mais pesada que houver (linha 15). O mesmo ocorre quando uma região está comprimida entre as fronteiras de outras regiões (Figura [TODO:ref]), e não possui nenhum vizinho livre, precisando buscar células em outros lugares. Isso pode gerar regiões fragmentadas e possivelmente aumentar o corte de aresta do grafo que representa o ambiente virtual. No entanto, isso acontecerá com mais freqüência nos últimos passos da distribuição, quando a maior parte das células já estaria alocada em regiões. Pelo fato do algoritmo ser guloso, quando se chegasse a essa fase de sua execução, as células que ficariam isoladas seriam, provavelmente, células muito mais leves que as outras, causando pouco overhead.

[TODO: figura região comprimida]

5.2.2.3 ProGReGA-KH

Um possível efeito indesejável do algorito ProGReGA é que, ao liberar todas as células e redistribuí-las, pode ocorrer de uma ou mais regiões mudar completamente de lugar, fazendo com que vários jogadores precisem desconectar-se de um servidor e reconectar-se a outro. Para tentar reduzir a probabilidade disso acontecer, é proposto o **ProGReGA-KH** (*proportional greedy region growing algorithm keeping heaviest cell*). Este novo algoritmo (exibido em Algoritmo 4) é semelhante ao ProGReGA, exceto que cada região mantém sua célula mais carregada (linhas 6 e 8), a partir da qual pode ser formado uma região semelhante à que se tinha anteriormente, evitando que vários jogadores precisem migrar de servidor (Algoritmo 4). No entanto, ao se manter uma das células de cada região, pode-se estar impedindo que um balanceamento melhor ocorra, ou introduzindo-se overhead, se a presença daquela célula impedir que a região seja completamente contígua.

Algoritmo 4 ProGReGA-KH

1: $carga\_a\_dividir \leftarrow 0$
2: $capacidade\_livre \leftarrow 0$
3: **para** cada região R na $lista\_de\_regiões$ **faça**
4: $\quad carga\_a\_dividir \leftarrow carga\_a\_dividir + w_r(R)$
5: $\quad capacidade\_livre \leftarrow capacidade\_livre + p(s(R))$
6: $\quad c \leftarrow$ célula mais pesada de R
7: $\quad$ libere temporariamente todas as células de R
8: $\quad R \leftarrow R \cup \{c\}$
9: **fim para**
10: ordene a $lista\_de\_regiões$ em ordem decrescente de $p(s(R))$
11: **para** cada região R na $lista\_de\_regiões$ **faça**
12: $\quad parcela\_da\_carga \leftarrow carga\_a\_dividir \times \frac{p(s(R))}{capacidade\_livre}$
13: $\quad$ **enquanto** $w_r(R) < parcela\_da\_carga$ **faça**
14: $\quad\quad$ **se** existe alguma célula de R vizinha a uma célula livre **então**
15: $\quad\quad\quad R \leftarrow R \cup \{$célula vizinha livre ligada pela aresta mais pesada$\}$
16: $\quad\quad$ **senão se** existe alguma célula livre **então**
17: $\quad\quad\quad R \leftarrow R \cup \{$célula livre mais pesada$\}$
18: $\quad\quad$ **senão**
19: $\quad\quad\quad$ pare. não há mais células livres.
20: $\quad\quad$ **fim se**
21: $\quad$ **fim enquanto**
22: **fim para**

5.2.2.4 ProGReGA-KF

Outra maneira de tentar minimizar a migração de jogadores entre servidores por causa do rebalanceamento é com o **ProGReGA-KF**, ou *proportional greedy region growing algorithm keeping usage fraction* (Algoritmo 5). Neste algoritmo, cada região vai liberando suas células em ordem crescente de carga, até que a fração de carga da região seja menor ou igual à fração de capacidade do seu servidor. Dessa forma, mantêm-se as células mais carregadas no mesmo servidor e, portanto, a maior parte dos jogadores não precisará migrar. Após isso, as células que foram liberadas vão sendo distribuídas entre as regiões

com menor uso dos recursos de seu servidor (linha 13). A desvantagem deste algoritmo é, tal como no ProGReGA-KH, a possibilidade de fragmentar as regiões em muitas células isoladas por causa da redistribuição das células livres, aumentando o overhead.

Algoritmo 5 ProGReGA-KF

1: $carga\_a\_dividir \leftarrow 0$
2: $capacidade\_livre \leftarrow 0$
3: **para** cada região R na $lista\_de\_regiões$ **faça**
4: $carga\_a\_dividir \leftarrow carga\_a\_dividir + w_r(R)$
5: $capacidade\_livre \leftarrow capacidade\_livre + p(s(R))$
6: $lista\_de\_células \leftarrow$ lista de celulas de R em ordem crescente de carga
7: **enquanto** $frac_r(R) > frac_p(s(R))$ **faça**
8: $C \leftarrow$ primeiro elemento de $lista\_de\_células$
9: remova C de R
10: remova C de $lista\_de\_células$
11: **fim enquanto**
12: **fim para**
13: ordene $lista\_de\_regiões$ em ordem crescente de $u(s(R))$
14: **para** cada região R na $lista\_de\_regiões$ **faça**
15: $parcela\_da\_carga \leftarrow carga\_a\_dividir \times \frac{p(s(R))}{capacidade\_livre}$
16: **enquanto** $w_r(R) < parcela\_da\_carga$ **faça**
17: **se** existe alguma célula de R vizinha a uma célula livre **então**
18: $R \leftarrow R \cup \{$célula vizinha livre ligada pela aresta mais pesada$\}$
19: **senão se** existe alguma célula livre **então**
20: $R \leftarrow R \cup \{$célula livre mais pesada$\}$
21: **senão**
22: pare. não há mais células livres.
23: **fim se**
24: **fim enquanto**
25: **fim para**

5.2.2.5 BFBCT

O **BFBCT** (*best-fit based cell transference*) é proposto aqui como uma alternativa ao ProGReGA e suas variantes. O objetivo do algoritmo é verificar qual é a carga excedente em cada região e transferí-la para regiões cuja capacidade livre é a mais próxima desse valor. Isso é feito transferindo células cuja carga se aproxime mais da capacidade livre da região que as receberá, observado duas restrições: primeiro, a carga total transferida não pode ser maior que a capacidade da região de destino e, segundo, não se deve transferir uma carga maior do que o necessário para eliminar a sobrecarga. A segunda restrição se justifica porque uma maior transferência de carga provavelmente implicaria em uma maior quantidade de jogadores migrando. A exceção a essa regra seria quando uma célula está mais carregada que a outra, não por ter mais avatares, mas porque estes estão mais próximos, conferindo crescimento quadrático ao tráfego entre os jogadores. O Algoritmo 6 descreve com detalhes o funcionamento do BFBCT.

Algoritmo 6 BFBCT

para cada região R_i em $lista\_de\_regiões$ **faça**
  $carga\_a\_perder \leftarrow w_r(R_i) - W_{total} \times frac_p(s(R_i))$
  $regiões\_de\_destino \leftarrow lista\_de\_regiões - \{R_i\}$
  ordene $regiões\_de\_destino$ em ordem decrescente de $u(s(R))$
  para cada região R_j em $regiões\_de\_destino$ **faça**
    $capacidade\_livre \leftarrow frac_p(s(R_j)) \times W_{total} - w_r(R_j)$
    $carga\_para\_esta\_região \leftarrow min(carga\_a\_perder, capacidade\_livre)$
    enquanto $carga\_para\_esta\_região > 0$ **faça**
      se R_i tem uma célula com carga menor ou igual a $carga\_para\_esta\_região$ **então**
        $C \leftarrow$ célula de R_i com o peso mais próximo de $carga\_para\_esta\_região$, porém não maior
        $R_i \leftarrow R_i - \{C\}$
        $R_j \leftarrow R_j \cup \{C\}$
        $carga\_para\_esta\_região \leftarrow carga\_para\_esta\_região - w_c(C)$
        $carga\_a\_perder \leftarrow carga\_a\_perder - w_c(C)$
      senão
        continue com o próximo R_j.
      fim se
    fim enquanto
  fim para
fim para

5.2.2.6 Refinamento com o algoritmo Kernighan-Lin

Depois de balancear a carga entre os diferentes servidores na fase 2, cada servidor terá uma taxa de uso de seus recursos semelhante à dos outros. No entanto, a fase 2 pode ter gerado regiões fragmentadas, implicando em muitos pares de células vizinhas que não são gerenciadas pelo mesmo servidor. Isso aumenta a probabilidade de dois jogadores, cada um em uma dessas células, interagirem entre si, por estarem próximos da mesma fronteira, o que, como foi mostrado em seções anteriores, causa um desperdício de recursos do sistema servidor.

Para minimizar esse problema, é proposto aqui o uso de uma variante do algoritmo de Kernighan e Lin (KERNIGHAN; LIN, 1970). Embora o algoritmo receba como entrada um grafo, os vértices, arestas, pesos e partições podem ser interpretados, respectivamente, como células, interações, cargas e regiões. Dado um par de regiões, o Kernighan-Lin busca pares de células que, se trocados entre essas regiões, diminuirão o overhead.

Sejam R_A e R_B duas regiões e a uma célula tal que $a \in A$. A interação externa da célula a é definida como $E(a) = \sum_{b \in R_B} Int_c(a, b)$ e a interação interna é definida como $I(a) = \sum_{a' \in R_A} Int_c(a, a')$. A célula a está propensa a trocar de região se sua interação com R_B for maior que sua interação com R_A, ou seja $D(a) = E(a) - I(a) > 0$. Supondo que exista uma célula $b \in R_B$, tal que $D(a) + D(b) - 2 \times Int_c(a, b) > 0$, a troca irá reduzir o overhead entre as regiões. A expressão $D(a) + D(b) - 2 \times Int_c(a, b)$ é chamada de $gain(a, b)$, pois representa o ganho (redução do overhead) ao serem trocadas a e b de região.

No caso de um ambiente virtual distribuído entre várias regiões – geralmente, há mais do que duas regiões – executa-se o Kernighan-Lin para cada par de regiões. Além disso,

após cada troca deverá ser mantido o balanceamento – caso contrário, todas as células poderiam ir para a mesma região, eliminando completamente o overhead. Considera-se que é retornado pelo algoritmo um valor de verdadeiro se foi realizada alguma troca e falso, se não foi. Ele é executado para todos os pares de regiões até que retorne um valor de falso, indicando que não há mais trocas possíveis.

O Algoritmo 7 mostra de que maneira o algoritmo de Kernighan-Lin é chamado.

Algoritmo 7 Uso do Kernighan-Lin

```
1:  swapped ← true
2:  enquanto swapped = true faça
3:      swapped ← false
4:      para cada região R_i em lista_de_regiões faça
5:          para cada região R_j em lista_de_regiões faça
6:              se Kernighan–Lin(R_i, R_j) = true então
7:                  swapped ← true
8:              fim se
9:          fim para
10:     fim para
11: fim enquanto
```

5.3 Implementação

Nesta seção será descrito como foi implementado o protótipo com objetivo de validar o esquema proposto, utilizando a linguagem de programação C++. Avatares, células, regiões e servidores foram modelados como classes, que se relacionavam para formar o ambiente do jogo. Além disso, foi criada uma interface gráfica para visualizar o funcionamento do protótipo, corrigir erros e interferir disparando diferentes algoritmos para verificar como funcionava cada um. As classes implementadas foram: **`Server`**, **`Region`**, **`Cell`** e **`Avatar`**, representando os servidores, regiões, células e avatares.

5.3.1 A classe `Avatar`

A classe **`Avatar`** é a mais importante de todas, pois reproduz as ações dos jogadores, é a responsável pela carga imposta ao servidores, além de ser a base de todo o esquema de balanceamento de carga proposto. Cada objeto desta classe possui uma referência para um objeto da classe **`Cell`**, representando a célula onde ele está localizado. Além disso, ele possui duas coordenadas (**`int posx, posy`**), que representam sua posição exata no ambiente virtual. Além deste, a classe Avatar possui outros dois pares ordenandos, **`int dirx, diry`** e **`int destx, desty`**, que representam sua direção e destino atuais, inicializados com valores aleatórios quando o objeto é criado.

Em um jogo real, cada ação de um jogador é executa pelo seu avatar, permitindo-lhe interagir com o ambiente virtual e com avatares de outros jogadores. No entanto, aqui todos os avatares foram simulados utilizando o modelo de *random waypoint* [TODO:ref] com *random wait* [TODO:ref]. A cada passo – **`void step(unsigned long delay)`** –, o avatar verifica se já chegou ao destino que foi definido (**`distance(posx, posy, destx, desty) < MAX_TARGET_DIST`**). Se já chegou, define um tempo para espera aleatória e depois calcula seu próximo destino, com uma probabilidade de escolher um dos pontos de interesse pré-definidos. Se ainda não chegou, ele incrementa sua posição,

com uma velocidade variável, em direção à posição de destino:

```
//(...)
int speed = rand() % MAX_SPEED + 1;
incr_y = (desty - posy) / distance(posx, posy, destx, desty);
incr_x = (destx - posx) / distance(posx, posy, destx, desty);
posy += incr_y * speed;
posx += incr_x * speed;
//(...)
```

Além de mover-se de posição (ou permanecer na mesma, se estiver em espera), o avatar verifica se ele atravessou alguma fronteira entre células. Isso é possível porque as células têm posição estática, bastando verificar se a nova posição do avatar pertence a uma célula diferente. Caso isso tenha ocorrido, ele envia mensagens, através de chamadas de método (`my_cell->unsubscribe(this)` e `new_cell->subscribe(this)`) para os objetos `Cell` que representam sua célula antiga e sua célula nova, notificando a mudança.

Por fim, a classe `Avatar` implementa mais dois métodos. Um objeto `Avatar` invoca o método `int OtherRelevance(Avatar* other)` quando se quer saber o quão relevente para ele é o avatar `*other`. Já o método `long getInteraction(Cell* c)` retorna a soma das relevâncias de todos os avatares na célula `*c` em relação ao avatar que o invocou. Para calcular a relevância, decidiu-se utilizar um algoritmo semelhante ao que foi proposto em (BEZERRA; CECIN; GEYER, 2008).

5.3.2 A classe `Cell`

Cada célula tem uma posição fixa e todas as células tem o mesmo tamanho, estando dipostas em forma de grade. A classe `Cell` tem uma matriz estática de ponteiros do tipo `Cell*`, sendo que o colunas multiplicado pela largura de cada célula é igual à largura do ambiente virtual. Da mesma forma, o número de linhas multiplicado pelo comprimento de cada célula é igual ao comprimento do ambiente virtual [TODO:reffigmatriz]. Dessa forma, pode-se encontrar os índices do objeto do tipo `Cell` na matriz `Matrix[i][j]` a partir da posição ocupada pelo avatar em tempo constante e com apenas informações locais, da seguinte forma:

$$\texttt{i} = \left\lfloor \text{N\_COLUNAS} \times \frac{\texttt{posx}}{\text{LARGURA TOTAL}} \right\rfloor$$

e

$$\texttt{j} = \left\lfloor \text{N\_LINHAS} \times \frac{\texttt{posy}}{\text{COMPRIMENTO TOTAL}} \right\rfloor$$

Cada objeto da classe `Cell` possui a lista dos avatares (`list<Avatar*> avatars`) presentes nela, uma referência à região que contém a célula (`Region* parentRegion`) e um vetor com os pesos das arestas (interações) ligando-a com as células vizinhas (`long edgeWeight[NUM_NEIGH]`).

Para calcular a carga da célula, é utilizado o método `long getCellWeight()`, que é a soma das cargas dos avatares. Uma otimização importante desta implementação foi a definição de células cujo comprimento e largura são ambos maiores que o alcance máximo de visão dos avatares. Dessa forma, avatares que estejam a mais de uma célula de distância terão sempre relevância nula um para o outro. Assim, para se calcular a carga de um avatar, basta ter a lista de avatares das células vizinhas às daquela onde ele está situado, economizando largura de banda e tempo de processamento (Figura [TODO:reffigoptIM]).

A classe **`Cell`** implementa também outros métodos, que são necessários para os algoritmos descritos na seção 5.2.2, tais como: **`long getDesireToSwap(Region* r)`**, **`static Cell* getHighestEdgeFreeNeighbor(list<Cell*> &cellList)`** e **`static Cell* getHeaviestFreeCell()`**.

5.3.3 A classe `Region`

Cada objeto da classe **`Region`** tem uma lista (**`list<Cell*> cells`**) de referências para as células que a compõem, além de uma lista das regiões que fazem fronteira com ela (**`list<Region*> neighbors`**) e uma referência para o objeto **`Server`** que representa seu servidor. É na classe **`Region`** que estão implementados os algoritmos que foram apresentados na seção 5.2.2.

Para verificar se uma região está balanceada (i.e. $u(s(R)) \leq 1$ ou $u(s(R)) \leq U_{TOTAL}$), a aplicação invoca o método **`void checkBalancing()`** da classe **`Region`**. Se essa condição de balanceamento não estiver satisfeita, é iniciada a seleção local de servidores com o método **`void startLocalBalancing()`**, correspondente à fase 1 do balanceamento (seção 5.2.2.1). Após este método concluir a formação do grupo local de regiões, ele invoca um dos algoritmos da fase 2, que correspondem aos seguinte métodos de **`Region`**:

`rebalance_progrega(list<Region*> regionsToRebalance)`,
`rebalance_progrega_kh(list<Region*> regionsToRebalance)`,
`rebalance_progrega_kf(list<Region*> regionsToRebalance)` e
`rebalance_bfbct(list<Region*> regionsToRebalance)`, todos **`static`** e **`void`**.

Quando a chamada ao algoritmo de fase 2 termina, é invocado o método **`static bool refineKL_kwise(list<Region*> regionsToRefine, int passes)`** que por sua vez irá invocar o método **`static bool refineKL_pairwise(Region* r1, Region* r2)`** para cada par de regiões, tal como foi descrito na seção 5.2.2.6 a respeito do refinamento da divisão (fase 3).

5.3.4 A classe `Server`

Cada objeto da classe **`Server`** tem como atributos principais uma referência para um objeto da classe **`Region`** e um valor inteiro (**`long`**) que representa sua capacidade. No protótipo implementado, sua única função é fornecer o valor de sua capacidade (**`long serverPower`**) ao objeto **`Region`** associado a cada servidor, além de fornecer o valor de $frac_p(S)$ e P_{total}, através dos métodos **`double getPowerFraction()`** e **`static long getMultiserverPower()`**.

5.3.5 Uso de valores inteiros

Apesar da relevância ter sido definida como um valor real entre 0 e 1 (BEZERRA; CECIN; GEYER, 2008), o uso de variáveis do tipo **`float`** ou **`double`** mostrou-se propício a erros por causa da precisão flutuante. A carga total do sistema é igual à soma das cargas das regiões, onde cada região tem como carga a soma das cargas das células, e a carga de cada célula é igual à soma das cargas dos avatares e, por fim, a carga de cada avatar é igual à soma das relevâncias de todos os outros avatares em relação a ele.

No entanto, ao ser feito um somatório de centenas de parcelas de precisão flutuante, as últimas parcelas do somatório têm uma probabilidade considerável de serem descartadas, pelo fato de seu valor provavelmente estar abaixo da precisão do resultado parcial. Embora existam maneiras de contornar isso, como somatório em árvore, preferiu-se definir

a relevância, as cargas, as interações, a capacidade do servidor, o overhead etc. como inteiros do tipo `long`.

Dessa forma, evitaram-se os problemas de imprecisão, porém mantendo o código simples. O valor da relevância de um avatar em relação ao outro foi redefinido, portanto, como um valor **inteiro**, de 0 a 100, inclusive, o que pode ser considerado suficiente se for levado em conta que, tradicionalmente, há apenas dois níveis de relevância.

5.4 Simulações e resultados

Para realizar a simulação do balanceamento de carga, foi simulado um ambiente virtual com diversos avatares que se movem segundo o modelo de *random waypoint* [TODO:ref] com *random wait*. Porém, a escolha do waypoint ao qual se dirigir não foi completamente aleatória. Foram definidos três pontos de interesse, e havia uma probabilidade de o havatar escolher um desses pontos de interesses como waypoint para seu próximo movimento. O objetivo disso foi tornar a distribuição dos avatares não uniforme no ambiente virtual, pondo à prova os algoritmos da fase 2 propostos. Aqueles que levassem em conta a existência de pontos de interesse formariam as regiões com base nisso, reduzindo o overhead da distribuição.

O ambiente consistia em um espaço bidimensional, divido em 225 células, formando uma matriz de ordem 15. Nele estavam presente 750 avatares. Todas as células sempre pertenciam a alguma região, embora pudessem ser transferidas de uma para outra. Havia oito servidores ($S_1, S_2, ..., S_8$), cada um com uma região associada, sendo que cada uma podia ter de 0 a todas as 225 células. A capacidade de todos os servidores era diferente, sendo que $p(S_i) = i \times 20000$. Assim o servidor S_1 tinha capacidade de 20000 e S_8, de 160000, por exemplo. Dessa forma, pôde-se testar se a distribuição estava atendendo o critério de balanceamento proporcional de carga. Cada seção de jogo simulada foi de 20 minutos. A carga total do jogo foi propositalmente ajustada de maneira que fosse maior que a capacidade total do sistema, forçando à execução do balanceamento de carga, devido à sobrecarga dos servidores.

No gráfico da Figura 5.8, pode-se ver que todos os algoritmos propostos atendem ao critério de proporcionalidade no balanceamento de carga. No entanto, alguns deles introduzem mais overhead que os outros. A razão disso é a fragmentação que causam nas regiões. Quanto menor o número de fragmentos que têm as regiões, menor será a superfície de contato entre elas e, consequentemente, menor será o overhead total.

Observa-se que o algoritmo ProGReGA é o que menos causa overhead, pois ele foi desenvolvido justamente de forma a criar o máximo possível de regiões contíguas, buscando células ligadas pelas interações mais pesadas. O algoritmo BFBCT, no entanto, foi desenvolvido com o intuito de distribuir a carga com base em uma alocação *best-fit*, procurando balanceá-la tanto quanto possível, ignorando, na fase 2, a existência de interações entre células e regiões. Por esta razão, o BFBCT foi o que mais gerou fragmentação nas regiões e, consequentemente, overhead entre os servidores.

A Figura 5.9 mostra a variação do overhead total sobre o sistema servidor, em função do tempo de jogo. Percebe-se que o overhead gerado por cada esquema de balanceamento varia relativamente pouco com o tempo, sendo o ProGReGA o que apresenta o menor overhead em qualquer instante do jogo, e o BFBCT é o maior de todos, durante quase toda a simulação. Os algoritmos ProGReGA-KH e ProGReGA-KF se revesam com valores intermediários de overhead.

No entanto, a introdução de overhead não é o único critério considerado. Na figura

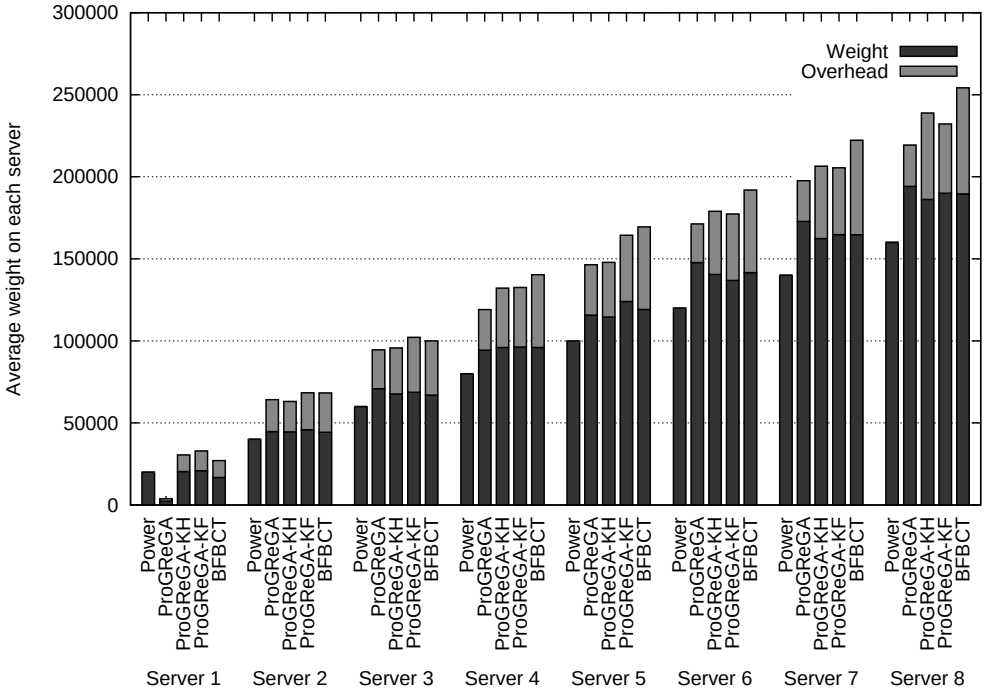

Figura 5.8: Overall comparison of proposed load balancing algorithms

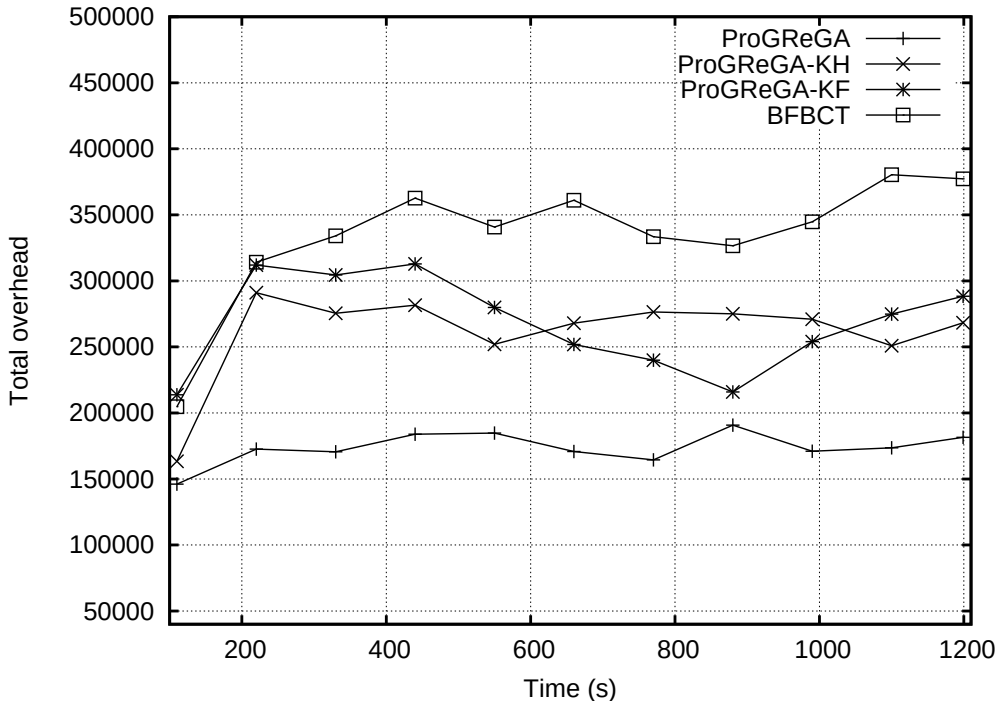

Figura 5.9: Overhead introduced by each balancing algorithm during the game session

5.10 é mostrado o quanto de migrações de usuários ocorreram entre servidores durante todo a sessão de jogo simulada, para cada algoritmo utilizado na fase 2. O valor de migração foi dividido em dois: *walk migration*, que ocorre quando um jogador troca de servidor porque moveu-se de uma região para outra; e *still migration*, que ocorrem quando um jogador troca de servidor sem ter se movido. A still migration ocorre porque

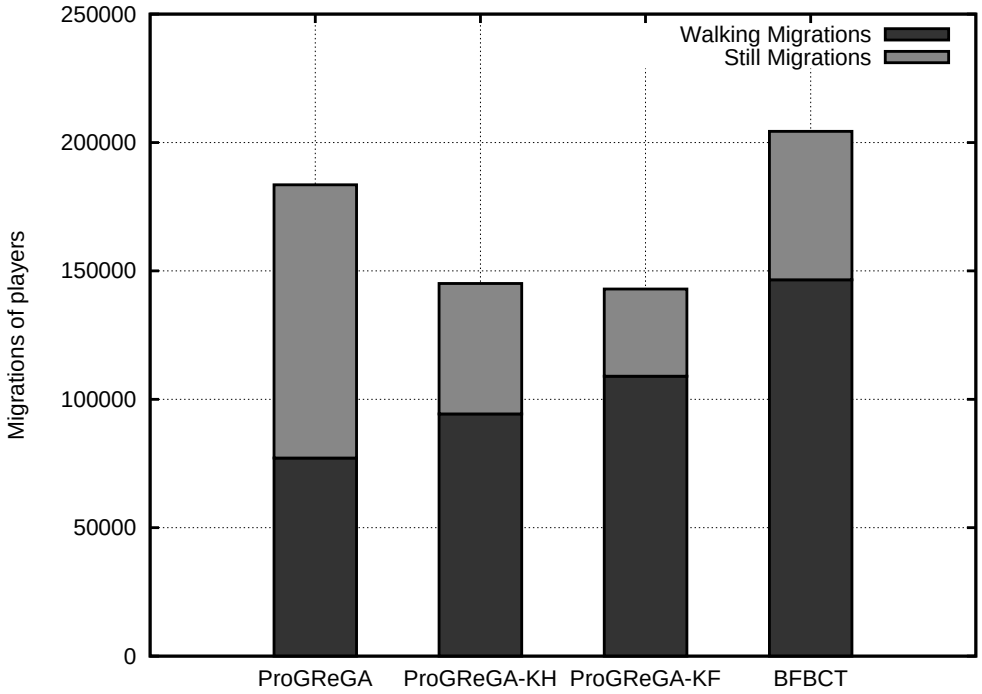

Figura 5.10: Players migrating while still or walking depending on algorithm

a célula onde está o avatar do jogador foi transferida para outra região como resultado de um rebalanceamento da carga do jogo. Já a walk migration é mais provável de acontecer quando as regiões não são contíguas.

Podemos observar que o ProGReGA é o que tem menor quantidade de walk migrations, justamente por suas regiões serem contíguas. No entanto, pelo fato de não tentar minimizar as transferências de células – o ProGReGA tem como principal objetivo minimizar o overhead – a quantidade de still migrations é a maior de todos os algoritmos, e o número total de migrações é o segundo maior, sendo melhor apenas que o BFBCT, que também foi o pior algoritmo neste critério. A estratégia do ProGReGA-KH e, mais fortemente, do ProGReGA-KF de manter o máximo de células possível quando é feito o rebalanceamento tem como resultado os menores números de migrações de jogadores dos algoritmos simulados.

Outro detalhe a ser considerado é o da uniformidade de distribuição: todos os servidores devem apresentar uma taxa de uso de seus recursos o mais parecida possível, para que a distribuição seja considerada fair. Para medir essa uniformidade, foi calculado o desvio padrão, σ, do $u(S)$ de todos os servidores, para cada algoritmo simulado. A figura 5.11 mostra a variação de σ em função do tempo.

Mesmo com os servidores apresentando a maior variação da taxa de uso de seus recursos, o ProGReGA foi o que reduziu ao máximo o uso real de recursos ($W_{total}+Overhead$) de todos os algoritmos propostos. Já a sua variante ProGReGA-KF reduz ao máximo as migrações dos usuários, em troca de um aumento do overhead em 48%, em média, se comparado à versão original do algoritmo. Qual dos dois seria melhor no geral dependeria muito do jogo sendo utilizado. Em um jogo de tempo-real, usuários migrando constantemente de servidor pode introduzir atraso e prejudicar a interação entre os jogadores, sendo o ProGReGA-KF o mais indicado, tendo apresentado um número total de migrações 22% menor que o ProGReGA. Por outro lado, pode não ser possível o uso de algum

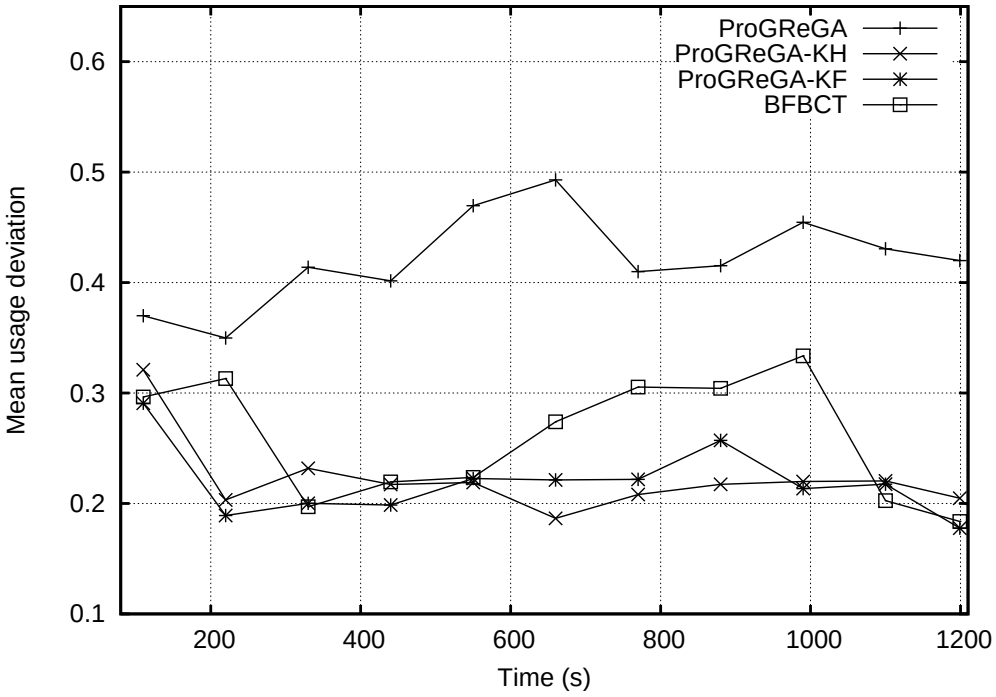

Figura 5.11: Deviation of the ideal usage value

tipo de "degradação graciosa" para reduzir dinamicamente a qualidade do jogo, precisando economizar ao máximo seus recursos. Neste caso, o ProGReGA seria a melhor opção.

5.5 Conclusões e trabalhos futuros

Foi proposto aqui um esquema de balanceamento de carga para servidores distribuídos de MMOGs, levando em conta o uso de banda de envio dos nodos servidores. Foram considerados aspectos importantes, como o crescimento quadrático do tráfego quando os avatares estão próximos, e o overhead inerente à distribuição quando jogadores conectados a servidores diferentes têm de interagir. Ao esquema, que é dividido em três fases, foram propostos diferentes algoritmos para a fase 2 (balanceamento em si), sendo que o ProGReGA foi o que apresentou o menor overhead de todos, e o ProGReGA-KF foi o que apresentou menor número de migrações de usuários.

Como trabalhos futuros, os algoritmos apresentados podem ser ainda mais refinados, se a carga considerada para o balanceamento for uma média da carga em um período – nos últimos minutos, por exemplo – ao invés do valor instantâneo. Dessa forma, seriam evitados rebalanceamentos desnecessários que seriam causados por uma carga muito oscilante.

Outro possível trabalho futuro é criar um esquema de balanceamento de carga onde não haverá mais apenas um nodo servidor, mas um grupo de nodos servidores em cada região. Neste caso, podem ser investigadas maneiras de balancear a carga do sistema, considerando essa distribuição em dois níveis. Uma idéia seria a de usar apenas réplicas, com o intuito de prover tolerância a falhas na região. A carga dentro de uma região seria redistribuída apenas quando a mesma estivesse sobrecarregada e não fosse viável reparticioná-la.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Como parte do mestrado, definiu-se um modelo preliminar, que visa justamente ao particionamento do ambiente virtual em regiões, cada uma atribuída a um servidor, sobre o qual foi definido um esquema de balanceamento de carga. O Cosmmus – *Cooperative System for Massive Multiplayer Service* – será um novo modelo, a ser definido durante o doutorado, que também utilizará o princípio usado no mestrado de distribuição do jogo através de particionamento do ambiente virtual em regiões, porém atacando problemas mais críticos, que foram deixados de lado até então.

O Cosmmus deverá incluir soluções para os principais aspectos relacionados a MMOGs, como distribuição de tarefas, sincronização da simulação, persistência distribuída e assim por diante. Embora este modelo não esteja definido ainda, já há algumas idéias de como ele poderá ser e que tipo de abordagens ele deve utilizar, baseado em pesquisa relacionada. Nas seções seguintes, serão apresentadas estas idéias iniciais, que virão a ser melhor desenvolvidas durante o curso de doutorado.

6.1 Distribuição da carga do jogo

Já foram feitos alguns trabalhos relacionados ao particionamento do ambiente virtual do jogo (DE VLEESCHAUWER et al., 2005; MORILLO; FERNÁNDEZ; ORDUÑA, 2006; LUI; CHAN, 2002). Para o Cosmmus, pensa-se em fazer a distribuição da carga do jogo entre os nodos servidores em dois níveis:

1. Assim como no mestrado, será utilizada a idéia de células de mesmo tamanho e posição fixa, que são áreas do ambiente. Essas células então serão agrupadas em regiões, cujo tamanho e formato irão depender da maneira como os avatares dos jogadores estão mais ou menos concentrados em determinados lugares. Isto deverá ser feito de maneira transparente para o jogador, que verá um mundo grande e contíguo através do qual poderá mover-se livremente;

2. Cada região poderá ser designada a mais de um nodo servidor, com dois objetivos principais: primeiramente, prover tolerância a falhas por meio de replicação e, em segundo lugar, permitir que regiões que já estão muito sobrecarregadas sejam gerenciadas por mais de um nodo servidor.

No caso de uma região estar muito sobrecarregada, poderia-se pensar em fazer um novo particionamento, em duas ou mais novas regiões, mas isto poderia ser contraproducente em determinadas situações, como no caso em que a região engloba um hotspot e sua divisão resultaria em duas regiões excessivamente dependentes entre si. Para isso deverá

ser criado também um algoritmo, métrica ou heurística que determine se uma região deve ou não ser subdividida em novas regiões.

Foi proposto durante o mestrado um esquema de balanceamento de carga dinâmico, baseado em particionamento de grafo e refinamento de suas partições (KERNIGHAN; LIN, 1970; KARYPIS et al., 1999; KARYPIS; KUMAR, 1998; DUTT, 1993). O mundo é dividido em células de mesmo tamanho e posição fixa, cada uma representada por um vértice em um grafo. A interação (medida em kilobytes por segundo) entre as células são representadas pelas arestas dos vértices. O algoritmo consiste no particionamento do grafo, sendo que cada partição representa uma região a ser atribuída a um servidor. Tal balanceamento leva em conta diferentes níveis de recursos entre os diferentes servidores e leva em conta as necessidades dos MMOGs, que se concentram com maior peso no uso de largura de banda dos servidores.

Também seria necessário um esquema de balanceamento de carga, que, pretende-se, seguirá o mesmo princípio do esquema proposto no mestrado. A questão é que, diferente do modelo do mestrado, não haverá mais apenas um nodo servidor por região, mas um grupo de nodos servidores. Neste caso, serão investigadas maneiras de balancear a carga do sistema, considerando essa distribuição em dois níveis. Uma idéia seria a de usar apenas réplicas, com o intuito de prover tolerância a falhas na região. A carga dentro de uma região seria redistribuída apenas quando a mesma estivesse sobrecarregada e não fosse viável reparticioná-la.

6.2 Formação da rede lógica do sistema servidor

Quanto à formação da rede overlay sobre a qual será executado o sistema servidor, existem propostas, como em (TA et al., 2008; RATNASAMY et al., 2002; HAN; WATSON; JAHANIAN, 2005; ZHANG et al., 2004), que buscam montar a rede de forma que nodos com boa conexão de rede entre si estejam adjacentes na rede lógica. Foi proposto, por exemplo, agrupar os nodos de acordo com o atraso de rede entre eles. Pretende-se utilizar estas técnicas de maneira que certos nodos servidores servindo regiões adjacentes e, principalmente, nodos que estejam servindo à mesma região, pertençam ao mesmo grupo. Pode-se também adaptá-las, incluindo outros critérios além do atraso, como a velocidade da transmissão de dados entre eles, em bits por segundo.

O propósito de montar a rede overlay seguindo essa estratégia seria o de minimizar o atraso na interação entre jogadores. Essa atraso na interação seria o tempo decorrido desde o envio de uma ação de um jogador, processamento pelo servidor e recebimento do estado resultante por outro jogador. Se dois jogadores quaisquer estiverem interagindo um com o outro, porém conectados a nodos servidores diferentes, será necessário que cada uma de suas interações seja intermediada pelos dois servidores. Assim, se estes dois servidores tiverem baixo atraso entre eles, a interação dos jogadores será menos prejudicada pelo acréscimo de um segundo servidor intermediário.

6.3 Sincronização de estado

Uma das questões centrais que o Cosmmus deverá tratar é a da manutenção da consistência da simulação entre os diferentes nodos servidores. Ainda que cada jogador veja apenas uma parte do mundo virtual, este mundo deve ser o mesmo para todos os servidores. Assim, se dois jogadores estão com seus avatares no mesmo lugar do ambiente virtual, eles deverão visualizar o mesmo ambiente. Outra questão importante é a ordena-

ção dos eventos. Supondo que dois jogadores estivessem lutando no jogo, e o avatar de um deles atirasse com uma arma qualquer e o outro se movesse, a ordem desses eventos seria crucial para decidir se o outro jogador teve seu avatar morto, apenas ferido ou se evitou completamente o disparo.

No caso em que os dois jogadores estivessem conectados ao mesmo servidor, como é tradicionalmente feito por empresas de jogos, isto seria trivial, bastando verificar qual das ações chegou primeiro. Ainda que se utilizasse um mecanismo de compensação de atraso – o *timestamp* da ação de cada jogador seria igual ao tempo atual menos o seu atraso –, como é feito em Half-Life (VALVE, 1998), por exemplo, não seria muito problemático ordenar estes eventos. No caso de um servidor distribuído, seria necessário que os nodos servidores entrassem em algum tipo de acordo ou sincronia no que diz respeito a essa ordenação, cuja complexidade cresceria com o número de servidores envolvidos.

Levando em conta que há várias regiões formando um mundo contíguo, e que em cada uma existe um ou mais servidores, estas questões referentes a manter a consistência da simulação se tornam ainda mais difíceis quando um avatar se aproxima de uma fronteira da região onde está localizado. Espera-se que a divisão do ambiente não seja percebida pelo jogador. Então, ao se aproximar de uma fronteira, ele deverá ser capaz de ver tudo que estiver além dela e de mover-se livremente de uma para outra. Para isto, será necessário que o servidor ao qual ele está conectado comunique-se com aquele que serve a região, além da fronteira, que está sendo visualizada pelo jogador. Serão transmitidas informações como: estado do ambiente, avatares que estejam próximos, estados desses avatares e assim por diante.

Em um trabalho do GPPD (CECIN et al., 2006), foi proposto um esquema de sincronização de estado que consiste de dois simuladores: um conservador e um otimista. Enquanto que o otimista executa as ações dos jogadores assim que são recebidas e exibe imediatamente seu resultado na tela do jogador, o simulador conservador divide o tempo em turnos e somente executa as ações dos jogadores após haver uma sincronização. Esta sincronização consiste em que cada nodo executando a simulação receba as ações de todos os participantes envolvidos – ou a notificação de que estes não fizeram nada –, que foram em seguida ordenadas e processadas. Somente após este passo é que o simulador conservador avança para o próximo turno. No caso da simulação otimista discrepar além do tolerável da simulação conservadora, esta sobrescreve o seu estado sobre a primeira, que continua a partir daí. No entanto, este esquema foi proposto para uma arquitetura completamente descentralizada, sendo necessário analizá-lo e fazer as adaptações necessárias para que possa ser utilizado em um sistema servidor distribuído em dois níveis, como se pretende para o Cosmmus. Existem também outros trabalhos que visam a manter consistente a simulação, como (ZHOU; SHEN, 2007; MOON et al., 2006; MÜLLER; GÖSSLING; GORLATCH, 2006), que serão também estudados e possivelmente alguns de seus princípios farão parte da solução para consistência a ser definida para o modelo final.

6.4 Persistência distribuída

No que tange ao armazenamento do estado do mundo virtual e dos jogadores, deverá haver uma maneira de fazer isto de maneira distribuída, já que não haverá um único servidor central. Poderia ser mais simples escolher um dos nodos servidores para guardar estas informações, no entanto isso poderia fazer com que o estado do jogo dependesse da disponibilidade e confiabilidade deste nodo escolhido e, ainda que estes dois requi-

sitos fossem satisfeitos, este servidor poderia ser saturado pelo número de requisições, tornando mais lenta a recuperação dos estados nele armazenados.

O Cosmmus deverá então utilizar algum esquema de armazenamento distribuído, onde porções da informação de estado são armazenados em diferentes lugares. Além disso, deverá ser utilizada replicação para que o estado esteja disponível ainda que alguns nodos do sistema se desconectem inesperadamente. Redes lógicas baseadas em DHT, como Pastry (ROWSTRON; DRUSCHEL, 2001), Tapestry (ZHAO et al., 2004), Chord (STOICA et al., 2001) e Koorde (KAASHOEK; KARGER, 2003), parecem ser possíveis soluções para prover o tipo de armazenamento distribuído e redundante que se quer para o estado do jogo. Pretende-se, portanto, investigar o quanto essas redes são realmente adequadas para este fim. Existem simuladores, como por exemplo o OverSim (BAUMGART; HEEP; KRAUSE, 2007), que permitem a simulação de redes overlay, tais como as que foram citadas, e através de simulação será avaliado o comportamento do sistema utilizando essas redes lógicas.

O que pode ocorrer é que seja excessivo o custo de manter uma rede como a Pastry sobre o sistema servidor, recuperando e armazenando os estados utilizando multicast no nível de aplicação e utilizando replicação, se somado ao próprio custo que os nodos servidores já enfrentarão para executar a simulação em si, recebendo ações dos jogadores e devolvendo-lhes a resposta. Para não sobrecarregar o sistema com o tráfego referente à persistência do estado do jogo, poderiam ser ajustados parâmetros, tais como freqüência de atualização e, no nível da aplicação, o nível de detalhamento do estado a ser persistido. Por exemplo, ao invés de armazenar a localização, direção e ação sendo executada por cada avatar em uma área do ambiente virtual, poderia ser armazenada apenas sua localização.

6.5 Otimização do uso de largura de banda

Pelo fato dos nodos servidores serem, idealmente, de baixo custo, é necessário economizar seus recursos tanto quanto for possível, mas sem prejudicar a qualidade do jogo. Por exemplo, podem ser feitas filtragens por interesse, onde a cada jogador só é enviado aquilo que ele pode visualizar, de forma a economizar tráfego tanto entre clientes e servidores, quanto entre os próprios servidores. Seguindo esse mesmo princípio, servidores de duas regiões diferentes podem ter uma versão antiga do estado um do outro, se não estiver havendo interação entre jogadores conectados a servidores diferentes. Dessa forma, estes servidores não precisarão atualizar-se um ao outro com tanta freqüência.

Periodicamente, ou no momento em que for necessário (e.g. dois jogadores de diferentes regiões interagindo próximo à fronteira), poderão ser trocados os estados das regiões. A granularidade dessa informação pode ser ainda mais fina, e cada servidor poderia enviar para o outro apenas os dados mais essenciais. Por exemplo, cada servidor enviaria ao outro apenas o estado do ambiente virtual próximo à fronteira entre as regiões, ao invés do estado da região inteira.

Diversos propostas já foram realizados com o intuito de fazer filtragem por interesse (BEZERRA; CECIN; GEYER, 2008; AHMED; SHIRMOHAMMADI, 2008; MORGAN; LU; STOREY, 2005; MINSON; THEODOROPOULOS, 2005). Além de investigar qual a que melhor se aplica ao modelo proposto, e de verificar possíveis adaptações que sejam necessárias, outras maneiras de economizar os recursos do sistema servidor serão estudadas.

REFERÊNCIAS

AHMED, D. T.; SHIRMOHAMMADI, S. A Dynamic Area of Interest Management and Collaboration Model for P2P MMOGs. **Proceedings of the 12th IEEE International Symposium on Distributed Simulation and Real Time Applications, 2008, Vancouver, BC, Canadá**, [S.l.], p.27–34, 2008.

ARENANET. **Guild Wars**. http://www.guildwars.com/.

ASSIOTIS, M.; TZANOV, V. A distributed architecture for MMORPG. **Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games**, [S.l.], 2006.

BAUMGART, I.; HEEP, B.; KRAUSE, S. **OverSim: a flexible overlay network simulation framework**. 2007. 79–84p.

BEZERRA, C. E. B.; CECIN, F. R.; GEYER, C. F. R. A3: a novel interest management algorithm for distributed simulations of mmogs. **Proceedings of the 12th IEEE International Symposium on Distributed Simulation and Real Time Applications, 2008, Vancouver, BC, Canadá**, [S.l.], p.35–42, 2008.

BLIZZARD. **World of Warcraft**. http://www.worldofwarcraft.com/.

BOULANGER, J.; KIENZLE, J.; VERBRUGGE, C. Comparing interest management algorithms for massively multiplayer games. **Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games**, [S.l.], 2006.

CECIN, F.; GEYER, C.; RABELLO, S.; BARBOSA, J. A peer-to-peer simulation technique for instanced massively multiplayer games. **Proceedings of the Tenth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'06)-Volume 00**, [S.l.], p.43–50, 2006.

CECIN, F.; REAL, R.; OLIVEIRA JANNONE, R. de; RESIN GEYER, C.; MARTINS, M.; VICTORIA BARBOSA, J. FreeMMG: a scalable and cheat-resistant distribution model for internet games. **Distributed Simulation and Real-Time Applications, 2004. DS-RT 2004. Eighth IEEE International Symposium on**, [S.l.], p.83–90, 2004.

DE VLEESCHAUWER, B.; VAN DEN BOSSCHE, B.; VERDICKT, T.; DE TURCK, F.; DHOEDT, B.; DEMEESTER, P. Dynamic microcell assignment for massively multiplayer online gaming. **Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games**, [S.l.], p.1–7, 2005.

DUTT, S. **New faster Kernighan-Lin-type graph-partitioning algorithms**. 1993. 370–377p.

EL-SAYED, A. Application-Level Multicast Transmission Techniques over the Internet. **University of Paris**, [S.l.], 2004.

FIDUCCIA, C.; MATTHEYSES, R. **A Linear-Time Heuristic for Improving Network Partitions**. 1982. 175–181p.

HAN, J.; WATSON, D.; JAHANIAN, F. **Topology aware overlay networks**. 2005. v.4.

IIMURA, T.; HAZEYAMA, H.; KADOBAYASHI, Y. Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games. **Proceedings of ACM SIGCOMM 2004 workshops on NetGames' 04: Network and system support for games**, [S.l.], p.116–120, 2004.

KAASHOEK, M.; KARGER, D. Koorde: a simple degree-optimal distributed hash table. **LECTURE NOTES IN COMPUTER SCIENCE**, [S.l.], p.98–107, 2003.

KARYPIS, G.; KUMAR, V. **Multilevel algorithms for multi-constraint graph partitioning**. 1998. 1–13p.

KARYPIS, G.; KUMAR, V.; CENTER, A. H. P. C. R.; MINNESOTA, U. of. Parallel multilevel k-way partitioning scheme for irregular graphs. **SIAM Review**, [S.l.], v.41, n.2, p.278–300, 1999.

KERNIGHAN, B.; LIN, S. An efficient heuristic procedure for partitioning graphs. **Bell System Technical Journal**, [S.l.], v.49, n.2, p.291–307, 1970.

KIM, J.; CHOI, J.; CHANG, D.; KWON, T.; CHOI, Y.; YUK, E. Traffic characteristics of a massively multi-player online role playing game. **Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games**, [S.l.], p.1–8, 2005.

LEE, K.; LEE, D. A scalable dynamic load distribution scheme for multi-server distributed virtual environment systems with highly-skewed user distribution. **Proceedings of the ACM symposium on Virtual reality software and technology**, [S.l.], p.160–168, 2003.

LUI, J.; CHAN, M. An Efficient Partitioning Algorithm for Distributed Virtual Environment Systems. **IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS**, [S.l.], p.193–211, 2002.

MCCANNE, S.; FLOYD, S. et al. Network simulator ns-2. **Available for download at http://www.isi.edu/nsnam/ns**, [S.l.], 2006.

MINSON, R.; THEODOROPOULOS, G. An adaptive interest management scheme for distributed virtual environments. **Proc. of PADS '05**, [S.l.], p.273–281, 2005.

MOON, K.; MUTHUKKUMARASAMY, V.; NGUYEN, A.; PAN, Z.; AYLETT, R.; DIENER, H.; JIN, X.; GOBEL, S.; LI, L. Efficiently Maintaining Consistency Using Tree-Based P2P Network System in Distributed Network Games. **LECTURE NOTES IN COMPUTER SCIENCE**, [S.l.], v.3942, p.648, 2006.

MORGAN, G.; LU, F.; STOREY, K. Interest management middleware for networked games. **Symposium on Interactive 3D Graphics: Proceedings of the 2005 symposium on Interactive 3D graphics and games**, [S.l.], v.3, n.06, p.57–64, 2005.

MORILLO, P.; FERNÁNDEZ, M.; ORDUÑA, J. M. Solving the Partitioning Problem in Distributed Virtual Environment Systems Using Evolutive Algorithms. **Handbook of bioinspired algorithms**, [S.l.], p.531–554, 2006.

MORSE, K. et al. Interest management in large-scale distributed simulations. **Technical Report ICS-TR-96-27, University of California, Irvine**, [S.l.], 1996.

MÜLLER, J.; GÖSSLING, A.; GORLATCH, S. **On correctness of scalable multi-server state replication in online games**. 2006.

NCSOFT. **Lineage II**. http://www.lineage2.com/.

RAK, S.; VAN HOOK, D. Evaluation of grid-based relevance filtering for multicast group assignment. **Proc. of 14th DIS workshop**, [S.l.], p.739–747, 1996.

RATNASAMY, S.; HANDLEY, M.; KARP, R.; SHENKER, S. Topologicallyaware overlay construction and server selection. **Proceedings of IEEE INFOCOM'02, New York, NY, June 2002.**, [S.l.], 2002.

ROWSTRON, A.; DRUSCHEL, P. **Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems**. 2001.

SCHIELE, G.; SUSELBECK, R.; WACKER, A.; HAHNER, J.; BECKER, C.; WEIS, T. Requirements of Peer-to-Peer-based Massively Multiplayer Online Gaming. **Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid**, [S.l.], p.773–782, 2007.

SMED, J.; KAUKORANTA, T.; HAKONEN, H. A Review on Networking and Multiplayer Computer Games. **Turku Centre for Computer Science**, [S.l.], 2002.

STOICA, I.; MORRIS, R.; KARGER, D.; KAASHOEK, M.; BALAKRISHNAN, H. **Chord: a scalable peer-to-peer lookup service for internet applications**. 2001. 149–160p.

SVOBODA, P.; KARNER, W.; RUPP, M. Traffic Analysis and Modeling for World of Warcraft. **Communications, 2007. ICC'07. IEEE International Conference on**, [S.l.], p.1612–1617, 2007.

TA, D.; ZHOU, S.; CAI, W.; TANG, X.; AYANI, R. Network-Aware Server Placement for Highly Interactive Distributed Virtual Environments. **Proceedings of the 12th IEEE International Symposium on Distributed Simulation and Real Time Applications, 2008, Vancouver, BC, Canadá**, [S.l.], p.95–102, 2008.

VALVE. **Half-Life**. http://www.half-life.com/.

YAN, J.; RANDELL, B. A systematic classification of cheating in online games. **Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games**, [S.l.], p.1–9, 2005.

YU, Y.; LI, Z.; SHI, L.; CHEN, Y.; XU, H. Network-Aware State Update For Large Scale Mobile Games. **Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on**, [S.l.], p.563–568, 2007.

ZHANG, X.; ZHANG, Q.; ZHANG, Z.; SONG, G.; ZHU, W. A construction of locality-aware overlay network: moverlay and its performance. **Selected Areas in Communications, IEEE Journal on**, [S.l.], v.22, n.1, p.18–28, 2004.

ZHAO, B.; HUANG, L.; STRIBLING, J.; RHEA, S.; JOSEPH, A.; KUBIATOWICZ, J. Tapestry: a resilient global-scale overlay for service deployment. **Selected Areas in Communications, IEEE Journal on**, [S.l.], v.22, n.1, p.41–53, 2004.

ZHOU, S.; SHEN, H. **A Consistency Model for Highly Interactive Multi-player Online Games**. 2007. 318–323p. v.26, n.28.

ZOU, L.; AMMAR, M.; DIOT, C. An evaluation of grouping techniques for state dissemination in networked multi-user games. **Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on**, [S.l.], p.33–40, 2001.