Modelo de Comunicação Híbrido para Jogos Massivos Multi-jogador

Solon A Rabello Fábio Cecin* Jorge L. V. Barbosa Cláudio F. R. Geyer*

Universidade do Vale do Rio dos Sinos, Mobilab/Pipca, Brasil
*Universidade Federal do Rio Grande do Sul, Brasil

Abstract

We propose a peer-to-peer event ordering and simulation technique aimed at networked real-time action games. Partially based on replicated simulators, its goal is to support decentralized playout in small-scale game sessions on instanced action spaces while being resistant to collusion cheating. The action spaces are linked to persistent-state social spaces of larger scale which are supported by centralized simulation. Together, these two kinds of spaces offer support for massively multiplayer on-line games (MMOGs) that offer a mix of socialization on large-scale persistent environments and fast interaction on small-scale temporary ones. Although player nodes on action spaces are required to run a conservative and an optimistic simulator simultaneously, we show that 2.2 simultaneous simulation steps are executed on average and that 11.95 simultaneous steps occur as the average peak situation for 20-player sessions with 150ms to 300ms network delays between nodes, 5% probability of any late events introducing errors, and rollback and re-execution operations having their execution spread through 100ms of real time or longer.

Keywords: Communication, peer-to-peer, massive multiplayer game

Resumo

Este artigo apresenta uma técnica de comunicação de dados para jogos de ação em tempo real, usando de forma híbrida os modelos cliente-servidor e ponto-a-ponto. Baseado parcialmente na réplica de simuladores, tem como objetivo suportar a computação descentralizada de sessões de jogo de pequena escala, sendo essas instâncias (espaços de ação) resistentes às trapaças de conluio (*Collusion cheat*). Esses espaços de ação são unidos pelo espaço social, que é o estado persistente do servidor suportando alta escalabilidade. Juntos, esses dois tipos de espaços dão suporte aos jogos massivos multi-jogador (MMOGs) através da união de espaços sociais persistentes e centralizados com espaços de ação de pequena escala, temporários e descentralizados. Nesse modelo os nodos dos jogadores são os responsáveis pela execução simultânea da simulação conservadora e da otimista. Será mostrado que em média 2,2 simulações simultâneas são executadas por passo e que a média dos máximos das simulações por passo é 11,95 para sessões de teste de 20 jogadores com *lag* de rede entre 150ms e 300ms, possuindo 5% de probabilidade do atraso de eventos introduzir erros, e tendo a computação de *rollbacks* e de execuções corretivas diluída em 100ms, ou mais, de tempo real.

Palavras-chave: comunicação, ponto-a-ponto, jogos massivos multi-jogador

Authors' contact:
{solonr,jbarbosa}@unisinos.br
*{fcecin,geyer}@inf.ufrgs.br

1. Introdução

Os jogos multi-jogador massivos *online*, ou *massive multiplayer online games* (MMOGs) são aplicações em tempo real de simulação distribuída com popularidade crescente. Este tipo do jogo suporta grandes quantidades de jogadores que interagem em tempo real em um mundo virtual de estados persistentes. Warcraft [BLIZZARD 2006] e Guild Wars [ARENANET 2006] são exemplos atuais de MMOGs. Como uma referência desse crescimento de popularidade destaca-se o World of Warcraft, suportando atualmente centenas de milhares de jogadores simultaneamente [BLIZZARD 2005].

A maioria, se não todos, os MMOGs comerciais usam a estrutura cliente-servidor, onde toda a simulação do mundo virtual é realizada de forma distribuída entre nodos servidores. Esses nodos são servidores dedicados que ficam em lugares seguros, com uma grande capacidade de tráfego à sua disposição, assegurando uma baixa latência na comunicação mesmo com centenas de conexões simultâneas. A manutenção desse tipo de estrutura impõe um custo significativo ao projeto, que poderia ser reduzido se fossem utilizadas outras soluções para a distribuição da carga computacional

A distribuição da simulação dos MMOGs nas máquinas dos jogadores é atualmente um tópico de pesquisa, tendo como objetivo eliminar ou diminuir o custo com os servidores dedicados. Nesse artigo, será apresentado um modelo de distribuição simples, apropriado para MMOG. Nesse modelo, o mundo do jogo é representado não como um espaço contíguo, mas como uma coleção de espaços separados. Como inspiração para tal representação foi utilizado o *design* do jogo Guild Wars [ARENANET 2006].

O modelo proposto de distribuição (seção dois) delega algumas instâncias da simulação (espaços do jogo) diretamente às máquinas dos jogadores. Assim como apontado pelos trabalhos [BAUGHMAN E LEVINE 2001; KABUS 2005; YAN E RANDELL 2005] isso pode conduzir a trapaças no jogo. Para lidar com isso, o modelo propõe uma nova técnica de simulação ponto-a-ponto (seção três) inspirada no *Trailing State Synchronization* (TSS) [CRONIN ET AL 2004]. Nesse modelo, réplicas da simulação de cada espaço virtual são mantidas em máquinas dos jogadores, dependendo da rede e do poder de computação delas. Na seção quatro, são apresentados resultados relacionados ao uso de CPU das máquinas dos clientes, obtidos a partir das simulações realizadas. Em seqüência serão discutidos os trabalhos relacionados, encerrando na seção seis com as conclusões e os trabalhos futuros.

2. Modelo das Instâncias do Jogo

Muitos jogos massivos multi-jogador, como o PlanetSide [SONY. PLANETSIDE, 2006] e World of Warcraft, oferecem ao jogador a ilusão de um ou diversos grandes espaços virtuais contíguos, que podem ser explorados. Mas esses grandes espaços são na realidade uma coleção de pequenos espaços unidos de forma transparente ao jogador. Essa divisão só é percebida pelo lado do servidor onde cada nodo servidor é responsável por um número limitado de áreas do jogo [DUONG AND ZHOU 2003, VLEESCHAUWER ET AL, 2005]. Alguns desses "grandes" espaços virtuais podem ser unidos através de "portais", que permitam que o jogador se transporte para espaços de jogo diferentes itens.

O jogo Guild Wars [ARENANET 2006] introduz um novo aspecto no design de jogos MMOs, que no escopo desse artigo se refere a um modelo de instâncias. Guild Wars provê dois tipos de espaços virtuais. O primeiro tipo é um espaço onde alguns milhares de usuários interagem socialmente: vendendo, trocando itens, conversando ou organizando sessões de jogo, que ocorrem no segundo tipo de espaço. O segundo tipo de espaço virtual é pequeno, conectando algumas dezenas de jogadores, mas que requer uma alta consistência, é nesse espaço que acontecerá a ação do jogo. Nesse artigo será utilizado o termo "espaço social" referindo-se ao primeiro espaço e "espaço de ação" referindo-se ao segundo.

A interação entre os espaços acontece da seguinte maneira. A sessão de jogo é iniciada quando um grupo de jogadores se reúne no espaço social e decide ir a um espaço de ação. Na sessão de ação eles lutam, interagem e ganham bens virtuais, como itens, status entre outros. Quando a sessão de jogo no espaço de ação termina, os jogadores retornam para o ambiente social, provavelmente carregando seus novos bens virtuais. Os espaços sociais são como os espaços persistidos do World of Warcraft e do PlanetSide, enquanto que os espaços da ação são criados dinamicamente para suportar somente uma sessão provisória do jogo com um pequeno número de jogadores.

Assim como outros MMOGs, Guild Wars foi implementado seguindo o modelo cliente-servidor. Nessa proposta é apresentada uma fusão dos modelos cliente-servidor e ponto-a-ponto como solução da comunicação em MMOGs. Os espaços sociais (conversa e trocas) continuam no modelo cliente-servidor, mas têm a exclusiva função de "espaço social". Com essa modificação a necessidade de consistência é baixa, por isso os servidores podem diminuir sua taxa de comunicação com os clientes sem que isso interfira nos objetivos do espaço social. Por outro lado, os espaços de ação adotam um modelo ponto-a-ponto, onde um grupo de usuários fica conectado diretamente, e o processamento é feito em suas máquinas. Em outras palavras, o servidor é liberado da necessidade de maior taxa de comunicação e poder de processamento, os tornado responsabilidade do cliente. Sendo assim, a estrutura e os custos do servidor são reduzidos e a estabilidade de cada nodo servidor aumentaria, pois em grande parte do tempo a maioria dos jogadores se encontraria no espaço de ação, ou seja, instâncias ponto-a-ponto mantidas pelos próprios usuários. A figura 1 ilustra esse modelo.

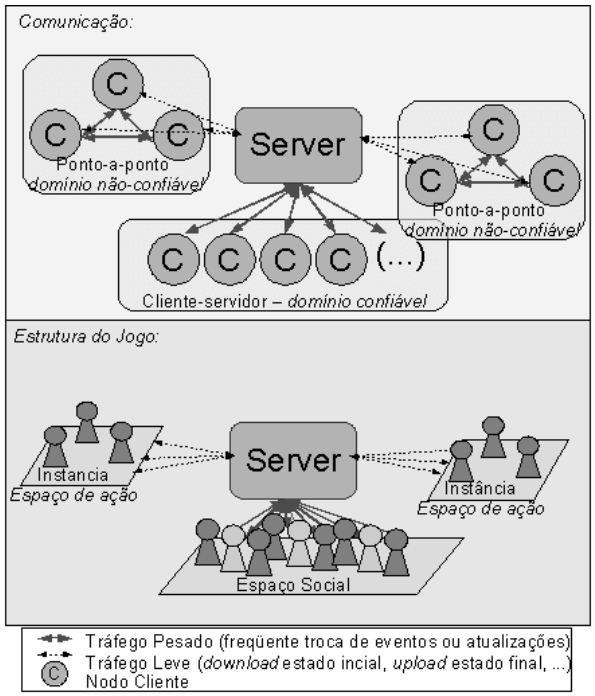

Figura 1. Ilustra o cenário proposto.

Essa proposta, por outro lado, introduz problemas relacionados à trapaça no jogo. Dependendo do modelo de sincronização adotado pelos espaços de ação, alguns jogadores podem manipular o jogo para obter vantagem para si próprios, retornando para o espaço social com somas arbitrárias de bens virtuais, como dinheiro, itens e experiência. Esse problema pode ser inibido pela réplica das entradas sincronizadas de

dados dos jogadores nos simuladores dos nodos [CECIN, ET AL 2004]. Se os eventos forem executados na mesma ordem, usando os mesmos *timestamps* nos simuladores de todos os nodos, então um único nodo honesto é requerido para se detectar inconsistências e com isso negar os bens teoricamente adquiridos na sessão da instância do jogo [CECIN, ET AL 2004]. Entretanto ainda existe a possibilidade de trapaças de conluio (*Collusion cheat*) [CECIN, ET AL 2004], onde todos os nodos manipulam ilegalmente as instâncias dos jogos simultaneamente. Esse tipo de trapaça pode ser eliminado pela utilização de nodos observadores [CECIN, ET AL 2004], que são réplicas adicionais alocadas randomicamente para uma sessão de instância do jogo, reduzindo assim a probabilidade de todos os nodos executarem mudanças ilegais sincronizadas nos simuladores de cada nodo.

3. Técnica de Simulação Ponto-a-ponto

Nessa seção será proposta uma técnica de simulação para um jogo de ação de baixa escalabilidade com grandes volumes de trocas de dados. A proposta defende o uso de duas cópias do estado do jogo em cada máquina: sendo a primeira a conservadora e a segunda a otimista.

A cópia conservadora será atualizada depois de o nodo receber todas as informações, referentes ao passo, de todos os jogadores. Por outro lado, a otimista é atualizada seguindo o modelo do "mais rápido que puder", onde as ações dos jogadores são executadas assim que são recebidas e as ações dos jogadores que não enviaram suas ações são extrapoladas com base na simulação conservadora. A cópia conservadora tem por objetivo detectar a utilização de trapaças de conluio pela replicação dos estados da simulação, sendo também a base para os *rollbacks* e execuções corretivas de passos quando eventos chegam atrasados.

Na simulação foi utilizado um modelo de rede ponto-a-ponto, com a garantia de existência de caminho entre todos os nodos. A simulação assume que existe um canal de comunicação entre todos os jogadores que possibilita a troca de mensagens TCP, RTP ou UDP.

A técnica proposta requer que todos os nodos de um grupo compartilhem um relógio assíncrono. Esse relógio pode ser utilizado através de protocolos como o NTP [MILLS, 1992]. Quanto menos sincronizados os relógios estiverem, mais as simulações poderão divergir temporalmente. Entretanto, o algoritmo sempre garante a convergência das simulações, mesmo que seus relógios não estejam sincronizados, ou seja, mesmo que os pacotes cheguem com vários segundos de atraso as simulações em todas as máquinas manterão a sua consistência. A única evidência desse atraso será o atraso da ação do personagem do jogador com *lag*, a qual pode ser eliminada ou pelo envio das ações atrasadas ou por "voto de desconexão" como feito no FreeMMG [CECIN, ET AL 2004]. Isso não acontece com o protocolo *Bucket Synchronizarion* [GAUTIER AND DIOT, 1997], onde um atraso grande de pacotes pode resultar em falta permanente de sincronia entre os nodos [CRONIN, ET AL, 2004; GAUTIER, AND DIOT , 1997].

No modelo proposto nesse artigo assume-se que todos os nodos podem sincronizar seus relógios com o servidor por tentativa e erro, dispensando uma terceira entidade para sincronização do tempo do servidor. A técnica também requer que todos os nodos comecem com uma cópia sincronizada do estado do mundo virtual e um acordo sobre o tempo inicial da simulação.

3.1 Detalhes do Modelo

Como mencionado anteriormente, esse modelo trabalha com duas cópias do estado do jogo na memória em cada máquina: a otimista e a conservadora. A cada *tick* a simulação compara essas cópias, e quando encontra divergências significativas entre a cópia otimista e a cópia conservadora, realiza um *rollback* de estados com base na cópia conservadora e executa novamente alguns passos da simulação. Esse *rollback* tem o objetivo de alinhar as cópias otimistas e conservadoras, corrigindo assim a divergência introduzida pela extrapolação executada na cópia otimista. O custo computacional para execução do *rollback* e de alguns passos de simulação depende do atraso da rede entre o nodo local e o nodo remoto.

Um evento gerado localmente, na sua criação, recebe um *timestamp*, um valor do seu relógio local. Partindo da premissa que todos os relógios estão, teoricamente, sincronizados, isso significa que o evento é executado simultaneamente em todos os nodos, de acordo com a técnica de escalonamento empregada, como o *Bucket Synchronization* [GAUTIER AND DIOT, 1997; GAUTIER AND DIOT, 1998] ou *stop-and-wait* [CECIN, 2004]. Entretanto, isso é o mesmo que afirmar que todos os eventos gerados localmente estão sempre "atrasados" para o nodo remoto, que compara o *timestamp* do pacote recebido com o seu valor de relógio local. Mesmo com esse problema essa escolha foi mantida, pois em jogos de ação executar os movimentos "o mais rápido possível" é preferencial a executá-los localmente com um atraso para manter todos os nodos com a mesma velocidade de visualização.

3.2 Simulador Conservador

O simulador conservador em um nodo trabalha em turnos, que são divisões fixas de tempo *Sc* (foi utilizado *Sc* de 50ms no protótipo). A cópia conservadora só é atualizada para o tempo *Tc* quando se garante que todos os eventos de todos os jogadores com o *timestamp* no intervalo [*Tc, Tc + Sc*] foram recebidos. Desde que a rede forneça entrega com garantia, é garantido que o nodo em questão recebe ao

menos um evento com o *timestamp* maior que *Tc* + *Sc* de todos os outros nodos. Para evitar "travadas" desnecessárias no simulador, todos os nodos devem enviar ao menos um evento a cada intervalo de tempo *Sc*. Se o nodo não gerou nenhum evento significativo naquele intervalo, ele envia um "*ping*" vazio.

Essa requisição de eventos no simulador conservador deve ser acoplada com o restante do código determinístico do simulador (como movimento de objetos e colisão), com isso serão produzidos os mesmos resultados para todos os jogadores em todos os nodos, se os módulos de física dos simuladores também forem determinísticos.

3.3 Simulador Otimista

O simulador otimista é aquele responsável pela imagem que o usuário vê na tela. Ele a atualiza de acordo com a última mensagem recebida. Esse simulador avança seu estado recebendo os eventos de todos usuários ou não.

O simulador otimista avança seu tempo *To* usando como referência o tamanho do passo *So*. No modelo proposto foi usado um *So* de 50ms.

Quando avança o tempo de *To* para *To* + *So*, o valor final *To* + *So* deve ser, idealmente, igual ao instante atual *To* do relógio sincronizado da simulação. Durante o passo, todos os eventos conhecidos no intervalo [*To, To+So*] são usados na computação para o próximo estado.

Nesse protótipo, quando se deve avançar um passo do simulador otimista e um jogador ainda não enviou seus eventos, sua posição é extrapolada para onde ele deveria estar no momento atual (*dead reckoning*). Isso é feito com base no simulador conservador. Entretanto se o jogador passar mais de três ciclos do simulador otimista sem enviar nenhum evento, seu personagem ficará imóvel, evitando assim diferenças abruptas de posição do seu personagem quando seus eventos atrasados chegarem e for executado *rollback* e execução corretiva de passos.

Essas extrapolações ajudam a diminuir a inconsistência entre o que o jogador está vendo e o que ele deveria estar vendo, se soubesse os eventos de todos os usuários. Mesmo assim, um projétil do seu personagem pode acertar um corpo de outro personagem na sua tela (simulação otimista), mas quando receber os eventos desse jogador, atualizando a simulação conservadora, o projétil não passará pela posição onde o personagem inimigo se encontrará. Quanto maior o *lag* na rede mais problemas de colisão poderão acontecer. A figura 2 ilustra um caso prático desse problema no protótipo, onde o cliente A fica 230ms sem receber informações do cliente B. Isso força que a simulação de A extrapole as ações do cliente B. Essa extrapolação causa inconsistências que serão corrigidas, através do *rollback* das ações, assim que o pacote de dados do cliente B chegar em A.

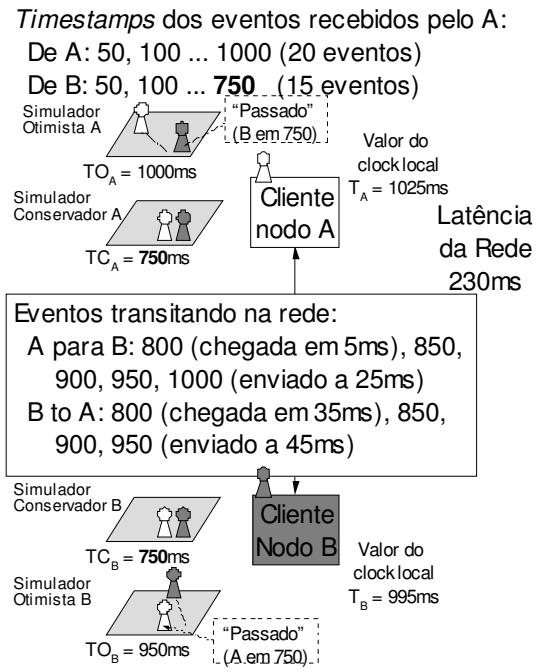

Figura 2. Captura de um possível estado de execução no espaço de ação com dois jogadores.

Alguns jogos *on-line* de tiro em primeira-pessoa, como o Half-Life, lidam com esse problema aplicando a compensação de *lag* [BERNIER, 2001]. Nesse caso, eles verificam as interações (por exemplo, colisão de objetos) entre os objetos locais e os objetos remotos usando a visão obsoleta da simulação local. Sendo Half-Life um jogo cliente-servidor, os objetos remotos em questão são enviados pelo servidor, que por sua vez é atualizado pelo envio dos comandos dos clientes. Nesse modelo um jogador que acabou de cruzar uma esquina pode ser atingido por um jogador na rua ortogonal, caso esse estivesse com atraso de rede alto. Isso acontece porque devido à "visão no passado" do jogador da rua ortogonal, o personagem que está atravessando a esquina ainda não o fez, sendo assim possível atingi-lo.

Visualmente, foi concluído que os resultados da atualização de *dead reckoning* foram menos aceitos do que mostrar o estado passado do jogador. Não foi testada no protótipo a compensação de *lag*, mas isso já foi feito por Bernier no Half-Life [BERNIER, 2001].

3.4 Conservador: Resolvendo Interações

Quando um jogador vê um objeto com atraso (através do simulador otimista que é responsável pelo desenhado na tela), é necessário aplicar a compensação

de *lag* no simulador conservador, ao invés de simplesmente deixar os objetos colidirem no momento *Tc*, ou senão os resultados apresentados pelo simulador otimista seriam de pouca utilidade.

O modelo trabalha da seguinte maneira: os nodos dos jogadores enviam a cada intervalo de tempo *Sc*, para todos outros nodos participantes, um vetor de inteiros informando a última posição conhecida de cada jogador. O simulador conservador, ao passar do *Tc* para *Tc* + *Sc*, pode usar essa informação para, por exemplo, checar a colisão entre um projétil do jogador *P* no momento *Tc* e o corpo do personagem do oponente *O* no momento *Tc* - *A*, onde *A* é o atraso enviado por *P* relativo a vinda na mensagem pela rede de *O*. Em um jogo isso pode ser percebido quando um jogador é atingido depois que ele se escondeu atrás de uma caixa. Dessa forma, o tiro que foi executado no momento *Tc* deveria atingir o corpo do oponente no momento *Tc* - *A*, ou seja, quando ele ainda não estava escondido.

3.5 *Rollback* e Execução Corretiva

Uma parte importante do simulador otimista é o modo como esse trata os eventos atrasados, ou seja, quando um nodo com o tempo do simulador otimista *To* recebe um evento com o *timestamp Te*, onde $Te \leq To$. Cada atraso pode introduzir erro, ou seja, diferença entre o que o simulador otimista mostra para o usuário e o que o simulador conservador aprova como feito pelos usuários. Entretanto, mostrar os objetos dos jogadores remotos "no passado" quando recebidos com atraso, só introduz erro se esses interagem com objetos controlados por outro jogador. Ou seja, eventos atrasados de tiro ou movimento só introduzirão erro se o jogador remoto estiver interagindo com eles, que em um jogo de tiro, por exemplo, acontece em uma pequena parcela de tempo se comparada com o tempo que se passa "fora de combate".

Depois de um passo do simulador otimista, o estado otimista se torna ou aceitável ou inaceitável, diferente do estado conservador que é sempre aceito. Se inaceitável, um *rollback* corretivo seguido de execução corretiva é necessário. Nesse ponto, faltam dois módulos no protótipo. O primeiro seria uma solução genérica para detectar quando dois estados do simulador diferem inaceitavelmente. O segundo, seria uma solução genérica para determinar quando dois eventos podem ser executados fora de ordem, sem introduzir erro. Atualmente essas duas tarefas são deixadas para a aplicação. Como uma solução inicial para jogos de baixa escalabilidade do tipo ação de tiro, é proposto nesse artigo o uso de esferas de influência circundando os objetos para detectar potenciais colisões entre objetos em movimento [BAUGHMAN AND LEVINE, 2001; OHLENBURG, 2004], e caso exista possíveis interceptações a ordem dos eventos deve ser respeitada, do contrário elas podem ser executadas em qualquer ordem, normalmente a ordem de chegada. Nessa solução inicial todos os movimentos recebidos fora de ordem sempre são corrigidos para evitar falta de sincronia.

É importante ressaltar que o acionamento do *rollback* e da operação de execução corretiva não necessitam ser executados todos de uma vez, bloqueando outras funcionalidades do jogo, como desenho ou comunicação de rede. O custo de CPU para executar um *rollback* e execuções corretivas de turnos pode ser muito alto, sendo necessário alguns segundos de processamento no mundo real. Por esse motivo a execução corretiva de muitos passos é dividida em etapas, suavizando assim o processamento na máquina do jogador, reduzindo o impacto visual e a diminuição da responsividade do controle.

4. Protótipo

A validação da técnica proposta foi baseada em um protótipo de jogo que tenta simular os aspectos de movimentação e tiro do jogo multi-jogador de código aberto Outgun [RITARI, 2006]. Outgun é um jogo de tiro comparável ao jogo Quake que foi usado como protótipo do TSS [CRONIN ET AL 2004]. Porém, Outgun e o protótipo apresentado nesse artigo simulam um ambiente 2D simples onde os jogadores movem *sprites* 2D e disparam pequenos círculos em direção aos seus oponentes em oito possíveis direções. O protótipo atual não implementa colisões entre os objetos, apenas a movimentação do personagem e o disparo dos projéteis. Portanto, a parte visual e interativa do protótipo foi suficiente para que se pudesse avaliar visualmente o desempenho da técnica de simulação em relação à movimentação dos personagens, que é rápida e brusca em um jogo de tiro tradicional. Porém, não se pode inspecionar visualmente o impacto da troca de tiros e das colisões entre tiros e personagens. Isto seria útil, por exemplo, para verificar se as correções temporais para tratamento de colisões no simulador conservador, descritas nas seções três e quatro, seriam mesmo aceitáveis, ou se produziriam algum efeito colateral indesejável.

O protótipo foi implementado em C# sobre a plataforma .NET 2.0. Todos os experimentos foram executados em duas máquinas, uma com processador Pentium-M de 1.1GHz e em outra com processador Athlon 2200+, ambas com sistema operacional Windows XP SP2 e 512MB de memória RAM, sobre uma rede local de 100MBits. A comunicação entre os processos foi feita através de *broadcast* UDP sobre a rede local. O atraso na comunicação dos casos de teste foi inserido artificialmente: cada processo esperou um tempo escolhido aleatoriamente dentro do intervalo especificado no respectivo caso de teste antes de enviar cada mensagem. Os relógios dos processos foram sincronizados através do envio de um pacote de relógio a cada 50ms por parte de um processo mestre.

Como o protótipo é um jogo incompleto, insuficiente para causar uma carga significativa nas

CPUs dos atuais PCs, foi contado o número de eventos relacionados ao uso de CPU. Durante os testes foram medidas quantidades de eventos recebidos, tempos de simulação e quantidades de *rollbacks* ocorridos. Para os testes, o atraso artificial de comunicação foi variado em intervalos, com o objetivo de verificar como as medidas relacionadas ao uso de CPU eram influenciadas pelo atraso de rede. Durante os testes ainda foi variado o modo como os *rollbacks* e as execuções corretivas eram computados: imediatamente, ou divididos ao longo do tempo.

Tabela 1: Cenário 1. *Rollback* e execução corretiva imediata

Latência da Rede (ms)	MER*	Méd. N R**	Méd. N R**	MSS***	MMSS****
20 x [0, 50] ms	115968	25,24	297,58	2,02	12,48
20 x [50, 100] ms	115400	43,92	297,56	2,07	18,32
20 x [150, 300] ms	114588	88,69	298,65	2,21	16,48
20 x [300, 600] ms	115323	172,11	306,20	2,50	24,24
19 x [0, 50] ms, 1 x [300, 600] ms	115859	151,32	303,40	2,43	19,73

*Média dos Eventos Recebidos
**Média do numero de rollbacks executados
***Média de simulações simultâneas
****Média do máximo de simulações simultâneas

Tabela 2: Cenário 2. *Rollback* e execução corretiva diluída

Latência da Rede (ms)	MER*	Méd. N R**	Méd. N R**	MSS***	MMSS****
20 x [0, 50] ms	116100	25,84	299,08	2,02	9,95
20 x [50, 100] ms	115647	52,87	275,51	2,10	13,70
20 x [150, 300] ms	114536	86,44	217,06	2,20	11,95
20 x [300, 600] ms	115344	199,96	252,00	2,59	12,65
19 x [0, 50] ms, 1 x [300, 600] ms	115826	181,76	261,87	2,53	9,51

*Média dos Eventos Recebidos
**Média do numero de rollbacks executados
***Média de simulações simultâneas
****Média do máximo de simulações simultâneas

Foram executadas 10 instâncias de cada comportamento simulado de jogador em cada uma das duas máquinas, para um total de 20 jogadores simulados presentes em todas as sessões de jogo. Para estes testes, foram desligados os gráficos do jogo, e o uso de CPU variou geralmente entre 20% e 40% em ambas as máquinas. A computação dividida de *rollbacks*, incluindo execução corretiva, foi implementada dividindo-se cada *rollback* em um conjunto de "tarefas de *rollback*" menores, e executando no máximo três tarefas para cada passo de simulação otimista de tamanho *So*. As tarefas são de dois tipos: a primeira é uma operação de cópia completa do estado (em memória) do simulador conservador sobre o simulador otimista; a segunda é a execução corretiva de um passo do simulador otimista. Foi determinado que a tarefa de cópia fosse executada no primeiro intervalo de tempo disponível para execução de *rollbacks* (por exemplo, em intervalos de tamanho *So*), e nos intervalos seguintes executando até três passos de execução corretiva de eventos do simulador otimista. Isto significa que, por exemplo, um *rollback* de *To* = 1000ms para *Tc* = 0ms é dividido em 21 tarefas de *rollback* (1 cópia e 20 passos de execuções corretivas) e leva 8 intervalos de tempo de tamanho *So* = 50ms, ou 400ms de tempo real, para ser concluído.

Como visto nas tabelas 1 e 2, com o aumento da latência da rede houve também um aumento do número de *rollbacks* e de execuções corretivas. Esse aumento, por conseqüência, ocasionou picos maiores de simulações simultâneas. Percebe-se também, como mostrado no último caso de teste de cada tabela, que a existência de ao menos um nodo com latência de rede elevada faz com que todos os outros nodos aumentem significativamente o número de *rollbacks* executados e o tempo de processamento gasto com as execuções corretivas. Isso acontece porque enquanto todos os nodos não receberem os eventos atrasados desse nodo com latência de rede alta, suas simulações conservadores não podem avançar do tempo *Tc* para o *Tc* + *Sc*.

5. Comparação

Esta seção descreve brevemente o TSS, o mecanismo de sincronização e execução de eventos distribuídos que serviu de inspiração principal para a técnica apresentada nesse artigo, além de oferecer uma comparação geral do TSS com o trabalho apresentado.

O TSS é um mecanismo de sincronização de simuladores distribuídos para arquiteturas com servidores-espelho (*Mirrored-server architectures*). Neste modelo de distribuição, os nodos clientes representam as máquinas dos jogadores, e os nodos servidores, representando uma infra-estrutura que provê uma única simulação ou jogo para um dado conjunto de servidores. A conexão entre um nó cliente e um nó servidor pode ter atrasos grandes, porém, a conexão entre os vários servidores deve possuir um atraso reduzido. Cada servidor é um "espelho" (*mirror*) que possui a informação completa da simulação em andamento. A premissa é que os clientes, ao conectarem-se ao "espelho" mais próximo, com menor atraso de rede, obterão uma qualidade de jogo melhor. Em princípio, o sistema de espelhos do TSS parece ser escalável, desde que a infra-estrutura de rede necessária, entre os servidores, esteja disponível.

O modelo TSS determina que cada nó execute vários simuladores simultaneamente, onde cada simulador possui um atraso constante para a execução dos eventos recebidos. Em outras palavras, todos simuladores do TSS são otimistas, mas alguns são "mais otimistas" do que os outros. Por exemplo, o simulador *0* de um nó pode executar todos os eventos imediatamente após a recepção de cada um, mas o simulador *1*, do mesmo nó, executa todos os eventos com 100ms de atraso, em relação ao *timestamp* do evento. Um evento recebido fora de ordem no TSS faz com que um ou mais simuladores iniciem um *rollback*

seguido de execução corretiva de eventos até o ponto relativo do tempo de simulação em que eles devem idealmente ficar. A principal diferença entre a proposta apresentada nesse artigo e o TSS é o mecanismo aqui proposto, que vai sempre garantir um estado sincronizado devido ao uso de um simulador conservador, ao passo que no TSS, se os eventos são atrasados por mais tempo do que o atraso de execução do simulador mais atrasado, então não existirá nenhum estado anterior para a realização de *rollback* e eventuais inconsistências oriundas da execução fora de ordem de eventos não podem ser restauradas.

O TSS foi prototipado sobre o jogo de ação Quake e os resultados apresentados pelos autores mostram que o TSS é adequado para estes jogos se as diferenças entre os atrasos configurados para os simuladores não forem muito grandes (no artigo que descreve o TSS, existem diferenças de até 900ms). Nesse trabalho, isto se traduz no maior atraso de rede entre quaisquer dois nodos, o que basicamente determina o atraso entre o simulador conservador e o simulador otimista em cada nodo.

6. Conclusão e Trabalhos Futuros

Neste trabalho, foi apresentada uma técnica de simulação, derivada do TSS, cujo objetivo é servir como suporte para MMOGs. Essa técnica utiliza os modelos de distribuição cliente-servidor e ponto-a-ponto para atender a um modelo particular de instanciação de espaços virtuais em MMOGs. Os experimentos foram focados na obtenção de medidas de uso de CPU relativas, sendo essas baseadas em contagens de eventos de interesse. Estas medidas poderiam ser combinadas com medidas reais de uso de CPU por parte de motores de física para que fosse possível obter dados mais concretos sobre o uso de CPU da técnica apresentada quando aplicada sobre jogos comercias "reais". Porém, se houver recursos de CPU suficientes disponíveis nas máquinas dos jogadores, a técnica proposta deve suportar os "ambientes de ação" dos MMOGs instanciados, visto que é semelhante ao TSS e ele por sua vez é adequado para jogos de ação em pequena escala reais [CRONIN, 2004], como o Quake.

Os resultados obtidos para “Média dos máximos de simulações simultâneas” podem ser considerados relativamente altos, se considerado que um valor igual a "1" corresponde à carga de CPU em um nodo inserido em uma implementação tradicional de jogo cliente-servidor. Porém, os valores obtidos entre 2 e 3 para os valores médios apontam para a viabilidade da técnica. Além disso, foi determinado que 5% dos eventos executados fora de ordem introduzem erros. Isto corresponde a um evento por segundo por jogador, visto que os jogadores enviam 20 pacotes de eventos por segundo (50ms). Em relação a esta questão, será desenvolvido como trabalho futuro o módulo que determina um ordenamento parcial entre os eventos do jogo, através do emprego de esferas de influência, especificamente para jogos de tiro. Espera-se que com isto ocorra uma queda no número de picos de *rollbacks* e execuções corretivas de eventos, tornando assim o modelo compatível com o nível de uso de CPU dos motores de física atuais de jogos de tiro.

Com relação ao modelo instanciado para jogos massivos multi-jogador, a principal preocupação durante o desenvolvimento desta técnica de simulação ponto-a-ponto, para os "ambientes de ação", foi evitar as trapaças de conluio. Este tipo de trapaça pode ser altamente danoso para MMOGs descentralizados, independente do tipo de jogo. Porém, o protocolo como descrito neste trabalho, deixa em aberto várias outras oportunidades de trapaça, algumas das quais são tratadas por outros trabalhos [BAUGHMAN, 2001; KABUS, 2005]. Prevenir outros tipos de trapaças é deixado como trabalho futuro.

Referências Bibliográficas

ARENANET 2006. Guild wars. http://www.guildwars.com/ [Acessado em Agosto 2006].

BAUGHMAN, N.E. E LEVINE, B.N., 2001. Cheat-proof playout for centralized and distributed online games. Em *IEEE INFOCOM*, volume 2, páginas 104–113, Abril 2001.

BERNIER Y.W., 2001. Método de compensação de latência no design e otimização de jogos com o protocolo cliente/servidor. Em *Proceedings of the Game Developers Conference*, Fevereiro 2001. http://www.gdconf.com/archives/2001/bernier.doc. [Acessado em Agosto 2006].

BLIZZARD 2005. World of warcraft alcança novo marco de dois milhoes de assinaturas pelo mundo, 2005. http://www.blizzard.com/press/050614-2million.shtml [Acessado Agosto 2006].

BLIZZARD 2006. World of Warcraft. http://www.worldofwarcraft.com/ [Acessado em Agosto 2006].

CECIN, F., REAL, R., MARTINS, M., JANNONE, R., BARBOSA, J. E GEYER, C., 2004. Freemmg: A scalable and cheat-resistant distribution model for internet games. Em *IEEE Distributed Simulation and Real-Time Applications*, páginas 83–90, 2004.

CRONIN, E., KURC, A.R., FILSTRUP, B. E JAMIN, S., 2004. An efficient synchronization mechanism for mirrored game architectures. *Multimedia Tools Appl.*, 23(1):7–30, 2004.

DUONG, T.N.B. E ZHOU, S., 2003. A dynamic load sharing algorithm for massively multiplayer online games. Em *IEEE International Conference on Networks*, páginas 131–136, 2003.

GAUTIER, L. E DIOT, L., 1997. Mimaze, a multiuser game on the internet. Research report: RR-3248, INRIA, Setembro. 1997.

GAUTIER, L. E DIOT, L., 1998. Design and evaluation of imaze, a multi-player game on the internet. Em *IEEE International Conference on Multimedia Computing and*

Systems, página 233, Washington, DC, USA, 1998. IEEE Computer Society.

KABUS, P., TERPSTRA, W.W., CILIA, M. E BUCHMANN, A.P., 2005. Addressing cheating in distributed mmogs. Em *ACM SIGCOMM workshop on Network and system support for games*, páginas 1–6, New York, NY, USA, 2005. ACM Press.

MILLS, D.L., 1992. Network Time Protocol (versão 3) pecification, implementation and analysis. RFC 1305, 1992.

OHLENBURG, L., 2004. Improving collision detection in distributed virtual environments by adaptive collision prediction racking. Em *IEEE Virtual Reality 2004 (VR'04)*, página 83, Washington, DC, USA, 2004. IEEE Computer Society.

RITARI, N., RIVINOJA, J., CECIN, F. E CONTRIBUIDORES, 2006. Outgun, 2006. http://koti.mbnet.fi/outgun/ [Acessado em Agosto 2006].

SONY. PLANETSIDE, 2006. http://www.planetside.com/ [Acessado em Agosto 2006].

VLEESCHAUWER, B.D., BOSSCHE, B.V.D., VERDICKT, T., TURCK, F.D., DHOEDT,, B. E DEMEESTER, P., 2005. Dynamic microcell assignment for massively multiplayer online gaming. Em *ACM SIGCOMM workshop on Network and system support for games*, páginas 1–7, New York, NY, USA, 2005. ACM Press.

YAN, J. E RANDELL, B., 2005. A systematic classification of cheating in online games. Em *ACM SIGCOMM workshop on Network and system support for games*, páginas 1–9, New York, NY, USA, 2005. ACM Press.