Utilizando kd-trees para particionar o ambiente virtual e balancear dinamicamente a carga sobre servidores de MMOGs

Carlos Eduardo B. Bezerra*
Grupo de Processamento Paralelo e Distribuído
Instituto de Informática - UFRGS
Av. Bento Gonçalves, 9500, Porto Alegre

Resumo

MMOGs (*massively multiplayer online games*, ou jogos online maciçamente multijogador), são aplicações que requerem conexões com grande largura de banda para funcionarem adequadamente. Essa demanda por largura de banda é maior principalmente nos servidores que hospedam o jogo. Como nesse tipo de jogo costuma haver milhares a dezenas de milhares de jogadores simultâneos, sendo que a interação entre cada par de jogadores é intermediada pelo servidor, é sobre este que recai o maior custo no que se refere a uso de largura de banda para realizar o envio de atualizações de estado do ambiente do jogo para os jogadores. Para contornar este problema, são propostas arquiteturas com vários servidores, onde cada um deles gerencia uma região do ambiente virtual, e cada jogador conecta-se somente ao servidor que gerencia a área onde ele está jogando. No entanto, para distribuir a carga entre os servidores, é necessário um algoritmo de particionamento do ambiente virtual. Para que se possa reajustar a distribuição de carga durante o jogo, esse algoritmo deve ser dinâmico. Alguns trabalhos já foram feitos nesse sentido, mas, utilizando um algoritmo geométrico mais adequado do que os encontrados na literatura, deve ser possível alcançar um nível melhor de granularidade da distribuição, sem comprometer o tempo de rebalanceamento, ou mesmo reduzindo-o. Neste trabalho, é proposta a utilização de uma KD-Tree para dividir o ambiente virtual do jogo em regiões, cada uma das quais sendo designada a um dos servidores. As coordenadas de divisão das regiões são ajustadas dinamicamente de acordo com a distribuição dos avatares no ambiente virtual.

Palavras-chave: MMOGs, balanceamento de carga, servidor distribuído, kd-trees

1 Introdução

Os MMOGs têm como principal característica a grande quantidade de jogadores interagindo simultaneamente, chegando a ter dezenas até centenas de milhares de participantes simultâneos [Schiele et al. 2007]. Ao se usar uma arquitetura cliente-servidor para que os jogadores se comuniquem entre si, é necessário que esse servidor intermedie a comunicação entre cada par de jogadores.

Para permitir que os jogadores interajam, o servidor recebe os comandos de cada jogador, calcula o estado resultante do jogo e envia o novo estado para todos os jogadores que estiverem interagindo com o primeiro. É fácil perceber que o número de mensagens de atualização de estado enviadas pelo servidor será proporcional ao quadrado no número de jogadores, se todos estiverem interagindo com todos. Obviamente, dependendo desse número de jogadores, o custo de manutenção de uma infra-estrutura centralizada como essa se torna muito alto, restringindo esse mercado de jogos MMOG a grandes empresas, que disponham de recursos suficientes para tal.

Buscando reduzir esse custo, tem-se buscado soluções descentralizadas. Algumas destas utilizam redes par-a-par: [Schiele et al. 2007; Rieche et al. 2007; Hampel et al. 2006; El Rhalibi e Merabti 2005; Iimura et al. 2004; Knutsson et al. 2004]. Outras propõem a utilização de um sistema distribuído composto por nodos servidores de baixo custo, conectados através da Internet, como é proposto em [Ng et al. 2002; Chertov e Fahmy 2006; Lee e Lee 2003; Assiotis e Tzanov 2006]. De qualquer forma, em todas essas abordagens, o "mundo", ou ambiente virtual do jogo, é dividido em regiões e, para cada região, é designado um servidor – ou um grupo de pares para administrarem-no em conjunto, no caso das redes par-a-par. Cada uma dessas regiões deve possuir um conteúdo tal que a carga imposta sobre o servidor correspondente não seja maior que a sua capacidade.

Quando um *avatar* (representação do jogador no ambiente virtual) é posicionado em uma determinada região, o jogador daquele avatar conecta-se ao servidor associado àquela região e é dele que o jogador receberá as mensagens de atualização para que o ambiente virtual seja visualizado no seu computador com o estado mais recente do jogo. Quando um servidor se torna sobrecarregado, devido a uma maior concentração de avatares em sua região e, consequentemente, mais jogadores a serem atualizados, a divisão do ambiente virtual deve ser recalculada de maneira a aliviar aquele servidor.

Geralmente, é feita uma divisão do ambiente virtual em grades de células, com posterior agrupamento das mesmas, formando regiões que são distribuídas entre os servidores. No entanto, tal abordagem tem uma limitação severa em sua granularidade, já que as células têm tamanho e posição fixos. Utilizando um algoritmo geométrico mais adequado, deve ser possível conseguir uma melhor distribuição dos jogadores entre os diferentes servidores, fazendo uso de técnicas tradicionais que geralmente são utilizadas na área de computação gráfica.

Neste trabalho, é proposto o uso de uma kd-tree para realizar o particionamento do ambiente virtual. Quando um servidor fica sobrecarregado, ele dispara o balanceamento de carga, reajustando os limites de sua região com a ajuda da estrutura de dados kd-tree. Foi feita a implementação de um protótipo, que foi utilizado para realizar simulações. Os resultados encontrados através das simulações serão comparados com resultados anteriores obtidos ao se utilizar a técnica de divisão do ambiente virtual do jogo em células estáticas.

O texto está organizado da seguinte forma: na seção 2, são apresentados alguns trabalhos relacionados; na seção 3, é apresentado em detalhes o algoritmo proposto aqui; nas seções 4 e 5 são apresentadas as simulações e os resultados alcançados e, na seção 6, são apresentadas as conclusões a que se chegou com este trabalho.

2 Trabalhos relacionados

Diferentes autores já atacaram o problema de particionamento do ambiente virtual em MMOGs para distribuição entre vários servidores [Ahmed e Shirmohammadi 2008; Bezerra e Geyer 2009]. Geralmente, é feita uma divisão em células estáticas, de posição e tamanho fixos. As células são então agrupadas em regiões (Figura

*e-mail: carlos.bezerra@inf.ufrgs.br

Figura 1: *Divisão em células e agrupamento em regiões*

1), e cada região é delegada a um dos servidores. Quando um deles está sobrecarregado, ele busca outros servidores, que possam absorver seu excesso carga. Isso é feito distribuindo uma ou mais células do servidor sobrecarregado a outros servidores.

[Ahmed e Shirmohammadi 2008] propõem um modelo de balanceamento de carga orientado a células. Para balancear a carga, seu algoritmo encontra, primeiro, todos os agrupamentos de células que são gerenciadas pelo servidor sobrecarregado. Seleciona-se o agrupamento que contiver o menor número de células e, deste agrupamento, é escolhida a célula que tiver menor interação com outras células do mesmo servidor – considerando que a interação entre duas células A e B é definida como o número de pares de avatares interagindo um com outro, estando um em A e o outro em B. A célula escolhida é, então, transferida para o servidor menos carregado, sendo que a "carga" é definida como o uso de largura de banda para enviar atualizações de estado aos avatares posicionados em células gerenciadas por aquele servidor. Esse processo se repete até que o servidor não esteja mais sobrecarregado ou que não haja mais servidores capazes de absorver a carga excedente – neste caso, uma opção seria diminuir a freqüência de envio das atualizações de estado [Bezerra et al. 2008].

Em [Bezerra e Geyer 2009], também é proposta a divisão em células. Para realizar a divisão, o ambiente é representado por um grafo (Figura 2), onde cada vértice representa uma célula. Cada aresta no grafo liga dois vértices que representam células vizinhas. O peso de um vértice equivale ao uso de largura de banda do servidor para enviar atualizações de estado aos jogadores cujos avatares estão na célula representada por aquele vértice. A interação entre cada duas células definirá o peso da aresta que liga os vértices correspondentes. Para formar as regiões, o grafo é particionado, utilizando um algoritmo guloso: começando do vértice mais pesado, a cada passo adiciona-se o vértice ligado pelas aresta mais pesada a algum dos vértices já selecionados, até que o peso total da partição do grafo (soma dos pesos dos vértices) atinja um determinado limite relacionado à capacidade total do servidor que receberá a região representada por aquela partição do grafo.

Embora essa abordagem funcione, há uma séria limitação na granularidade da distribuição que pode ser feita. Se for desejada uma granularidade fina, é necessário definir as células como sendo muito pequenas, aumentando o número de vértices no grafo que representa o ambiente virtual e, conseqüentemente, o tempo necessário para executar o balanceamento. Sendo assim, pode ser melhor utilizar uma outra abordagem para o particionamento do ambiente virtual que utilize uma estrutura de dados mais adequada, tal como a kd-tree [Bentley 1975].

Esse tipo de estrutura de dados costuma ser usado na computação

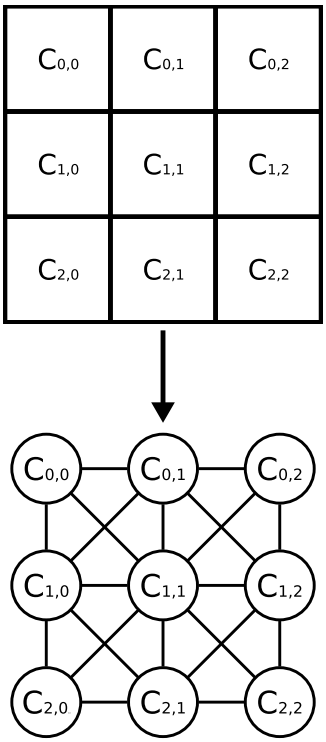

Figura 2: *Representação do ambiente em um grafo*

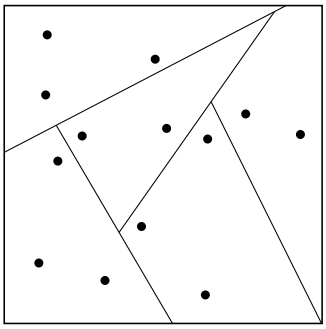

Figura 3: *Particionamento do espaço com uma árvore BSP*

gráfica. No entanto, como em MMOGs também há informação geométrica – como a posição de cada avatar no ambiente –, árvores de particionamento do espaço podem ser utilizadas. Além disso, já existem técnicas de distribuição de objetos no espaço, buscando manter o balanceamento entre as diferentes regiões definidas pela árvore. Em [Luque et al. 2005], por exemplo, busca-se reduzir o tempo necessário para calcular as colisões entre pares de objetos se movendo no espaço. Para isso, é utilizada uma árvore BSP (*binary space partitioning*, ou particionamento binário do espaço) para distribuir os objetos da cena (Figura 3). Obviamente, se cada objeto de um par está completamente contido em uma partição diferente, eles não colidem e não é necessário fazer um teste mais demorado para esse par. Partindo de uma divisão inicial, é proposto pelos autores que a árvore se ajuste dinamicamente à medida que os objetos se deslocam, balanceando a distribuição dos mesmos na árvore, evitando que o tempo necessário para o cálculo das colisões se eleve muito. Algumas das idéias propostas pelos autores podem ser utilizadas para o contexto de balanceamento de carga entre servidores de um MMOG.

3 Proposta

A proposta para balanceamento de carga dinâmico para MMOGs apresentada aqui tem como base dois critérios: primeiramente, deve-se considerar a possibilidade do sistema ser heterogêneo, ou seja, de que cada servidor tenha uma quantidade diferente de recursos. Aqui se define como "recursos" a largura de banda de envio que aquele servidor tem disponível para enviar atualizações de estado para os jogadores a ele conectados.

Essa escolha se deve ao fato de que cada jogador envia comandos para o servidor a uma taxa constante, logo o número de mensagens recebidas pelo servidor por unidade de tempo cresce linearmente em relação ao número de jogadores, enquanto que, como foi discutido, o número de atualizações de estado enviadas pelo servidor pode ser quadrático, no pior caso.

Como foi dito na introdução, para dividir o ambiente do jogo em regiões propõe-se utilizar uma estrutura de dados conhecida, que é a kd-tree. A grande maioria dos MMOGs, como World of Warcraft [Blizzard 2004], Ragnarök [Gravity 2001] e Lineage II [NCsoft 2003], apesar de possuir gráficos em três dimensões, o mundo simulado – cidades, florestas, pântanos e pontos de interesse em geral – nestes jogos é mapeado em duas dimensões. Por esse motivo, propõe-se o uso de uma kd-tree com k = 2.

A cada nó da árvore corresponde uma região do espaço e, além disso, neste nó é armazenado um valor correspondente a uma coordenada de divisão. Cada um dos dois filhos daquele nó representarão uma subdivisão da região representada pelo nó pai, sendo que um deles representa a sub-região com coordenada menor do que a coordenada de divisão, e o outro, a região com coordenada maior ou igual à coordenada de divisão. A cada nível da árvore, alterna-se o eixo da coordenada de divisão (no caso de duas dimensões, os eixos x e y). Cada nó folha também representa uma região do espaço, porém não armazena nenhuma coordenada de divisão, além de apontar para uma lista de avatares presentes naquela região. Por fim, cada nó folha é associado a um dos servidores do jogo. Quando um servidor fica sobrecarregado, ele dispara o balanceamento de carga, que utiliza a estrutura de dados kd-tree, reajustando as coordenadas de divisão que definem as regiões e, assim, diminuindo a quantidade de conteúdo gerenciada por ele.

A cada nó da árvore estão também associados outros dois valores: capacidade e carga da sub-árvore. A carga de um nó da árvore é igual à soma da carga de seus filhos. Da mesma forma, a capacidade de um nó intermediário é igual à soma da capacidade de seus nós filhos. No caso do nó folha, esses valores são os mesmos do servidor associado a ele. A raiz da árvore, por outro lado, tem como carga o tráfego total de mensagens de atualização de estado do jogo e, como capacidade, a soma das capacidades individuais dos servidores.

Nas seções a seguir, serão descritos a construção da árvore, o cálculo da carga associada a cada servidor e o algoritmo de balanceamento proposto.

3.1 Construção da kd-tree

Para fazer uma divisão inicial do espaço, é construída uma kd-tree balanceada (tanto quanto possível, pois depende do número de servidores – ou nós folha – ser igual a uma potência de 2). Para isto, foi utilizada a função recursiva exibida no Algoritmo 1 para a criação da árvore.

No Algoritmo 1, o id serve para calcular quantos filhos cada nodo deve ter e, nos nós folha, determina qual o servidor associado à região representada por aquela folha da árvore. O objetivo com isso é tentar criar uma árvore balanceada, onde o número de nós

Algoritmo 1 nó::constrói_árvore(id, nível, num_servidores)

se id + $2^{nível} \geq num\_servidores$ **então**
  $filho\_menor \leftarrow filho\_maior \leftarrow NIL$;
  retorne;
senão
  $filho\_menor \leftarrow$ novo_nó();
  $filho\_menor.pai \leftarrow$ **this**;
  $filho\_maior \leftarrow$ novo_nó();
  $filho\_maior.pai \leftarrow$ **this**;
  $filho\_menor$.constrói_árvore
  $(id, nível + 1, num\_servidores)$;
  $filho\_maior$.constrói_árvore
  $(id + 2^{nível}, nível + 1, num\_servidores)$;
fim se

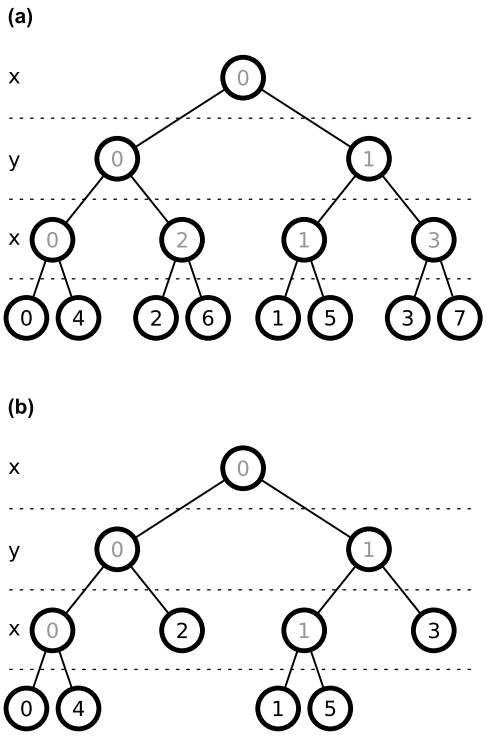

Figura 4: *Construção de kd-tree balanceada*

folha de cada uma das duas sub-árvores de cada nó difere de no máximo um. Na Figura 4 (a), temos uma kd-tree completa formada com esse algoritmo simples e, em (b), como seria uma kd-tree não completa com seis nós-folha. Como se pode perceber, cada nodo da árvore de (b) tem duas sub-árvores cujos números de nós folha diferem, no máximo, de um.

3.2 Cálculo da carga de avatares e de nós da árvore

A maneira como serão definidas as coordenadas de divisão das regiões nos nodos intermediários da kd-tree depende de como será feita a distribuição dos avatares nas regiões. 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. Para calcular a coordenada de divisão, bastaria ordenar em um vetor os avatares de acordo com o eixo usado para divisão (x ou y) pelo nodo da árvore e, então, calcular o índice, no vetor, tal que o número de elementos até esse índice fosse proporcional à capacidade do filho à esquerda e o número de

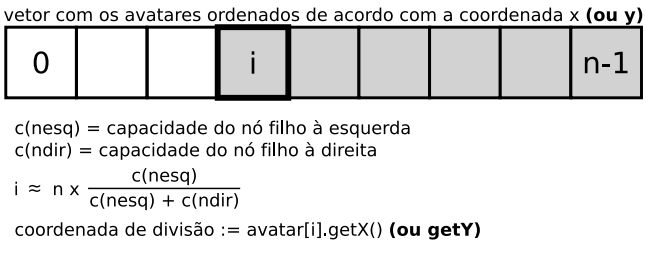

Figura 5: *Cálculo simples da coordenada de divisão*

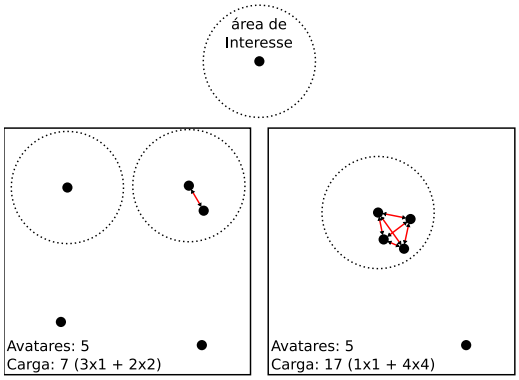

Figura 6: *Relação entre avatares e carga*

elementos a partir desse índice fosse proporcional à capacida do filho à direita do nó cuja coordenada de divisão está sendo calculada (Figura 5). A complexidade dessa operação seria $O(nlogn)$, devido à ordenação dos avatares.

Contudo, essa distribuição não funcionaria, pelo fato de que a carga imposta pelos jogadores depende também do quanto eles 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 – para estes, o crescimento do número de mensagens será linear em relação ao número de jogadores. No entanto, se esses avatares estiverem próximos um do outro, cada jogador deverá ser atualizado não apenas a respeito do resultado de suas próprias ações, como também das ações do outro jogador – neste caso, o crescimento do número de mensagens pode ser quadrático (Figura 6). Por esta razão, não é suficiente apenas dividir os jogadores entre os servidores, mesmo que proporcionalmente aos recursos de cada um destes.

Uma maneira mais adequada de se dividir o conjunto de avatares é de acordo com a carga que cada um gera sobre o servidor. O método força-bruta para calcular essas cargas seria utilizar dois laços aninhados para calcular a distância de cada avatar para cada um dos outros avatares e, baseado na interação entre cada par de avatares, chegar ao número de mensagens que cada jogador deve receber por unidade de tempo. Essa abordagem tem complexidade $O(n^2)$. Porém, se os avatares forem ordenados de acordo com a sua coordenada no eixo usado para divisão do espaço na kd-tree, pode-se realizar esse cálculo em um tempo menor.

Para isso, são também utilizados dois laços para varredura do vetor, sendo que cada um dos avatares contém uma *carga* inicializada com o valor de zero. Como o vetor está ordenado, o laço interno pode começar de um índice antes do qual sabe-se que nenhum avatar a_j terá relevância para aquele que está sendo indicado no laço externo, a_i. É utilizada uma variável $begin$, com valor inicial de zero:

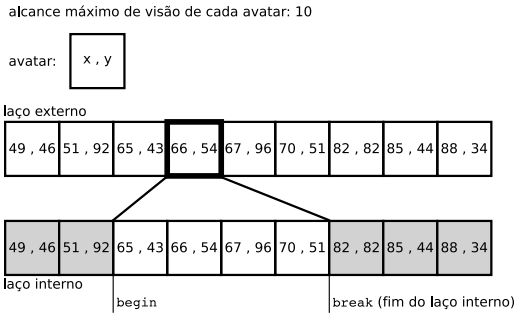

Figura 7: *Varredura do vetor ordenado de avatares*

quando a coordenada de a_j é menor que a de a_i, com uma diferença maior que o alcance máximo de visão dos avatares, a variável $begin$ é incrementada. Para cada a_j que está a uma distância menor que o alcance máximo de visão, a *carga* de a_i é incrementada segundo a relevância de a_j para a_i. Ao se chegar a um avatar a_j, tal que a sua coordenada é maior que a de a_i, com uma diferença maior que o alcance de visão, o laço externo avança imediatamente para o próximo passo, incrementando a_i e atribuindo a a_j o valor armazenado em $begin$ (Figura 7).

Sejam $width$ a extensão do ambiente virtual no eixo usado para divisão; $radius$, o alcance máximo de visão dos avatares e n, o número de avatares. O número de vezes em que é calculada a relevância de um avatar para o outro, considerando que eles estejam distribuídos uniformemente no ambiente virtual é de $O(m \times n)$, onde m é o número de avatares comparados no laço interno, ou seja, $m = \frac{2 \times radius \times n}{width}$. A complexidade da ordenação dos avatares em um dos eixos é de $O(nlogn)$. Embora ainda seja quadrático, o tempo de execução é reduzido significativamente, a depender do tamanho do ambiente virtual e do alcance da visão dos avatares. O algoritmo poderia seguir adiante e ordenar cada conjunto de avatares a_j que estão próximos (em um dos eixos de coordenadas) de a_i de acordo com o outro eixo de coordenadas e, novamente, fazer uma varredura eliminando os que estivessem distantes, agora nas duas dimensões. A complexidade do número de cálculos de relevância seria $O(p \times n)$, com p sendo o número de avatares próximos a a_i, considerando os dois eixos de coordenadas, ou seja, $p = \frac{(2 \times radius)^2 \times n}{width \times height}$. Nesse caso, $height$ é a extensão do ambiente no segundo eixo tomado como referência. Porém, embora haja uma redução considerável do número de cálculos de relevância, isso não compensa o tempo que se gasta ordenando o subvetor de avatares seleleciondos para cada a_i. Somando-se todo o tempo gasto com ordenações, seria obtida uma complexidade de: $O(nlogn + n \times mlogm)$.

Tendo sido calculada a carga imposta por cada avatar, utiliza-se esse valor para definir as cargas dos nós folhas e, recursivamente, dos nós ancestrais aos nós folha. A cada nó folha estão associados uma região no ambiente virtual e um servidor. A carga deste nó será igual ao uso de largura de banda do servidor associado para enviar atualizações de estado aos jogadores que controlam os avatares ali presentes. Dessa forma, a carga de cada nó folha será igual à soma das cargas dos avatares presentes na região definida por ele.

3.3 Balanceamento dinâmico de carga

Uma vez criada a árvore, cada servidor é associado a um nó folha – que determina uma região. Todas as atualizações de estado que devem ser enviadas a jogadores cujos avatares estejam naquela região deverão ser enviadas pelo servidor correspondente. Quando um servidor fica sobrecarregado, ele tem a possibilidade de transferir

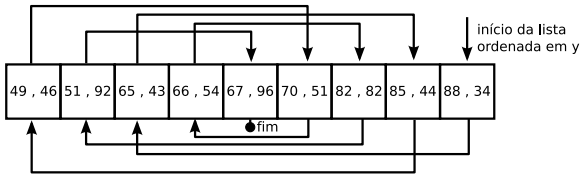

Figura 8: *Vetor ordenado em x contendo lista ordenada em y*

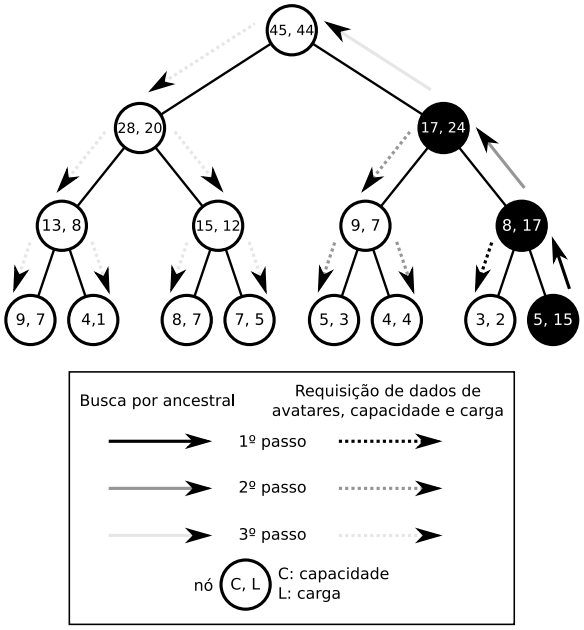

Figura 9: *Busca por um nó ancestral com capacidade suficiente*

parte da carga designada a ele para algum outro servidor. Para fazer isso, é utilizada a kd-tree, através do reajuste das coordenadas de divisão das regiões.

Cada servidor mantém um vetor dos avatares localizados na região por ele administrada, ordenados ao longo do eixo x, sendo que cada posição desse vetor contém também um apontador para um índice, formando uma lista, que é ordenada de acordo com a coordenada y dos avatares (Figura 8). Isso permite reduzir um pouco o tempo necessário para o balanceamento, diminuindo o tempo gasto com envio e recebimento de listas de avatares, além de economizar o tempo que seria gasto com ordenações no servidor que iniciou o balanceamento, já que este precisaria apenas fazer um *merge* das listas de avatares recebidas de outros servidores.

Quando o servidor sobrecarregado dispara o balanceamento, ele percorre a kd-tree a partir do nó que define sua região, subindo nos níveis da árvore e verificando se a capacidade de seu nó pai é maior ou igual à carga. Enquanto não for encontrado um nó ancestral que tenha uma capacidade maior ou igual à carga, o algoritmo continua subindo na árvore recursivamente, até chegar à raiz. A cada nó visitado, é enviada uma requisição do conjunto de avatares e dos valores de carga e capacidade aos servidores representados pelos nós folha das sub-árvores filhas desse nó (Figura 9). Dispondo desses dados, e da sua própria lista de avatares e valores de carga e capacidade, o servidor sobrecarregado pode calcular a carga e a capacidade do seu nó ancestral visitado na kd-tree.

Chegando a um nó ancestral com capacidade maior ou igual à sua carga – ou à raiz da árvore, caso não encontre –, o servidor que

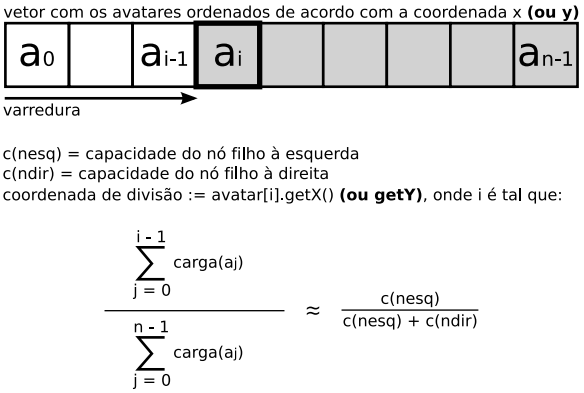

Figura 10: *Divisão do conjunto de avatares entre dois nós irmãos*

iniciou o balanceamento reajusta as coordenadas de divisão dos nós da kd-tree. Para cada nível, ele divide os avatares de acordo com a capacidade dos nós filhos. Para isso, calcula-se a fração da carga daquele nó que deve caber para cada um dos seus filhos. Percorre-se a estrutura de dados que contém os avatares ordenados até que a soma das cargas dos avatares nas posições percorridas atinja o valor definido para a carga do filho à esquerda do nó cuja coordenada de divisão está sendo calculada (Figura 10). Os filhos desse nó têm também, por sua vez, suas coordenadas de divisão reajustadas recursivamente, para que estas estejam dentro da região do nó pai. Nesse momento, já é feito um balanceamento dos nós filhos do nó de nível mais alto que teve sua coordenada de divisão reajustada.

Como os vetores de avatares recebidos dos outros servidores já contêm informação a respeito da ordem em relação a ambos os eixos de coordenadas, basta fazer um *merge* dessas estruturas de dados com a lista de avatares do servidor que iniciou o balanceamento. O tempo para reajustar toda a árvore é, no pior caso, $O(nlogS)$, onde n é o numero de avatares no jogo e S é o número de servidores. Essa complexidade se deve ao *merge* executado em cada nível da árvore – $O(n)$ –, seguido da busca seqüencial no vetor pelo índice que divide a carga dos avatares da maneira adequada – $O(n)$ – também em cada nível da kd-tree. Como se trata de uma árvore balanceada com S nós folha, tem-se uma altura de $\lceil logS \rceil$.

4 Simulações

Para avaliar a eficácia (e eficiêcia, em tempo) da proposta de balanceamento dinâmico com o uso de uma kd-tree, foi simulado um ambiente virtual com vários avatares em movimento. Partindo de um ponto aleatório no ambiente, cada avatar desloca-se segundo o modelo *random waypoint* [Bettstetter et al. 2002]. Para provocar um desbalanceamento e estressar o algoritmo definido na seção anterior, são definidos alguns *hotspots* – pontos de interesse para os quais os avatares se deslocam com maior probabilidade do que para outros pontos do mapa. Com isso, forma-se uma concentração maior de avatares ao redor dos pontos de interesse definidos. Embora o modelo de movimentação utilizado possa não ser muito realístico no que tange à maneira como os jogadores movimentam seus avatares em jogos reais, ele foi usado apenas para forçar um desbalanceamento da carga sobre os servidores. Para cada algoritmo testado, foram simulados dois tipos de situações: uma com a presença de hotspots e outra sem.

São feitas comparações com as abordagens de balanceamento de carga pra MMOGs apresentadas na seção 2. Porém, deve ser feita a observação de que o modelo empregado por [Ahmed e Shirmohammadi 2008] considera células hexagonais, enquanto na simulação

foram utilizadas células retangulares. Além disso, os autores consideraram que existe um limite para a taxa de envio de mensagens e que este limite é o mesmo para todos os servidores. Como aqui considera-se um sistema heterogêneo, o algoritmo foi simulado considerando o limite da taxa de envio de mensagens como sendo diferente para cada servidor, a depender da largura de banda de envio de cada um. Contudo, foi mantido o que se considera a idéia central da abordagem, que é a seleção do menor agrupamento de células gerenciadas pelo servidor sobrecarregado, seguida pela seleção daquela que tem a menor interação com outras células do mesmo servidor e, finalmente, a transferência dessa célula para o servidor menos carregado. Além deste algoritmo, foram testados os algoritmos Progrega e BFBCT, apresentados em [Bezerra e Geyer 2009].

O ambiente virtual simulado consistia de um espaço bidimensional, por onde transitavam 750 avatares, cujos jogadores estavam divididos entre 8 servidores ($S_1, S_2, ..., S_8$), cada um dos quais associados a uma das regiões determinadas pelo algoritmo de balanceamento. No caso da simulação das abordagens baseadas em células, considerou-se que o espaço estava dividido em uma grade de células de ordem 15, ou seja, o ambiente estava dividido em 225 células. A capacidade de cada servidor S_i era igual a $i \times 20000$, criando um sistema heterogêneo. Isso permitiria avaliar os algoritmos de balanceamento de carga simulados segundo o critério de proporcionalidade da distribuição de carga entre os servidores.

Além de avaliar os algoritmos segundo a proporcionalidade da distribuição da carga, foi considerado também o número de vezes em que haveria transferência de jogadores entre servidores. Cada uma dessas transferências implica uma conexão do jogador ao novo servidor e uma desconexão do antigo. Esse tipo de situação pode ocorrer em dois casos: o avatar se moveu, mudando de região e, conseqüentemente, mudando o jogador de servidor; ou o jogador estava com seu avatar parado e teve de mudar de servidor. Neste último caso, obviamente a transferência do jogador ocorreu devido a um rebalanceamento. Um algoritmo de balanceamento ideal realiza a redistribuição de carga necessitando do mínimo possível de transferências de jogadores entre servidores, ao mesmo tempo em que mantém a carga sobre cada servidor proporcional à sua capacidade.

Por último, será avaliado também o sobrecusto de comunicação entre servidores, que ocorre quando dois jogadores estão interagindo, porém cada um conectado a um servidor diferente. Embora o algoritmo de balanceamento proposto neste trabalho não busque atacar diretamente esse problema, é interessante avaliar o quanto a distribuição de carga efetuada por ele influencia na comunicação entre os servidores.

5 Resultados

As figuras 11 e 12 apresentam a carga média (mais o overhead da comunicação entre servidores) sobre cada servidor, para cada algoritmo testado. A primeira figura mostra esses valores em uma situação sem aglomerações de jogadores e, portanto, com menor carga total. A segunda, por sua vez, mostra a distribuição da carga em uma situação de sobrecarga do sistema servidor. Observa-se, na figura 11, que todos os algoritmos testados cumpriram com o objetivo de manter a carga sobre cada servidor menor ou igual à sua capacidade, quando o sistema dispõe de recursos suficientes para tal. Na figura 12, que representa uma situação de sobrecarga do sistema, percebe-se que todos os algoritmos conseguiram diluir de maneira aproximadamente proporcional a sobrecarga entre os servidores.

Nas figuras 13 e 14, é apresentado o quanto o balanceamento gerado por cada algoritmo desvia de um balanceamento ideal – isto é, o quanto, em média, a carga sobre os servidores desviam do valor exatamente proporcional à sua capacidade – ao longo do tempo. O

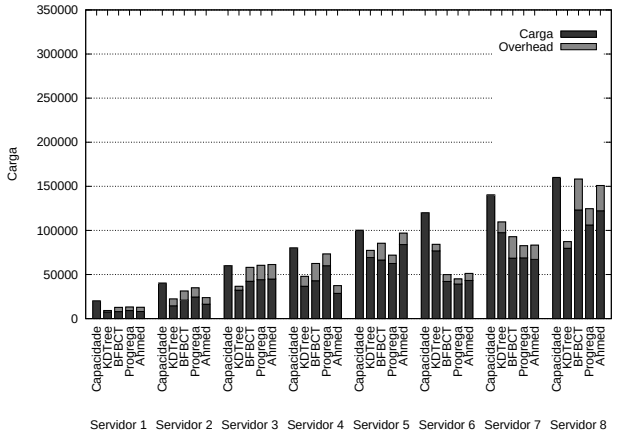

Figura 11: *Carga média em cada servidor (por algoritmo, sem hotspots)*

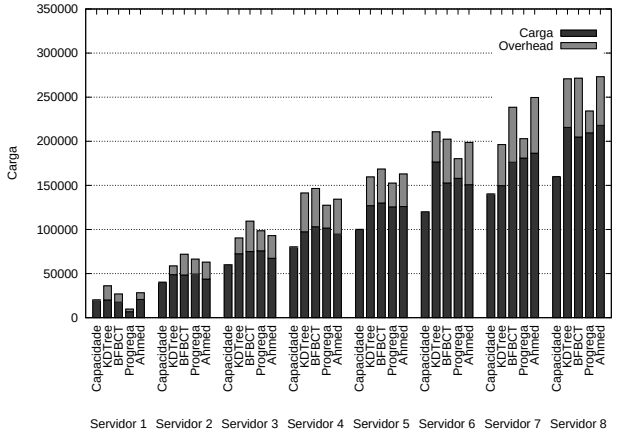

Figura 12: *Carga média em cada servidor (por algoritmo, com hotspots)*

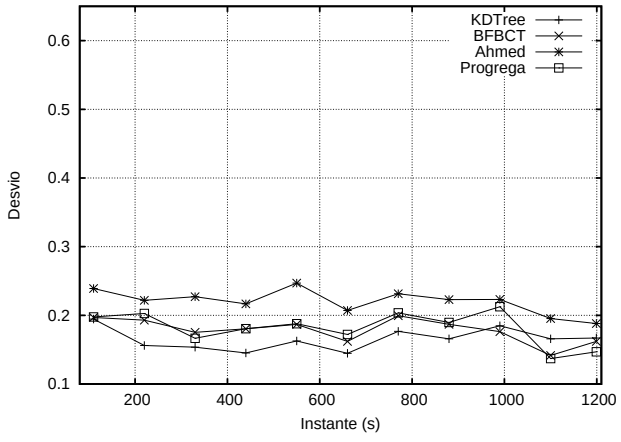

Figura 13: *Desvio médio da taxa de uso ideal dos servidores (sem hotspots)*

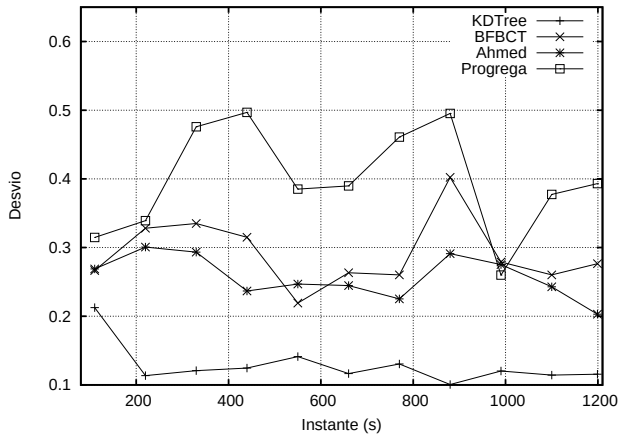

Figura 14: *Desvio médio da taxa de uso ideal dos servidores (com hotspots)*

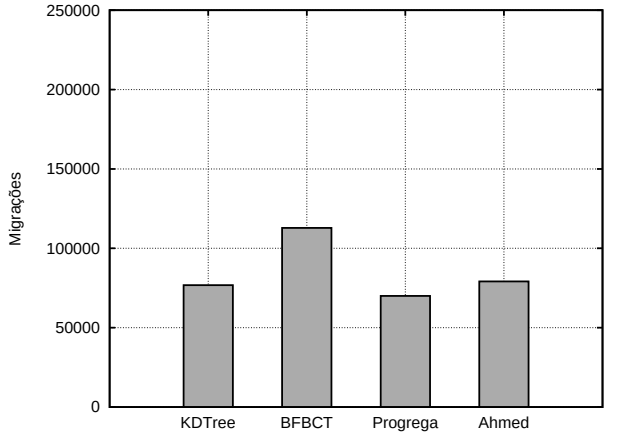

Figura 15: *Migrações de jogadores entre servidores (sem hotspots)*

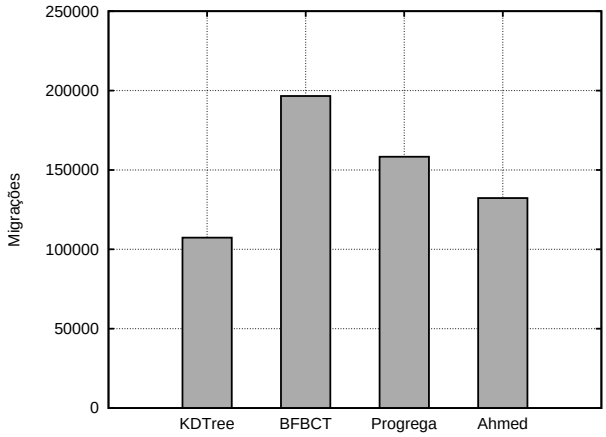

Figura 16: *Migrações de jogadores entre servidores (com hotspots)*

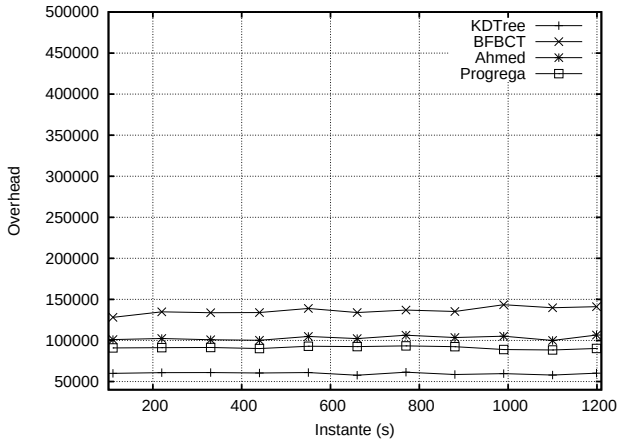

Figura 17: *Comunicação entre servidores para cada algoritmo ao longo do tempo (sem hotspots)*

que se descobre é que, em ambas as situações – com e sem hotspots – o algoritmo que utiliza a kd-tree tem o menor desvio. Isso se deve à granularidade mais fina da sua distribuição, que, diferentemente das outras abordagens testadas, não é limitada por um tamanho de célula. Na situação com hotspots, o algoritmo que utiliza a kd-tree é especialmente eficaz, pelo fato de balanceamentos serem necessários. Numa situação em que o sistema dispõe de muito mais recursos que o necessário, a proporcionalidade da distribuição não é tão importante, bastando que cada servidor administre uma carga inferior à sua capacidade.

No quesito migrações de jogadores, todos os algoritmos testados – exceto o BFBCT – causaram um número semelhante de migrações de usuários na ausência de hotspots (Figura 15). Isso se deve ao fato da carga do jogo ser inferior à capacidade total do sistema servidor, o que exigia poucos rebalanceamentos e, consequentemente, poucas migrações de jogadores entre servidores. Na Figura 16, porém, percebe-se que o algoritmo que utiliza a kd-tree teve um número significativamente menor de migrações de usuários do que as outras abordagens. Isso se deve, primeiro, ao fato de que as regiões definidas pelos nós folhas da kd-tree são necessariamente contíguas, e cada servidor estava associado a apenas um desses nós folha. Um avatar movendo-se por um ambiente com regiões muito fragmentadas atravessaria constantemente fronteiras entre essas regiões e faria, consequentemente, com que seu jogador trocasse de servidor repetidas vezes. Outra razão para este resultado é que cada rebalanceamento de carga executado com a kd-tree aproxima bastante a carga sobre os servidores do valor ideal, exigindo menos rebalanceamentos futuros e, consequentemente, causando menos migrações de jogadores do que nas abordagens com células.

Por fim, é mostrada a comunicação entre servidores, para cada algoritmo simulado, ao longo do tempo. Na Figura 17, todos os algoritmos têm um resultado semelhante, sendo que o que utiliza a kd-tree tem um pouco menos de overhead do que os outros. Isso também se explica pelo fato das regiões serem contíguas, minimizando o número de fronteiras entre as mesmas e, consequentemte, reduzindo a probabilidade de haver interações entre pares de avatares em que cada um está em uma região diferente. Na Figura 18, percebe-se que o algoritmo Progrega causou uma comunicação entre servidores significativamente menor que os outros em uma situação de sobrecarga. A razão para isto é que o objetivo principal – além de balancear a carga – do Progrega é de justamente reduzir a comunicação entre servidores. No entanto, mesmo não considerando esse sobrecusto, o algoritmo que utiliza a kd-tree ficou em segundo lugar nesse critério.

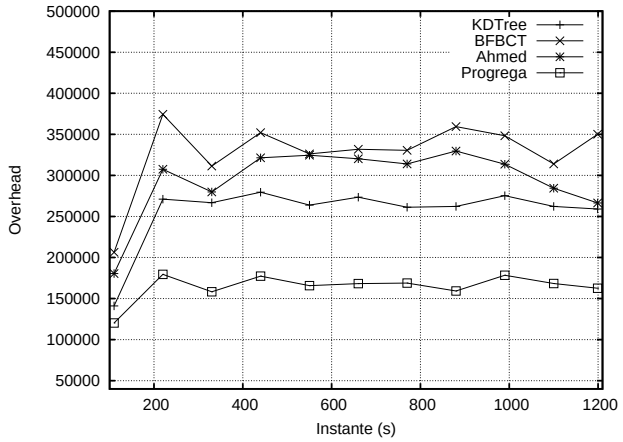

Figura 18: *Comunicação entre servidores para cada algoritmo ao longo do tempo (com hotspots)*

6 Conclusões

Neste trabalho, foi proposto o uso de kd-trees para particionar o ambiente virtual de MMOGs e permitir o balanceamento de carga dos servidores com ajustes recursivos das coordenadas de divisão armazenadas nessa estrutura de dados. Uma das conclusões a que se chegou foi que o uso de kd-trees para fazer este particionamento permite uma granularidade fina na distribuição do espaço, ao mesmo tempo em que o reajuste das partições se torna mais simples – percorrendo recursivamente a árvore – do que as abordagens mais comuns, baseadas em células e/ou particionamento de grafos.

Essa granularidade mais fina permite um balanceamento melhor, fazendo com que a carga designada a cada servidor seja mais próxima do valor ideal que deveria delegado a cada um. Essa melhor balanceamento também permitiu reduzir o número de migrações, ao se fazer menos operações repetidas de rebalanceamento. O fato de que as regiões definidas pela kd-tree são necessariamente contíguas foi um dos fatores que contribuiu para os bons resultados do algoritmo proposto, tendo sido melhor que os outros algoritmos simulados na maior parte das medições realizadas.

Para concluir, foi possível utilizar métodos que podem reduzir a complexidade de cada operação de rebalanceamento. Isso se deve, em primeiro lugar, à diminuição do número de operações de cálculo da relevância entre os pares de avatares ao se fazer uma varredura da lista de avatares ordenados de acordo com sua posição no espaço e, em segundo lugar, ao se manter em cada servidor a lista de avatares já ordenada em ambas as dimensões, reduzindo o tempo de envio e economizando o tempo que seria gasto com ordenação dos avatares quando estes fossem recebidos pelo servidor que disparou o rebalanceamento.

Agradecimentos

O desenvolvimento deste trabalho foi apoiado pelo Conselho Nacional de Pesquisa (CNPq) e pela Coordenação de Aperfeiçoamento do Pessoal de Ensino Superior (CAPES).

Referências

AHMED, D., E SHIRMOHAMMADI, S. 2008. A Microcell Oriented Load Balancing Model for Collaborative Virtual Environments. In *Proceedings of the IEEE Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems, VECIMS*, Piscataway, NJ: IEEE, Istanbul, Turkey, 86–91.

ASSIOTIS, M., E TZANOV, V. 2006. A distributed architecture for MMORPG. In *Proceedings of the ACM SIGCOMM workshop on Network and system support for games, NetGames, 5.*, New York: ACM, Singapore, 4.

BENTLEY, J. 1975. Multidimensional binary search trees used for associative searching.

BETTSTETTER, C., HARTENSTEIN, H., E PÉREZ-COSTA, X. 2002. Stochastic Properties of the Random Waypoint Mobility Model: epoch length, direction distribution, and cell change rate. In *Proceedings of the ACM international workshop on Modeling analysis and simulation of wireless and mobile systems, 5.*, New York: ACM, Atlanta, GA, 7–14.

BEZERRA, C. E. B., E GEYER, C. F. R. 2009. A load balancing scheme for massively multiplayer online games. *Massively Multiuser Online Gaming Systems and Applications, Special Issue of Springer's Journal of Multimedia Tools and Applications*.

BEZERRA, C. E. B., CECIN, F. R., E GEYER, C. F. R. 2008. A3: a novel interest management algorithm for distributed simulations of mmogs. In *Proceedings of the IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, DS-RT, 12.*, Washington, DC: IEEE, Vancouver, Canada, 35–42.

BLIZZARD, 2004. World of warcraft. 2004. Disponível em: <http://www.worldofwarcraft.com/>. Acesso em: 26 jun. 2009.

CHERTOV, R., E FAHMY, S. 2006. Optimistic Load Balancing in a Distributed Virtual Environment. In *Proceedings of the ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV, 16.*, New York: ACM, Newport, USA, 1–6.

EL RHALIBI, A., E MERABTI, M. 2005. Agents-based modeling for a peer-to-peer MMOG architecture. *Computers in Entertainment (CIE) 3*, 2, 3–3.

GRAVITY, 2001. Raganarök online. 2001. Disponível em: <http://www.ragnarokonline.com/>. Acesso em: 26 jun. 2009.

HAMPEL, T., BOPP, T., E HINN, R. 2006. A peer-to-peer architecture for massive multiplayer online games. In *Proceedings of the ACM SIGCOMM workshop on Network and system support for games, NetGames, 5.*, New York: ACM, Singapore, 48.

IIMURA, T., HAZEYAMA, H., E KADOBAYASHI, Y. 2004. Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games. In *Proceedings of the ACM SIGCOMM workshop on Network and system support for games, NetGames, 3.*, New York: ACM, Portland, USA, 116–120.

KNUTSSON, B., ET AL. 2004. Peer-to-peer support for massively multiplayer games. In *Proceedings of the IEEE Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM, 23.*, [S.l.]: IEEE, Hong Kong, 96–107.

LEE, K., E LEE, D. 2003. A scalable dynamic load distribution scheme for multi-server distributed virtual environment systems with highly-skewed user distribution. In *Proceedings of the ACM symposium on Virtual reality software and technology*, New York: ACM, Osaka, Japan, 160–168.

LUQUE, R., COMBA, J., E FREITAS, C. 2005. Broad-phase collision detection using semi-adjusting BSP-trees. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM New York, NY, USA, 179–186.

NCSOFT, 2003. Lineage ii. 2003. Disponível em: <http://www.lineage2.com/>. Acesso em: 26 jun. 2009.

NG, B., ET AL. 2002. A multi-server architecture for distributed virtual walkthrough. In *Proceedings of the ACM symposium on Virtual reality software and technology, VRST*, New York: ACM, Hong Kong, 163–170.

RIECHE, S., ET AL. 2007. Peer-to-Peer-based Infrastructure Support for Massively Multiplayer Online Games. In *Proceedings of the 4th IEEE Consumer Communications and Networking Conference, CCNC, 4.*, [S.l.]: IEEE, Las Vegas, NV, 763–767.

SCHIELE, G., ET AL. 2007. Requirements of Peer-to-Peer-based Massively Multiplayer Online Gaming. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid, CCGRID, 7.*, Washington, DC: IEEE, Rio de Janeiro, 773–782.