MODELOS DE SUPORTE DISTRIBUÍDO PARA JOGOS ONLINE MACIÇAMENTE MULTIJOGADOR

Carlos Eduardo Benevides Bezerra*

Fábio Reis Cecin*

Cláudio Fernando Resin Geyer *

***Universidade Federal do Rio Grande do Sul**

{carlos.bezerra, fcecin, geyer}@inf.ufrgs.br

Abstract

Massively multiplayer online games require a large amount of resources from the server, when using a centralized architecture to provide support to them. In this work, we present a few distribution models for these games, in order to relieve the server from such costs, making it possible for small companies and independent game development groups to release such a game, for example.

Resumo

Jogos online maciçamente multijogador requerem grande quantidade de recursos do servidor, quando se utiliza uma arquitetura centralizada para lhes prover suporte. Neste trabalho, são apresentados alguns modelos de distribuição para esses jogos, visando a desonerar o servidor de tais custos, tornando possível a disponibilização de um jogo desta natureza por empresas de menor porte e/ou grupos independentes de desenvolvimento de jogos, por exemplo.

Palavras-chave: balanceamento, MMOGs, p2p, trapaça.

Introdução

1. MMOGs

Jogos online maciçamente multijogador (ou MMOGs) têm se popularizado bastante nos últimos tempos. Além de poderem ser jogados online, permitem a interação simultânea de um grande número de participantes. Nos casos de maior sucesso, como *World of Warcraft* (BLIZZARD, 2004), por exemplo, é dado suporte a uma base de dezenas a centenas de milhares de jogadores simultâneos (CHEN e MUNTZ, 2006). Estes jogadores podem, então, interagir entre si e com o ambiente virtual do jogo.

Geralmente, cada jogador controla uma entidade chamada de *avatar*, que executa as ações determinadas por ele, interferindo na história e no encaminhamento do jogo. Cada ação executada por cada avatar deve ser processada pelo servidor, que calculará o estado resultante dessa ação no jogo. Esse novo estado é então difundido para os outros jogadores. No entanto, percebe-se que tal mecanismo pode facilmente saturar a do servidor. Geralmente o que se faz é montar uma infra-estrutura com conexão à Internet de algumas centenas ou milhares de MBps (FENG, 2007), de onde vem o maior custo dos MMOGs.

É aí que surge a idéia de utilizar abordagens descentralizadas, com o propósito de desonerar o servidor do jogo. Diversas abordagens foram propostas para prover um suporte distribuído – e, portanto, mais barato – para jogos MMOG. Porém, para evitar que o custo do sistema como um todo seja semelhante ao custo de um servidor tradicional, é necessário fazer algumas otimizações, que é um dos objetivos dos modelos que serão apresentados aqui. Além disso, pelo fato de não haver um único árbitro central para decidir o encaminhamento do jogo e verificar a existência de trapaça, foi criado um modelo de distribuição que visa a mitigar a possibilidade de violações das regras por parte dos jogadores passarem desapercebidas pelo sistema. Nas próximas seções serão apresentados alguns dos conceitos utilizados e os modelos propostos, com resultados obtidos.

2. Trapaça em MMOGs

Jogos online são atividades competitivas, o que os diferencia de trabalhos em ambientes virtuais colaborativos (CVEs) (ROBERTS e WOLFF, 2004). Em um CVE, não é um problema distribuir o estado do mundo virtual entre as máquinas dos participantes. Por exemplo, pode-se deixar a cargo da máquina de cada participante

a função de manter e a de determinar o estado do próprio avatar. Em outras palavras, cada participante tem a *autoridade final* sobre parte do estado do mundo virtual – no exemplo, a autoridade é sobre o estado do seu próprio avatar. Este tipo de solução geralmente é mais simples de ser suportado e resulta em um aumento da responsividade do sistema. Por exemplo, quando um participante interage com dispositivos de entrada (teclado, mouse), o seu processo simulador local pode mover a versão local do avatar do jogador imediatamente, já que possui autoridade para tal, e só então enviar o movimento realizado pela rede, para que os outros participantes percebam o seu comando. O sistema resultante é responsivo pois cada participante tem a sensação de controlar o seu avatar sem latência significativa entre a realização do comando e a visualização do resultado do comando.

Tais soluções simples, que delegam partes do estado do mundo virtual para as máquinas dos participantes de forma indiscriminada, são vulneráveis à trapaça. Por exemplo, em um jogo de tiro, onde os jogadores disparam projéteis uns contra os outros com o objetivo de "matar" os avatares adversários, um jogador que arbitra sobre o estado do seu próprio avatar pode simplesmente decidir que o seu avatar nunca "morre". Para isso, basta ele alterar o seu programa para simplesmente não receber mensagens de tiro oriundas de máquinas remotas, por exemplo. Este jogador está *trapaceando*, pois, no jogo online competitivo, ele quebra as regras do jogo e efetivamente obtém uma clara vantagem sobre os seus adversários. Se o sistema possui este tipo de vulnerabilidade, e não provê métodos para detectar jogadores que exploram estas vulnerabilidades, o aspecto competitivo do jogo é anulado – que sentido há em tentar derrotar adversários que podem programaticamente decidir se são derrotados ou não?

Em jogos online comerciais, o problema da trapaça é tratado primeiramente pela adoção da arquitetura cliente-servidor (KABUS et al., 2005). A maioria dos jogos online comerciais são sistemas cliente-servidor, centralizados, com pouquíssimas exceções (GAUTHIERDICKEY et al., 2004). Neste modelo, o servidor, que é mantido por uma parte confiável, é responsável pela maioria das tarefas que seriam passíveis de trapaça se delegadas para partes não-confiáveis. Por exemplo, o servidor é responsável por manter o estado corrente, oficial do jogo. Além disso, o servidor é responsável por realizar transições de estado, isto é, de aplicar as regras do jogo para atualizar o estado do jogo periodicamente, movendo objetos, resolvendo colisões, etc. Desta forma, através de uma escolha de arquitetura, os

jogos online comerciais evitam a maior parte das trapaças. Outras trapaças não tratadas implicitamente pela arquitetura cliente-servidor são resolvidas através de módulos que detectam jogadores trapaceiros e os removem dos servidores (KABUS et al., 2005; WEBB e SOH, 2007).

Apesar da sua simplicidade de implementação em software, e da sua popularidade, o modelo cliente-servidor possui algumas desvantagens. No contexto de jogos massivos, onde milhares de máquinas clientes (jogadores) precisam conectar ao servidor, este se torna um gargalo de processamento. Além disso, o custo de comunicação imposto sobre a infra-estrutura servidora é significativo, podendo a manutenção do serviço de comunicação chegar a milhões de dólares mensais no caso de jogos massivos cliente-servidor (FENG, 2007). Isto torna a barreira de entrada muito alta para entidades com menor poder financeiro, como pequenas empresas ou grupos de pesquisa.

Neste contexto, muita pesquisa tem sido dedicada para suporte descentralizado a jogos massivos. Por exemplo, tem-se tentado utilizar o paradigma *par-a-par*, que já é utilizado para dar suporte a sistemas descentralizados de troca de arquivos, para suportar jogos massivos. Porém, muitos modelos par-a-par de suporte a jogos massivos já propostos tem apresentado problemas de trapaça que são uma conseqüência direta do abandono do paradigma cliente-servidor. Estas vulnerabilidades se somam a outras já presentes mesmo em soluções centralizadas. Muitos modelos par-a-par de suporte a jogos massivos simplesmente delegam tarefas críticas, que requerem uma parte confiável para a sua execução, para as máquinas dos jogadores. Como mencionado anteriormente, este tipo de solução é geralmente passível de trapaça.

2.1 Tratando seletivamente de trapaças

Existem diversas maneiras de se trapacear em jogos online, e já existem até taxonomias de trapaças (YAN e RANDELL, 2005). Como se tem visto na prática, em jogos online cliente-servidor, é muito difícil projetar um sistema de jogo que seja imune a todos os tipos concebíveis de trapaças. Se torna ainda mais difícil tratar todas as trapaças possíveis em uma arquitetura par-a-par (descentralizada) de suporte a jogos online massivos pois, neste contexto, é necessário confiar em muito mais resultados de computação fornecidos pelas máquinas dos jogadores, que estão ativamente buscando oportunidades para obterem vantagem fácil no jogo.

Portanto, torna-se necessário tratar do problema de forma seletiva, priorizando-se algumas trapaças sobre outras. Uma maneira de priorizar é ignorar trapaças irrelevantes para um determinado tipo de jogo: é evidente que cada vulnerabilidade possui um impacto que depende principalmente da aplicação. Para validar esta afirmação, basta considerarmos um caso limítrofe: o suporte a ambientes virtuais colaborativos (CVEs). Um CVE pode ser visto como um jogo online que, porém, não possui competição. Neste contexto, não é possível um participante obter vantagem sobre os outros participantes, simplesmente porque não há uma semântica de "vantagem" em um ambiente colaborativo. De forma similar, em um jogo online de onde não é possível extrair uma vantagem sobre os adversários através da manipulação da posição de avatares, não é um problema delegar a tarefa de cálculo e disseminação da posição dos avatares para as máquinas dos respectivos jogadores.

Quando não pode-se simplesmente ignorar um tipo de trapaça, visto que o seu impacto não é nulo, a nossa abordagem é a de avaliar o impacto relativo de cada vulnerabilidade. Mais especificamente, nós buscamos dedicar uma maior quantidade de recursos para tratar de trapaças de maior impacto. Por exemplo, pode-se utilizar um algoritmo mais custoso, que emite mais mensagens e que realiza mais processamento, para prevenir ou detectar trapaças de alto impacto, ao passo em que emprega-se um algoritmo mais simples e menos custoso, com menor taxa de detecção por exemplo, para tratar de trapaças de menor impacto.

A solução para um determinado tipo de trapaça não precisa ser algorítmica. Por exemplo, uma maneira simples, e perfeitamente válida, para se tratar de trapaças em uma arquitetura par-a-par para jogos massivos consiste em propor que o estado do jogo seja mantido e atualizado por um conjunto reduzido de super-pares (*super-peers*) cujos administradores confiam mutuamente uns nos outros. Este tipo de solução, porém, precisa ser avaliado não só dos pontos de vista tradicionais, como escalabilidade, como também deve ter seus aspectos não-funcionais considerados. Por exemplo, deve-se considerar se o modelo de confiança é escalável: qual a dificuldade de se estabelecer uma rede de confiança entre 10, 100 ou 1000 super-pares? Quantos super-pares são necessários para se suportar uma determinada quantidade de jogadores?

2.2 Economias virtuais e a trapaça de fabricação de estado (state cheating)

Um jogo massivo é, em princípio, um ambiente virtual online onde um número elevado de participantes podem interagir simultaneamente, em tempo real, para modificar um estado de jogo que é persistido em longo prazo. Na prática, existe apenas um tipo específico de jogo que se popularizou sobre este paradigma massivo, que são os *Role Playing Games* (RPGs) massivos (MMORPGs). Em um MMORPG frequentemente existe uma economia virtual. Os jogadores acumulam bens virtuais ("itens"), que podem ser comprados ou vendidos através de uma moeda virtual, que por sua vez também pode ser acumulada. Desde os primeiros MMORPGs sabe-se que a manutenção destas economias virtuais é vital para o sucesso de um MMORPG (THOMPSON, 2000). As economias virtuais são tão significativas para os jogadores de MMORPGs que os mesmos estão dispostos a comprar itens virtuais utilizando dinheiro real. Em 2007, estimou-se que a compra e venda de riquezas de economias virtuais movimentou o equivalente a US$ 2 bilhões (LEHTINIEMI e LEHDONVIRTA, 2007).

O estado atual de uma economia virtual, isto é, a quantidade de riqueza atualmente alocada para cada jogador, faz parte do estado de um jogo massivo, da mesma maneira que outras variáveis como a posição corrente de cada avatar. Porém, nós entendemos que proteger a economia virtual de um jogo massivo contra trapaças deve ser uma prioridade. A maioria, senão todos, os MMORPGs comerciais atuais são sistemas cliente-servidor, e a arbitragem sobre o estado da economia virtual é sempre realizado por máquinas servidoras, sob o controle centralizado do operador do serviço. Desta forma, o estado das economias virtuais não pode ser manipulado por jogadores, salvo devido a defeitos no software servidor ou no protocolo (MCGRAW e HOGLUND, 2007) que são passíveis de conserto.

A trapaça de fabricação de estado, ou *state cheating*, consiste em uma vulnerabilidade, especificamente encontrada em arquiteturas descentralizadas de suporte a jogos massivos, onde um jogador é capaz de alterar o estado do jogo de forma arbitrária. Por exemplo, um jogador é capaz de determinar um valor arbitrário para a quantidade de moeda virtual que possui, ou este é capaz de acrescentar uma quantidade arbitrária de itens virtuais quaisquer ao estado do jogo. Visto que o conteúdo da memória na máquina de um jogador ou de um *peer* anônimo na Internet não pode ser garantido, deve-se portanto ter cuidado ao delegar autoridade sobre

partes do estado do jogo, especialmente quando as partes do estado se referem a uma economia virtual. Um esquema de distribuição simples, onde cada jogador determina localmente, sem supervisão, qual a quantidade de riqueza virtual que possui, é vulnerável a *state cheating*. Este sistema, portanto, não é viável para, ao menos, suportar um MMORPG cuja jogabilidade depende de uma economia virtual funcional.

O problema de *state cheating* pode ser tratado de várias maneiras. Como mencionado anteriormente, ele pode não ser relevante dependendo do tipo de jogo a ser suportado. Por exemplo, pode-se estar planejando suportar jogos de tiro em primeira pessoa massivos, que não são atualmente suportados em escala massiva – com algumas poucas exceções, como o jogo PlanetSide (SONY ONLINE ENTERTAINMENT, 2008). Pode-se projetar jogos de tiro massivos que não dependem de economias virtuais, ou onde uma economia virtual possui um papel secundário. Outra solução é utilizar arquiteturas híbridas, que combinam o cliente-servidor com o par-a-par. Neste modelo, é possível manter o estado da economia virtual exclusivamente em máquinas servidoras mantidas por um operador central do jogo, ao passo que outras tarefas são descentralizadas em uma rede par-a-par a ser formada a partir das máquinas e das conexões Internet dos jogadores. Ainda outra solução seria a adoção de uma rede de super-pares, onde a honestidade dos super-pares é garantida implicitamente, como mencionado em exemplo anterior. Estes super-pares, então, poderiam concentrar a atividade de manter a economia virtual e outras partes do estado do jogo que não podem ser manipuladas por jogadores.

Por fim, também pode-se prevenir *state cheating* utilizando-se replicação (GAUTHIERDICKEY et al., 2004). Em arquiteturas baseadas em replicação, o estado do jogo é dividido em células, e cada célula é replicada em vários nós que, em princípio, não são confiáveis. A confiabilidade emerge na medida em que um nó, sozinho, não pode alterar o estado do jogo de forma arbitrária: ele pode apenas alterar a sua cópia local. Um observador externo qualquer de uma célula, seja um jogador ou um servidor, sempre irá perceber (ler) o estado da célula como um consenso entre as várias versões da célula. Partindo-se do princípio que a maioria dos participantes ou são honestos ou, ao menos, não estão trapaceando de forma massivamente coordenada, tem-se que um algoritmo de consenso (por exemplo, consenso bizantino) irá, na maioria absoluta dos casos, retornar uma versão do estado que é condizente com as regras do jogo.

Desenvolvimento

3. O modelo FreeMMG 2

O FreeMMG 2 é uma arquitetura híbrida, par-a-par e cliente-servidor, de suporte a jogos massivos. O modelo visa suportar jogos massivos com economias virtuais, portanto, tratar de *state cheating* se torna essencial. Como, no nosso entendimento, *state cheating* é a trapaça que possui maior impacto em um jogo massivo baseado em uma economia virtual, nós projetamos a arquitetura primeiramente pensando em como prevenir esta trapaça. Uma abordagem preventiva, ao contrário de uma abordagem de detecção, dispensa o desenvolvimento de um algoritmo que desfaça alterações ilegais ao estado do jogo. Pois, é evidente que, após a detecção de um jogador que altera ilegalmente o estado do jogo (a economia) de forma arbitrária, estas alterações devem ser desfeitas, pois a injeção discriminada de riqueza em uma economia pode destruir o valor dos bens (THOMPSON, 2000). Isto pode ocorrer mesmo quando os jogadores que realizaram tais injeções ilegais são identificados e removidos do jogo.

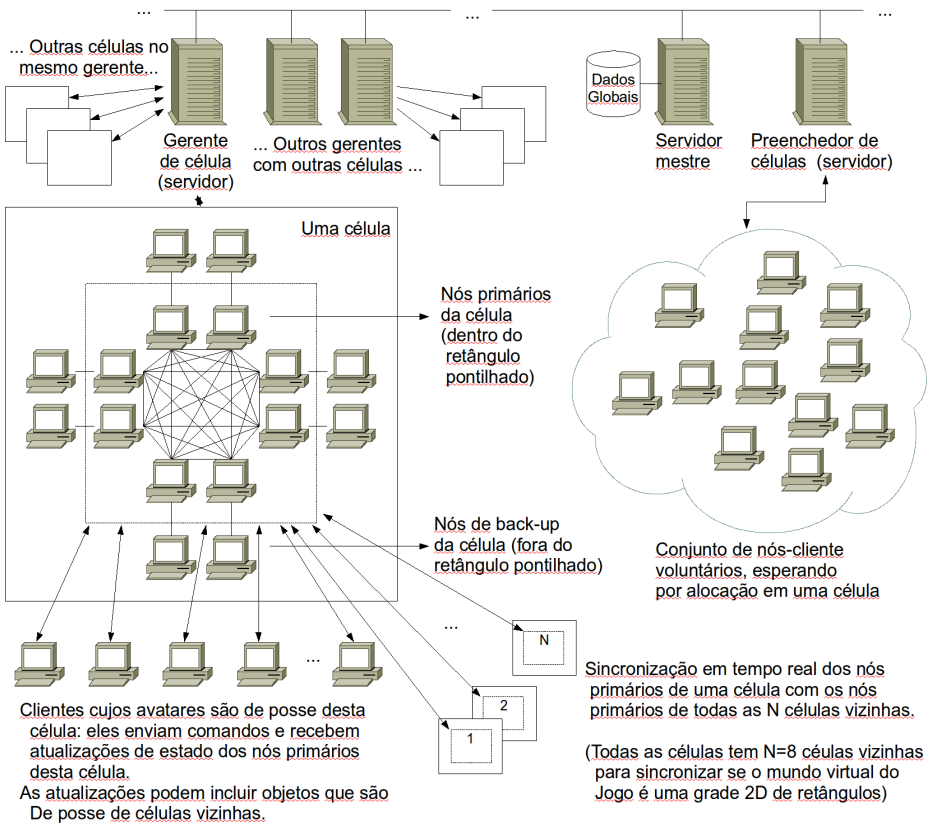

Figura 1: Visão geral da arquitetura FreeMMG 2.

A Figura 1 mostra uma visão geral da arquitetura, com ênfase na relação entre os recursos computacionais envolvidos. O FreeMMG 2 divide o espaço do mundo virtual em células, sendo que células que representam espaços contíguos do terreno virtual possuem uma relação de vizinhança entre si. Cada célula é mantida por um número fixo de nós primários, que são máquinas anônimas escolhidas aleatoriamente a partir de um conjunto de nós voluntários. Cada nó primário de uma célula mantém uma réplica do estado da célula. Como discutido anteriormente, isto torna a realização de *state cheating* difícil, sendo necessário coordenação entre nós escolhidos aleatoriamente a partir de um conjunto de voluntários onde a maioria é composta de nós honestos. Para manter-se o estado da célula replicado e atualizado, torna-se necessário executar um protocolo de sincronização entre os nós primários. Este protocolo está descrito em outro artigo (CECIN et al., 2006) e não será reproduzido aqui devido a limitação de espaço. Também, por motivos de espaço, não detalharemos o funcionamento dos nós de *back-up* de uma célula. Estes são responsáveis por tratar do problema de *churn* (STUTZBACH e REJAIE, 2006) em redes par-a-par, bem como minimizar o impacto de ataques de rede (por exemplo, *denial of service* (MOORE et al., 2006)) contra os nós primários.

Algumas tarefas-chave que possuem um baixo custo computacional e de comunicação são realizadas por máquinas servidoras, como a autenticação de jogadores e a gerência das células. O processo gerente de uma célula é responsável principalmente por garantir que os nós da célula descubram o endereço uns dos outros, e de tratar e recuperar situações de falha. Por exemplo, quando um nó primário atinge um tempo limite enquanto se comunica com outro nó da célula, ou quando dois nós primários divergem quanto a versão atual do estado da célula, o gerente é acionado para resolver a situação de falha. Para recuperar a célula de uma situação de falha, o gerente dispõe apenas de mecanismos simples: substituição de nós voluntários por outros, e remoção permanente de nós voluntários que estão frequentemente envolvidos em situações de falha. Ao invés de tentar detectar a causa exata de uma falha em uma célula, o gerente da célula considera todos os nós da célula envolvidos como igualmente culpados. Parte-se do princípio que a maioria dos nós é honesto e, portanto, apenas nós desonestos ou mal supridos (com má conectividade, etc.) estarão envolvidos em falhas com freqüência

suficiente para ultrapassarem um limiar de exclusão definitiva.

Em um jogo online cliente-servidor, o servidor é responsável por receber um fluxo contínuo de pacotes oriundos dos clientes (jogadores) que contém os seus comandos, e por enviar um fluxo contínuo de pacotes aos clientes que contém atualizações oficiais do estado do jogo, que está sendo continuamente computado pelo servidor, em tempo real. No FreeMMG 2, os nós primários de uma célula são os servidores da mesma. As máquinas dos jogadores propriamente ditos se conectam com todas as máquinas da célula onde atualmente se situa o seu avatar. Estes nós primários ficam então coletivamente responsáveis por receberem os comandos destes clientes, e enviarem atualizações para estes clientes. A solução adotada para minimizar a possibilidade de trapaça no recebimento de comandos de jogadores é a de permitir que jogadores enviem o mesmo comando, assinado digitalmente, para mais de um nó primário, ou para nós primários distintos. Desta forma, ataques do tipo atraso ou descarte intencional de comandos, a serem realizados pelos nós primários, são filtrados. Além disso, a assinatura digital impede que um nó primário fabrique comandos de jogadores. Alternativamente, pode-se evitar o uso de assinaturas digitais enviando-se o comando para vários nós primários e realizando consenso. Pode-se também simplesmente programar os clientes para enviar cada comando para um nó primário escolhido aleatoriamente, a fim de evitar que os (poucos) nós primários desonestos controlem todos os comandos de um jogador e minimizando o impacto desta trapaça. Uma implementação do FreeMMG 2 deverá deixar a escolha do mecanismo específico de disseminação de comandos de jogadores para a aplicação.

No sentido contrário, o envio de atualizações por parte dos nós primários para os jogadores, é feito de uma única forma. Naturalmente, utiliza-se simulação discreta em cada célula: o estado é atualizado de forma discreta. Em cada atualização ou “passo”, os nós primários fazem uma divisão aleatória dos clientes a serem atualizados. Por exemplo, se a célula possui oito (8) nós primários e dezesseis (16) jogadores conectados, cada nó primário será responsável por enviar dois (2) pacotes de atualização, para dois clientes distintos. Em cada passo da simulação da célula, o mapeamento entre nós primários e clientes são refeitos seguindo-se uma seqüência determinística, de forma a minimizar trapaças onde um nó primário

falsifica mensagens de atualização visando prejudicar um cliente específico. Além disso, pode-se exigir que o nó primário assine digitalmente cada atualização enviada. Como assinaturas digitais impedem não-repudiação e o estado do jogo é replicado em outros nós primários, é possível identificar um nó primário discordante inequivocamente comparando-se a sua atualização assinada com outras, também assinadas, enviadas por outros nós primários no mesmo passo.

Outro componente importante da arquitetura é a sincronização entre células vizinhas. Basicamente, tem-se que avatares localizados em células distintas, porém vizinhas, podem potencialmente interagir. Para que essa interação se torne possível, os nós primários de células vizinhas trocam atualizações de estado. Desta forma, os nós primários de uma célula "sabem" não só o estado de objetos localizados na célula em questão, mas também sabem o estado de objetos localizados nas células vizinhas. Assim, ao enviar mensagens de atualização para os clientes, os nós primários da célula podem incluir informações relativas a objetos presentes em células vizinhas. Nós resolvemos o problema desta forma pois, de qualquer maneira, para atualizar-se o estado de uma célula, tem-se que considerar a influência de objetos que não estão na célula. Por exemplo, se um avatar em uma célula remota realiza um disparo, é possível que um avatar na célula local "morra", e uma maneira eficaz de os nós primários locais perceberem essa influência é através de um fluxo de atualizações de estado oriundo da célula remota – o mesmo mecanismo empregado para manter-se clientes atualizados. Além disso, os clientes se tornam mais simples pois, em geral, não precisam sincronizar com mais de uma célula para que seus avatares possam interagir com objetos localizados em células adjacentes.

O FreeMMG 2 é um trabalho em andamento. Nas seções seguintes, outra abordagem de suporte par-a-par para jogos massivos será apresentada, com um foco maior em escalabilidade.

4. Um modelo baseado em servidor distribuído voluntário

Uma outra possível abordagem distribuída para dar suporte a MMOGs é utilizando a arquitetura cliente-servidor, com uma diferença: o servidor é distribuído geograficamente, entre nodos de baixo custo providos por voluntários ou por empresas que visem possibilidade de lucro ao disponibilizar seus recursos computacionais para participar do sistema servidor. Apesar de, geralmente, ser necessário uma infra-estrutura central poderosa e cara (FENG, 2007), um sistema

que seguisse essa proposta permitiria que recursos de empresas menores e usuários domésticos (com o mínimo exigido de capacidade de processamento e largura de banda, o que irá depender do jogo específico) fossem utilizados para formar um sistema para servir um jogo online maciçamente multijogador com qualidade comparável à dos MMOGs mais populares.

Contudo, para que um sistema de servidor distribuído como este funcione, são necessárias duas coisas: otimização do uso de largura de banda, já que se espera a formação de um sistema servidor composto por nodos de baixo custo, e balanceamento dinâmico da carga entre os nodos servidores, levando em conta que seus recursos provavelmente serão heterogêneos devido à natureza voluntária dos participantes. Nas seções seguintes serão apresentadas as abordagens propostas para essas questões.

4.1 A³: um algoritmo de otimização por área de interesse

Para evitar que o custo de manutenção do sistema distribuído servidor como um todo 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. Propõe-se aqui 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 et al., 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.

Área de interesse definida para o algoritmo A³

O princípio por trás da abordagem proposta aqui se baseia 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 – ou seja, tem uma relevância maior –, é desejável que o jogador disponha de seu estado mais recente assim que possível, para poder visualizar quaisquer mudanças

rapidamente.

A área de interesse que propomos, utilizando a idéia de diferentes níveis de relevância, leva em conta:

- 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 evitar que ocorram problemas caso o jogador faça seu avatar girar ao redor do próprio eixo muito rapidamente, melhorando 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 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 2, é ilustrada a área de interesse que acaba de ser definida.

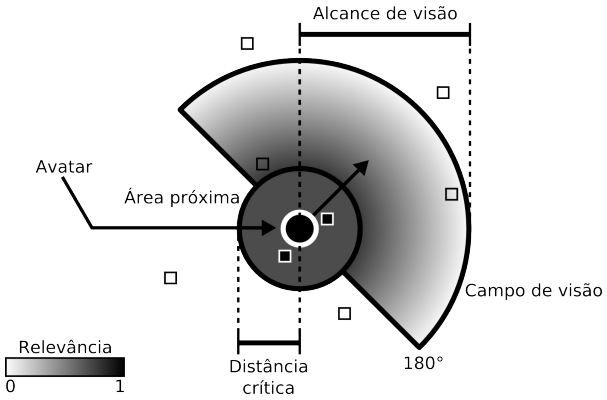

Figura 2: Área de interesse do A³

Simulações e resultados

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 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 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. Os algoritmos comparados foram os baseados em círculo, campo de visão (BOULANGER et al, 2006) e o A³.

Obteve-se, com o A³, uma redução de uso médio de largura de banda de envio de 63,51% e 33,58%, comparado respectivamente com o algoritmo de área de interesse circular e baseado em campo de visão. Reduziu-se também o pico de utilização em 52,03% e 33,10%, comparado com os mesmos algoritmos.

4.2 Balanceamento dinâmico de carga

Aqui, é proposto um esquema de balanceamento de carga que, levando em conta a largura de banda como critério do balanceamento, busca alocar a carga nos servidores proporcionalmente à capacidade de cada um e reduzir tanto quanto possível – e viável, considerando que o jogo não pode esperar indefinidamente pelo algoritmo de balanceamento – o *overhead* da distribuição. O esquema faz primeiro uma seleção local de servidores para participarem, e então o balanceamento em si. Foi proposto um algoritmo, ProGReGA-KF (*proportional greedy region growing algorithm, keeping usage fraction)*, que obteve um desempenho bom, quando comparado com outros algoritmos simulados.

Algumas questões importantes foram levadas em consideração para a definição do esquema de balanceamento de carga dinâmico para MMOGs. Primeiro, sendo um sistema heterogêneo, a carga delegada a cada servidor depende da sua disponibilidade de recursos. Segundo, o gargalo está na comunicação – seja no

quesito uso de largura de banda ou atraso nas mensagens – e não no uso de CPU. Além disso, a carga sobre cada nodo servidor *não* é proporcional à quantidade de jogadores conectados a ele; na verdade, essa carga irá depender da maneira como os jogadores estão interagindo. Se eles estiverem todos interagindo uns com os outros, mais mensagens deverão ser trocadas e, já que é o servidor que faz a ligação entre cada par de jogadores, o uso de sua largura de banda deverá ser maior. Por último, pares de jogadores que estão interagindo um com o outro devem, idealmente, estar conectados ao mesmo servidor. Caso contrário, os servidores aos quais eles estão conectados deverão trocar mensagens entre si, além das mensagens que trocariam normalmente com cada jogador, causando um desperdício de largura de banda, como também aumentará o número de saltos necessários para que as ações de um jogador possam ser visualizadas pelo outro.

Divisão do ambiente virtual em células

Para ser feita a distribuição de carga entre os nodos servidores, é necessário determinar qual será o grão de distribuição. Distribuir jogadores individualmente teria uma granularidade muito fina, aumentando a complexidade do algoritmo de balanceamento. Ao invés disso, será dividido o ambiente virtual em uma grade de células fixas (DE VLEESCHAUWER et al., 2005), que poderão ser agrupadas em regiões. Cada região, então, é designada a um dos nodos servidores, como ilustrado na Figura 3.

Figura 3: Ambiente virtual dividido em células, que são agrupadas em regiões

Quando um servidor estiver sobrecarregado, ele pode transferir algumas de suas células para outro, que esteja com mais recursos ociosos. A carga de uma célula é definida como a largura de banda de *upload* usada pelo servidor para manter atualizados os jogadores cujos avatares estão naquela célula. Obviamente, isso vai depender do quão relevante é cada jogador em relação aos outros da mesma célula.

Mapeamento para grafo e balanceamento de carga

Uma vez definido a unidade a ser distribuída, pode ser utilizado um algoritmo clássico da área de sistemas distribuídos para alocação de tarefas. Isso pode ser feito da seguinte forma: cada célula (unidade a ser distribuída) é representada por um vértice em um grafo, onde as arestas ligam pares de vértices que representam células adjacentes. O peso de cada vértice é corresponde à carga daquela célula. Cada aresta também tem um peso, que representa a interação entre jogadores cujos avatares estão cada um em uma das células cujos vértices são ligados por aquela aresta. Na Figura 4, é ilustrado esse mapeamento.

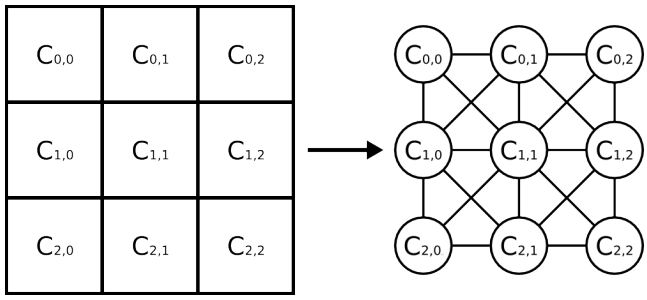

Figura 4: Representação em grafo

Considerando-se que já existe uma distribuição inicial das células entre os servidores, formando regiões, o algoritmo de balanceamento de carga aqui proposto é disparado quando algum dos servidores ultrapassa um determinado limite de utilização de seus recursos. Quando isso acontece, ele busca servidores próximos a ele e os adiciona um por um, até que a capacidade total dos servidores selecionados seja maior que a carga total atribuída àquele conjunto de servidores. Uma vez formado esse conjunto que necessariamente possui recursos suficientes para lidar com toda a carga atribuída a ele, é iniciado um algoritmo de redistribuição das células pertencentes a todos esses servidores. O algoritmo consiste em:

1. O servidor S, sobrecarregado, libera sua célula mais leve;
2. O passo 1 se repete até que a carga sobre S seja menor ou igual à sua capacidade;
3. Cada um dos outros servidores selecionados na fase inicial, começando daquele com mais recursos livres, adiciona gradativamente as células que S liberou, enquanto possuir recursos livres.

O algoritmo ProGReGA-KF define que, a cada passo, o servidor T, que receberá as

células, busca aquelas que estejam próximas e com grande interação com as células que ele já tem – no grafo, seria um vértice que tivesse uma aresta com os vértices próprios; se houver mais de um, escolhe-se aquele ligado pela aresta mais pesada. Se nenhuma célula que foi liberada for vizinha de alguma célula de T, este escolhe a célula livre mais pesada que houver.

A escolha da aresta mais pesada para determinar qual célula será adicionada tem por objetivo explorar a localidade da interação dos jogadores, já que pares destes devem idealmente estar conectados ao mesmo nodo servidor quando estiverem interagindo, reduzindo o *overhead* sobre o sistema.

Simulações e resultados

Foram realizadas simulações utilizando um ambiente virtual dividido em 225 células, com 8 servidores, cada um com capacidade diferente, onde avatares se moviam de maneira não uniforme, gerando desbalanceamento na carga do jogo. Os algoritmos simulados foram: ProGReGA (no passo 1, todos os servidores liberam todas suas células), ProGReGA-KH (no passo 1, cada servidor mantém apenas a sua célula mais pesada), **ProGReGA-KF**, BFBCT (*best-fit based cell transference*) e um algoritmo do estado-da-arte (AHMED e SHIRMOHAMMADI, 2008).

Considerando uma distribuição “justa” quando cada servidor lida com uma carga perfeitamente proporcional aos seus recursos, os resultados foram: ProGReGA, cujo *overhead* de comunicação entre servidores aumentou em 13% a carga do jogo, em média, mas causou o segundo maior número de migrações de jogadores entre servidores, além da distribuição menos justa; ProGReGA-KH, que apresentou uma carga extra de 28% devido a *overhead*, porém com o terceiro menor número de migrações, além de ter sido uma das mais justas; ProGReGA-KF, cujo aumento de carga foi de 22%, com o segundo menor número de migrações, além de ser um dos mais justos; BFBCT, que aumentou a carga sobre o sistema em 34%, com o maior número de migrações de todos e não tão justo quanto os outros algoritmos simulados. Por último, o algoritmo de Ahmed aumentou a carga do sistema em 29%, em média, com o menor número de migrações, porém o nível de justiça variando muito com o tempo.

Considerações finais

Aqui, foram apresentados dois modelos de distribuição do suporte de rede a MMOGs. Um deles, o FreeMMG 2, atingiu o seu objetivo de reduzir o custo do servidor, delegando a simulação aos *peers* dos jogadores. Para mitigar a possibilidade de haver trapaça, particularmente a de fabricação de estado, mais peers calculam e enviam o estado resultante das ações dos jogadores em uma dada região.

O outro modelo proposto, baseado em um sistema distribuído de nodos servidores voluntários, teve como objetivo a definição de técnicas que lidassem com a escassez e não-uniformidade dos recursos desse sistema. Para otimizar o uso de largura de banda dos servidores, foi apresentado um algoritmo de gerenciamento de interesse, o A³, 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. Obtiveram-se resultados significativos com o uso desta técnica. Por fim, para distribuir a carga entre os nodos servidores, reduzindo o *overhead* da distribuição e atribuindo cargas proporcionais à capacidade dos servidores, foi proposto também um esquema de balanceamento de carga dinâmico. 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. Comparando-o com outros esquemas, verificou-se que seu desempenho é razoavelmente melhor.

Agradecimentos

Gostaríamos de agradecer ao Conselho Nacional de Pesquisa (CNPq) e à Coordenação de Aperfeiçoamento do Ensino Superior (CAPES) pelo suporte a este trabalho.

Referências

AHMED, D.; SHIRMOHAMMADI, S. A Microcell Oriented Load Balancing Model for Collaborative Virtual Environments. **VECIMS 2008**. Piscataway, NJ: IEEE, 2008. p.86–91.

BLIZZARD. **World of Warcraft**. 2004. Disponível em:

<http://www.worldofwarcraft.com/>. Acesso em: 27/5/2009.

BOULANGER, J.; KIENZLE, J.; VERBRUGGE, C. Comparing interest management algorithms for massively multiplayer games. **NETGAMES 2006**. New York: ACM, 2006. p.6.

CECIN, F. R.; GEYER, C. F. R.; RABELLO, S.; BARBOSA, J. L. V. A peer-to-peer simulation technique for instanced massively multiplayer games. **DS-RT 2006**. Washington, DC: IEEE, 2006. p.43–50.

CHEN, A.; MUNTZ, R. Peer clustering: a hybrid approach to distributed virtual environments. **NETGAMES 2006**. New York: ACM, 2006. p.11.

DE VLEESCHAUWER, B. et al. Dynamic microcell assignment for massively multiplayer online gaming. **NETGAMES 2005**. New York: ACM, 2005. p.1–7.

FENG, W. **What's Next for Networked Games?** 2007. Disponível em: <http://www.thefengs.com/wuchang/work/>. Acesso em: 27/5/2009. (NetGames 2007 keynote talk, nov. 2007).

GAUTHIERDICKEY, C.; ZAPPALA, D.; LO, V.; MARR, J. Low latency and cheat-proof event ordering for peer-to-peer games. **NOSSDAV 2004**. New York: ACM, 2004, p.134–139.

KABUS, P.; TERPSTRA, W.; CILIA, M.; BUCHMANN, A. Addressing cheating in distributed MMOGs. **SIGCOMM 2005**. New York: ACM, 2005. p.1–6.

LEHTINIEMI, T.; LEHDONVIRTA, V. How big is the RMT market anyway. **Virtual Economy Research Network**, [S.l.: S.n.], v.3, 2007.

MCCANNE, S.; FLOYD, S. et al. **Network simulator ns-2**. 2006. Disponível em: <http://www.isi.edu/nsnam/ns>. Acesso em: 27/5/2009.

MCGRAW, C.; HOGLUND, G. Online games and security. **IEEE Security & Privacy**,

[S.l.], v.5, n.5, p.76–79, 2007.

MOORE, D.; SHANNON, C.; BROWN, D.; VOELKER, G.; SAVAGE, S. Inferring Internet denial-of-service activity. **ACM Transactions on Computer Systems**. [S.l.], v.24, n.2, p.115–139, 2006.

ROBERTS, D.; WOLFF, R. Controlling consistency within collaborative virtual environments. **DS-RT 2004**. Washington, DC: IEEE 2004. p.46–52.

SONY ONLINE ENTERTAINMENT. **PlanetSide**. 2008. Disponível em: <http://planetside.station.sony.com>. Acesso em: 27/5/2009.

STUTZBACH, D.; REJAIE, R. Understanding churn in peer-to-peer networks. **SIGCOMM 2006**. New York: ACM, 2006. p.189–202.

THOMPSON, Z. **The in-game economics of Ultima Online**. 2000. Disponível em: <http://www.mine-control.com/zack/uoecon/uoecon.html>. Acesso em: 27/5/2009.

WEBB, S.; SOH, S. Cheating in networked computer games: a review. **DIMEA 2007**. New York: ACM, 2007. p.105–112.

YAN, J.; RANDELL, B. A systematic classification of cheating in online games. **NETGAMES 2005**. New York: ACM, 2005. p.1–9.