Balanceamento de carga utilizando kd-trees para particionar dinamicamente o ambiente virtual 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 a ela associado 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 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 1, e cada região é delegada a um dos servidores. Quando um de-

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

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

les 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 – considerando apenas células gerenciadas pelo servidor sobrecarregado que disparou o balanceamento e 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, sendo o valor da interação entre células está relacionado ao número de pares de avatares interagindo, cada um deles em uma célula diferente. 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].

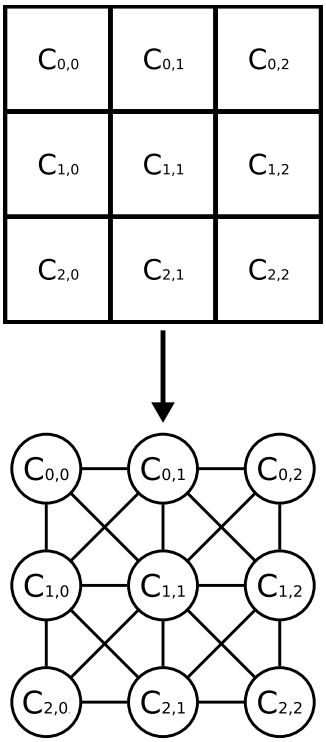

Figura 2: *Representação do ambiente em um grafo*

Esse tipo de estrutura de dados costuma ser usado na computação gráfica. No entanto, como em MMOGs também há informação geométrica – como a posição de cada avatar no espaço do ambiente virtual –, 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

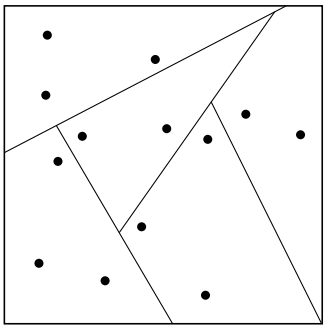

Figura 3: *Particionamento do espaço com uma árvore BSP*

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 nodo da árvore corresponde uma região do espaço e, além disso, neste nodo é armazenada um valor correspondente a uma coordenada de divisão. Cada um dos dois filhos daquele nodo representarão uma subdivisão da região representada pelo nodo 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 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 uma potência de 2). Para isto, foi utilizada a função recursiva exibida no Algoritmo 1 a para criação da árvore:

No algoritmo acima, o id serve para calcular quantos filhos cada

Algoritmo 1 nodo::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_nodo();
  $filho\_menor.pai \leftarrow$ **this**;
  $filho\_maior \leftarrow$ novo_nodo();
  $filho\_maior.pai \leftarrow$ **this**;
  $filho\_menor$.constrói_árvore
  $(id, nível + 1, num\_servidores)$;
  $filho\_maior$.constrói_árvore
  $(id + 2^{nível}, nível + 1, num\_servidores)$;
fim se

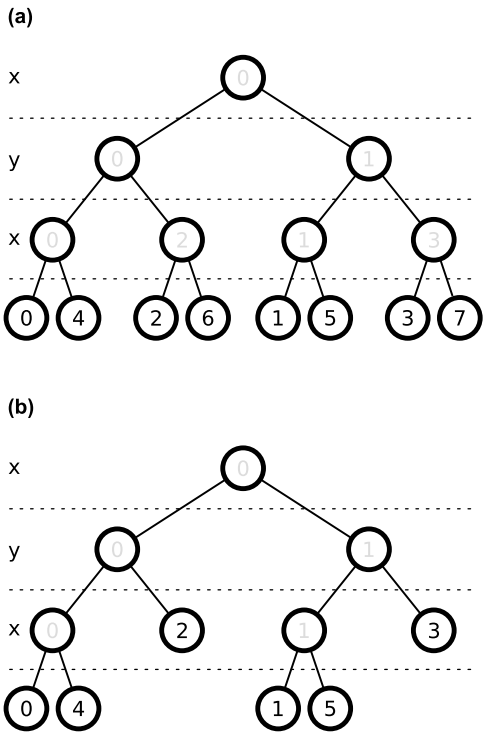

Figura 4: *Construção de kd-tree balanceada*

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 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 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 dependem 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

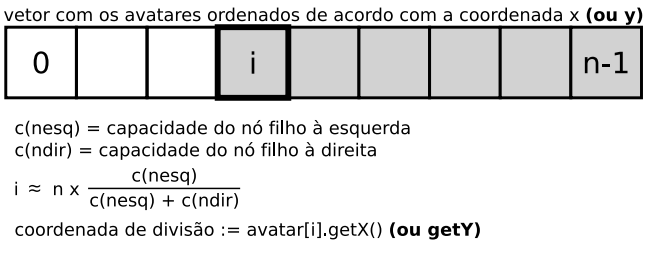

Figura 5: *Cálculo simples da coordenada de divisão*

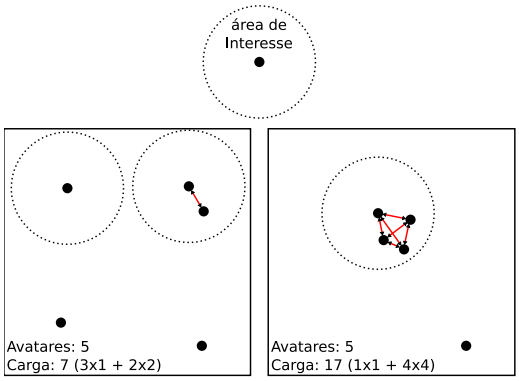

Figura 6: *Relação entre avatares e carga*

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 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

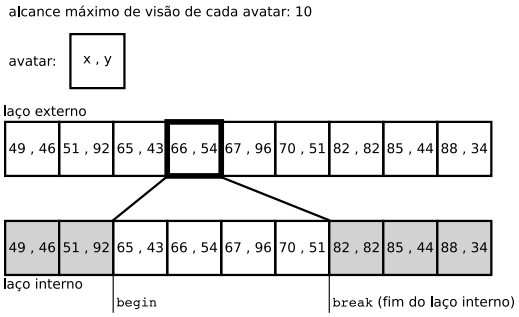

Figura 7: *Varredura do vetor ordenado de avatares*

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: 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 seleleciona dos 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 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 ordenação no nodo que iniciou o balanceamento.

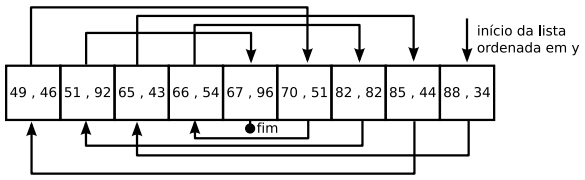

Figura 8: *Vetor ordenado em x contendo lista ordenada em y*

Quando o servidor sobrecarregado dispara o balanceamento, ele percorre a kd-tree a partir do nó que define sua região, subindo nos nível 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.

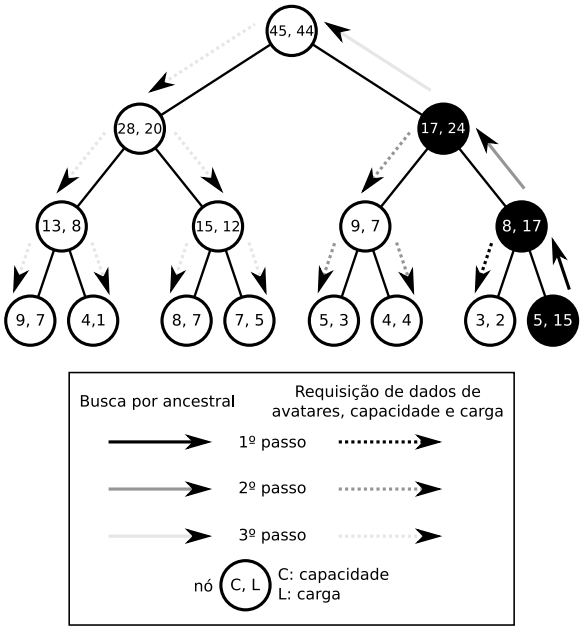

Figura 9: *Busca por um nó ancestral com capacidade suficiente*

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

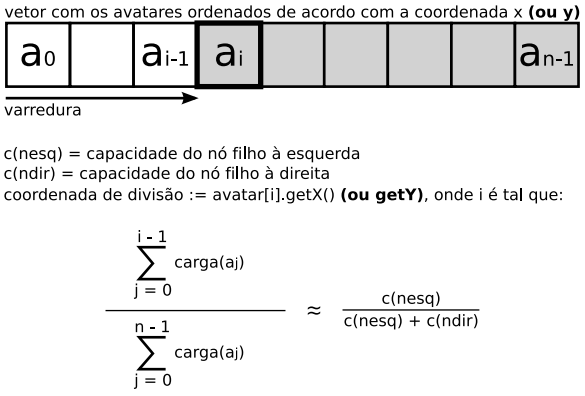

Figura 10: *Divisão do conjunto de avatares entre dois nós irmãos*

Como os vetores de avatares recebidos dos outros servidores já estão ordenados em 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, no máximo, $\lceil logS \rceil$.

4 Simulações

5 Resultados

6 Conclusões

Acknowledgements

To Robert, for all the bagels.

Referências

AHMED, D., AND 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., AND 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.

BEZERRA, C. E. B., AND 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., AND 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., AND 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., AND 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., AND 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., AND 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., AND 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., AND 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.