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

JOÃO VICENTE FERREIRA LIMA

Estudo sobre MPI-2 e Threads

Trabalho Individual I
TI-666

Prof. Dr. Nicolas Maillard
Orientador

Porto Alegre, dezembro de 2007

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

MPI	Message-Passing Interface
SMP	Symmetric Multi-Processor

LISTA DE FIGURAS

RESUMO

Este documento é um exemplo de como formatar documentos para o Instituto de Informática da UFRGS usando as classes LaTeX disponibilizadas pelo UTUG. Ao mesmo tempo, pode servir de consulta para comandos mais genéricos. *O texto do resumo não deve conter mais do que 500 palavras.*

Palavras-chave: MPI, Threads, Programação Paralela, Sistemas Operacionais.

Using LaTeX to Prepare Documents at II/UFRGS

RESUMO

This document is an example on how to prepare documents at II/UFRGS using the LaTeX classes provided by the UTUG. At the same time, it may serve as a guide for general-purpose commands. *The text in the abstract should not contain more than 500 words.*

Palavras-chave: Electronic document preparation, LaTeX, ABNT, UFRGS.

1 INTRODUÇÃO

2 UMA ARQUITETURA DISTRIBUÍDA PARA MMORPG

Neste trabalho, (**?**) apresentam uma abordagem para o suporte a jogos de rpg online maciçamente multijogador. Sua proposta começa por dividir o grande mundo virtual em regiões menores, cada uma atribuída a um diferente servidor. São apresentados algoritmos que (1) reduzem a largura de banda necessária para tantos os servidores quanto os clientes, (2) tratam problemas de consistência, hotspots, congestionamento e defeito do servidor, tipicamente encontrados em MMORPG e (3) permitem interação seamless entre jogadores situados em áreas atribuídas a diferentes servidores.

O objetivo da arquitetura é dar suporte a um grande número de usuários simultâneos. Seu design permite crescimento irrestrito do ambiente virtual, ao mesmo tempo em que permanece prático e pragmático no que diz respeito a como jogos MMORPG são implementados atualmente. Baseia-se no fato de que MMORPG apresenta forte localidade de interesse e, dessa forma, pode-se dividir o grande mundo virtual em regiões menores. Múltiplos servidores, ainda sobre o controle centralizado do produtor do jogo, são designados a lidar com tais regiões. Uma divisão estática, no entanto, não será capaz de reagir a repentinos picos de carga causada pelos assim chamados hotspots. Seu design permite a reorganização da divisão sem interromper o jogo significativamente. Também são propostos algoritmos e técnicas para lidar com interações entre jogadores situados em regiões atribuídas a diferentes servidores.

2.1 Definições

Para entender o modelo formal apresentado pelos autores, algumas definições foram feitas. Define-se um personagem do jogo controlado por um jogador humano como jogador e um personagem controlado por I.A. como NPC (non-player-character). Considere-se o mundo virtual W, que pode ser modelado como um mapa geográfico 2D, apesar de poder ser facilmente estensível a 3D. Jogadores, NPCs e itens no ambiente virtual são todos considerados objetos do jogo. Cada objeto i W tem um conjunto de coordenadas Ci(x,y) no mapa, assim como um estado Si, que é tratado simplesmente como um conjunto de bits. Cada objeto animado tem uma função de trajetória fi S que descreve o atual movimento do objeto no mundo em função do tempo.

2.1.1 Eventos

Cada objeto segue sua trajetória, a não ser que um evento do jogo ocorra. Um evento é uma transação atômica que acontece no mundo e muda o estado de um ou mais objetos. Eventos são discretos e de duração nula. Operações não-instantâneas são tratadas como um conjunto E de eventos, onde |E| > 1 e, onde necessário, encapsula movimento em

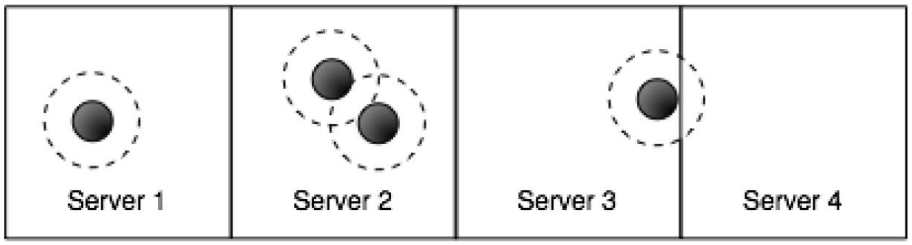


Figura 2.1: O conceito de área de interesse com múltiplos servidores

trajetórias. Como um exemplo, pode ser considerado o lançamento de um míssil. No modelo proposto, a operação toda não é apenas um evento, mas dois eventos: (1) o disparo inicial de um míssil, juntamente com a criação do objeto míssil e (2) o impacto final e a destruição do objeto míssil, assim como quaisquer outros objetos afetados pelo impacto. O caminho percorrido pelo míssil é encapsulado em sua trajetória.

Há dois tipos de eventos no sistema: (1) eventos causados por entradas de um jogador humano e (2) eventos causados pelas regras e I.A. do jogo. Além disso, é importante notar que eventos podem atuar em objetos apenas dentro de uma faixa R - onde R é o maior valor entre todos os alcances de eventos e capacidades sensoriais do jogador.

2.1.2 Ações

Um grande número de eventos no jogo são causados pelas entradas de jogadores. Diz-se que entradas de jogadores constituem uma ação que, por sua vez, cria um evento. Por exemplo, se um jogador deseja executar uma ação, tal como disparar um míssil, o cliente envia a ação ao servidor ao qual está conectado. O servidor cria um evento e atualizações de estado de todos os objetos afetados conforme necessário. O cliente é então notificado destas mudanças de estado e atualiza sua cópia local.

Cada ação enviada pelo cliente ao servidor carrega um número de ID monotonicamente crescente. Números de ID são utilizados em alguns cenários para garantir corretude e, assim, manter consistência.

2.1.3 Área de interesse

Provém naturalmente do estilo do MMORPG que jogadores estejam interessados apenas em eventos que ocorrem dentro de suas capacidades sensoriais. Um cliente não precisa receber eventos que o jogador não pode ver ou ouvir. Pode-se então definir uma área de interesse para um jogador como o conjunto de todos os pontos a uma distância de, no máximo, R de sua localização. Na Figura 2.1, da esquerda para a direita, a área de interesse do primeiro jogador reside completamente dentro do primeiro servidor. O segundo e terceiro jogadores também estão dentro do mesmo servidor, suas áreas de interesse interceptam um ao outro, que podem se ver. A área de interesse do quarto jogador se estende do servidor 3 ao 4.

2.2 Design

A arquitetura geral do sistema é baseada sobre a localidade espacial de interesse exibida pelos jogadores. Clientes podem ser agregados dependendo da localização de seus jogadores no mundo virtual. Assim sendo, o mundo virtual W pode ser divido entre regiões menores e disjuntas w1, w2, w3, ..., wn $\subset$ W - com cada região sendo designada a

um diferente servidor, como mostrado na Figura 2.1. A região designada a cada servidor pode ser qualquer polígono convexo. As múltiplas regiões menores são transparentes ao jogador, que apenas vê um grande mundo virtual.

Todo o tempo, o cliente tem apenas um ponto principal de contato: o servidor ao qual ele manda suas ações. Para cada cliente, o servidor que atua como ponto único de contato é determinado pela localização do jogador no ambiente virtual. O cliente continua a mandar ações para o mesmo servidor, até segunda ordem do mesmo. Apesar dos clientes enviarem ações a apenas um servidor, eles podem receber eventos de múltiplos servidores. Este é o caso em que jogadores estão localizados a uma distância menor que R de outra região.

A infra-estrutura do sistema consiste de múltiplos servidores, todos interconectados com uma latência média de Ls para comunicação entre servidores. Seja Lp a latência média de cliente para servidor e assuma-se que Lp $>>$ Ls. A suposição de latência é justificada no fato de que clientes tipicamente operam conectados à Internet utilizando banda larga ou conexões discadas. Produtores de jogos instalam servidores no mesmo datacenter ou interconectam-nos utilizando backbones de alta velocidade e baixa latência. Também assume-se que existe um servidor de login aos quais os clientes conectam-se inicialmente e de lá são redirecionados.

2.2.1 Mecanismos de travamento

2.2.1.1 Travas de região

Para atacar os vários desafios de consistência que surgem, foi introduzido o conceito de travas de região. Trata-se de travas sobre áreas geográficas no mundo virtual. A auotridade para garantir travas de cada região cabe inteiramente ao servidor a ela designado. Um servidor executando um evento que afeta uma área específica no mapa (e.g. uma explosão de uma bomba), poderia requisitar uma trava sobre aquela área. Uma vez que um servidor recebe a trava, outros servidores requisitando uma trava sobre uma área que se sobrepõe à área travada pelo primeiro servidor terão que esperar em uma fila.

2.2.1.2 Travas de objeto

Quando são processados eventos próximos a limites das regiões dos servidores, além das travas de região descritas, pode ser necessário obter travas para os objetos afetados. Quando um servidor obtém uma trava sobre um objeto, outro servidor requisitando o mesmo objeto para executar um evento terá que que esperar até que o primeiro servidor libere a trava.

Mais uma vez, a autoridade que garante ou libera uma trava é aquele que está servindo no momento a região ou objeto e gerenciando seu estado.

2.2.2 Mecanismo de anúncio de eventos

No design proposto pelos autores, os servidores são tanto assinantes como publicantes, assim como clientes são apenas assinantes, dos eventos que ocorrem. A subscrição é baseada em região; um assinante pode subscrever para receber eventos que ocorrerem apenas em uma pequena região.

No que diz respeito a eventos entre servidores, cada servidor S é assinante das regiões de todos seus vizinhos, até uma distância R das fronteiras de S. Por exemplo, seja um mundo virtual gerenciado por dois servidores, S1 e S2, tal que S1 gerencia a metade da direita e S2, a da esquerda. Seguindo a abordagem dos autores, S1 será assinante dos

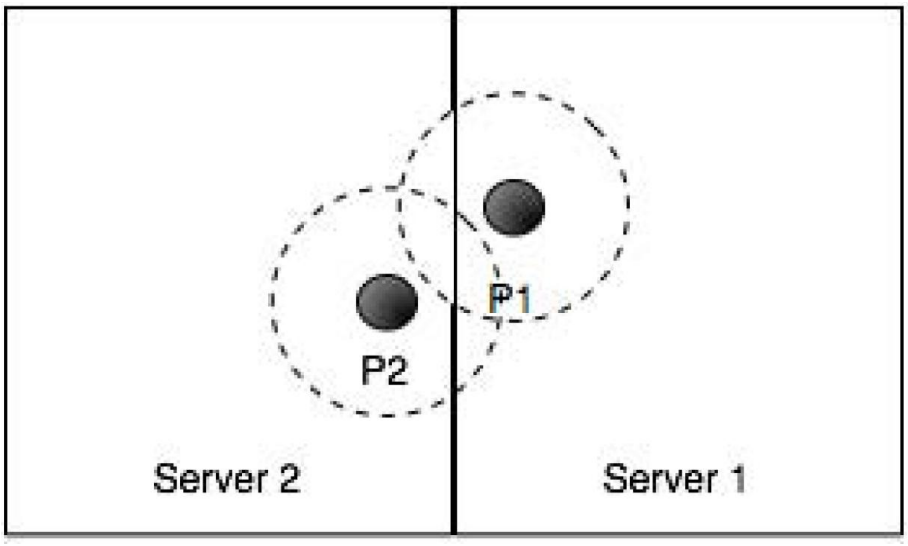


Figura 2.2: Dois jogadores em servidores diferentes interagindo entre si

eventos que ocorrerem na região mais à direita na região de S2, assim como S2 será assinante dos eventos que ocorrerem na região mais à esquerda na região de S1, sempre com uma distância máxima de R. Dessa forma, cada servidores irá sempre notificar o outro a cerca de eventos que ocorrerem a uma distância igual ou menor que R da fronteira da região gerenciada pelo seu vizinho. Tal característica, de ter servidores sendo notificados a respeito de eventos ocorrendo em seus vizinhos, é uma parte importante da arquitetura proposta.

Subscrição entre servidores permite uma experiência de jogo suave para os jogadores, mesmo em áreas próximas a fronteiras entre regiões. Para cada jogador situado em uma dada região de um servidor, este executa duas funções importantes: (1) processar ações recebidas dos clientes e (2) assinar/cancelar assinatura de eventos para cada cliente, de acordo com sua respectiva área de interesse. Em um cenário onde há um único servidor, esse iria subscrever e dessubscrever o cliente de acordo com a movimentação do jogador através do ambiente virtual, de forma que ele só recebesse eventos relevantes à área de raio R ao redor de suas coordenadas. Por exemplo, um jogador P1 pode entrar na área de interesse do jogador P2. O servidor recebe a ação de andar do jogador P1 e anuncia, para P1 e P2, o evento de que P1 andou. Se ambos os jogadores olharem na direção correta, poderão se ver mutuamente.

Num cenário com múltiplos servidores, sempre que as coordenadas de um jogador estiverem no máximo a uma distãncia R de um ponto na região de outro servidor, sua área de interesse pode se estender através de mais de um servidor e, dessa forma, o cliente deveria receber eventos de todos os servidores relevantes. Um exemplo trivial, mostrado na Figura 2.1, seria de um jogador caminhando em uma área gerenciada pelo servidor S3, indo em direção a uma área gerenciada por S4. Assim que o jogador está dentro da distância R da fronteira de S4, e porque S4 está subscrito para receber os eventos de S3, S4 irá saber da presença do jogador. S4 irá então, automaticamente, subscrever o cliente do jogador para receber eventos que aconteçam dentro da porção da área de interesse gerenciada por ele (S4). Analogamente, se o jogador começar a caminhar para longe de S4, então S4 receberá um evento de S3 de que o jogador se moveu e S4 irá cancelar a assinatura dos eventos de sua região para aquele cliente. Note-se que, mesmo que um jogador possa estar assinando os eventos - e, portanto, sendo notificado dos mesmos -

de múltiplos servidores, ele apenas envia ações para um único servidor. No exemplo da Figura 2.1, ele apenas enviaria ações para o servidor S3.

2.2.3 Eventos próximos à fronteira do servidor

São examinados algoritmos para lidar com a complexidade do cenário de múltiplos servidores, assim como prover os requisitos de consistência necessários para garantir corretude.

2.2.3.1 Requisitos de consistência

Para manter a corretude, dois requisitos de consistência devem ser sempre satisfeitos:

1. A ordem dos eventos que afetam o estado de qualquer dado objeto deve ser a mesma para todos os clientes;
2. Existe uma ordem global de eventos que é consistente com a ordem dos evendos que atuaram sobre cada objeto. Isto é, deve ser possível associar números a todos os eventos, tal que a seqüência dos números dos eventos que ocorreram com qualquer objeto é monotonicamente crescente.

2.2.3.2 Eventos

Para atacar o problema dos eventos que afetam áreas em múltiplos servidores, é usado o conceito de travas de região e o mecanismo de anúncio de eventos, descritos anteriormente. Seja um evento originado em um servidor S2 que afeta também áreas nos servidores S1 e S3. O algoritmo proposto executa o seguinte:

1. S2 requisita travas de região para as áreas geográficas afetadas pelo evento e travas de objeto para todos os objetos que participam do mesmo. Para prevenir deadlock, comece requisitando tanto travas de região e de objetos em ordem decrescente de ID. No exemplo, S2 requisita e obtém todas as travas de S3 primeiro, então obtém suas próprias travas e então as de S1. Travas são requisitadas por cada servidor de uma maneira "tudo-ou-nada- S2 irá requisitar todas as travas necessárias em cada passo e não irá continuar a requisitar travas de outro servidor até que todas estejam disponíveis primeiro.
2. S2 executa o evento atômico e notifica S1 e S3 sobre como eles devem mudar seus estados.
3. S2 libera todas travadas adquiridas.
4. Clientes recebem todas as atualizações de estado através do mecanismo de anúncio de eventos descrito.
5. Uma vez que o cliente recebe o resultado do evento de todos os servidores aos quais está subscrito, ele atualiza o estado de sua cópia local e atualiza a representação visual do jogo para o jogador, se necessário.
6. Se um cliente recebe um novo evento antes de receber o resultado do evento anterior de todos os servidores, então ele enfileira o novo evento e o executa assim que for notificado do evento anterior por todos os servidores.

No caso de o mundo virtual ser dividido em retângulos, o número de servidores envolvidos na execução de cada evento não é maior que 4. Dessa forma, o atraso adicional máximo introduzido é O(Ls).

2.2.3.3 Transferências de objetos

Um caso especial de evento ocorre quando um objeto se move de uma região gerenciada por um servidor para a região gerenciada por um outro servidor. A solução proposta pelos autores para este caso é bastante similar ao algoritmo descrito na seção anterior. No entanto, executar o evento inclui transferir o estado do objeto de um servidor para o outro. O cenário mais complexo ocorre quando o objeto é, na verdade, um jogador. Neste caso, não apenas é necessário trasferir o estado entre os servidores, como também os eventos que ocorrem durante a transferência devem ser processados normalmente. Considerando o cenário de um jogador P, movendo-se de S1 a S2. Propõe-se um algoritmo para ser executado por S1 com os seguintes passos:

1. S1 requisita travas de região para a pequena área que o jogador P irá percorrer, começando com o servidor que tem o maior ID. Neste exemplo, S1 requisita e obtém uma trava de S2 primeiro antes de obter sua própria trava, de S1.
2. Obtém uma trava de objeto de P.
3. S1 envia a S2 o ID da última ação processada de P.
4. S1 inicia a transferência do estado de P para S2.
5. Com o término da transferência, S1 libera as travas adquiridas anteriormente.
6. S1 notifica P de que S2 é agora seu novo ponto de comunicação.
7. Se P iniciar quaisquer eventos enquanto os passos 4, 5 ou 6 estiverem sendo executados, eles são encaminhados a S2.

Analogamente, o servidor de destino S2 executa os seguintes passos:

1. Permite que S1 adquira as travas de região.
2. Recebe o ID da última ação de P processada por S1 e a armazena como PlastID
3. Aceita a transferência de estado.
4. Quando a transferência de estado é completada, aceita a conexão de P.
5. Se S2 receber uma ação do jogador P com ID > PlastID + 1, enfileira. Se ID = PlastID + 1, processa a ação e termina o algoritmo.
6. Em qualquer momento, se uma ação de P é encaminhada para S2 por S1, processa-a e incremente o valor de PlastID.
7. Quando o primeiro evento na fila tiver um ID igual a PlastID + 1, processa todos os eventos na fila e termina o algoritmo.

O mecanismo de rastreamento com ID do evento garante que os requisitos de consistência definidos seja garantidos, mesmo que alguns eventos encaminhados de S1 para S2 cheguem ligeiramente atrasados.

Da perspectiva de um jogador, tudo continua de maneira suave. Um cliente continua enviando ações para S1 a não ser que seja notificado para agir de maneira diferente no passo 6 do primeiro algoritmo. O design com múltiplos publicantes permite que os

clientes continuem recebendo notificações de eventos ininterruptamente e com apenas um atraso adicional médio de Ls. Como Ls « Lp, o atraso não é perceptível.

O algoritmo funciona também com objetos que não sejam jogadores humanos. Em tais casos, não há necessidade de notificar o objeto da mudança de servidor - o novo servidor torna-se responsável por aplicar as regras da I.A. do jogo assim que for recebido com sucesso o estado do objeto.

2.2.4 Abortando eventos

Devido aos requisitos de consistência ou atrasos introduzidos durante as transferências através da rede, pode ser o caso de os pré-requisitos para a execução de um evento não serem mais válidos. Por exemplo, dois jogadores tentam apanhar o mesmo objeto aproximadamente ao mesmo tempo. O servidor tentará obter a trava em favor do cliente cuja ação foi recebida primeiro. O servidor irá então executar o evento e notificar o primeiro cliente que ele conseguiu apanhar com sucesso o objeto. Após fazer isso, o servidor libera as travas e tenta travar o mesmo objeto em favor do segundo cliente. No entanto, o objeto já foi apanhado pelo primeiro jogador e, naturalmente, o servidor não será capaz de adquirir as travas. Dessa forma, será abortado o evento em favor do segundo cliente, que será notificado que o objeto não está mais disponível.

2.3 Escalabilidade

2.3.1 Hotspots

Para lidar com congestionamentos e hotspots imprevisíveis, que ocorrem tipicamente em MMORPG, os autores propõem um algoritmo para o particionamento da área gerenciada por um servidor em duas ou mais partes. As partes congestionadas podem ser designadas ou a servidores novos ou a servidores já existentes, dependendo da capacidade atual do sistema.

Um exemplo seria o de dois servidores, S1 e S2. S1 está operacional mas tem uma pesada carga, já que ele está responsável por um grande número de objetos e está lidando com uma número inusitadamente alto de eventos. S2 não está operacional ainda e está sem nenhum estado. O algoritmo proposto executa os seguintes passos:

1. S1 designa uma área como pertencente a S2.

2. S1 começa a transferir dados do jogo a S2. Os dados do jogo consistem em objetos que podem ser serializados e enfileirados através da rede. Apenas dados do jogo na área designada no passo 1 como pertencente a S2 são transferidos.

3. Se, enquanto a transferência estiver ocorrendo, um objeto já transferido sofrer uma atualização de estado, S1 envia a cópia atualizada a S2.

4. Assim que a transferência dos dados terminar, S2 pode imediatamente tornar-se operacional, posto que ele tem todos os dados do jogo necessários, incluindo trajetórias de objetos.

5. S1 dispara o particionamento e S2 começa a publicar eventos.

6. Os últimos dois passos do algoritmo de transferência de jogador (visto anteriormente) são executados para cada jogador que está agora localizado em uma área gerenciada por S2.

O atraso médio adicional introduzido enquanto a transferência está ocorrendo é de Ls, onde Ls $<<$ Lp. Assim, mesmo para um grande número de participantes, a transição seria transparente para os jogadores.

2.3.2 Escalabilidade

Os autores sugerem um mecanismo simples para agregar um novo servidor Sn ao sistema:

1. Sn subscreve-se aos servidores a ele adjacente para todas as regiões distando até R de sua fronteira.

2. Todos os servidores adjacentes a Sn subscrevem-se para receber os eventos que ocorrerem nas regiões numa distância máxima R de suas respectivas fronteiras.

3. Sn torna-se operacional.

Para prover escalabilidade, os autores tratam apenas da questão do tamanho do ambiente virtual, de forma a caber mais servidores e mais jogadores no mesmo. No entanto, seria necessário também tratar do problema do aumento do número de jogadores na mesma área do ambiente.

2.3.3 Tolerância a falhas

Jogadores de MMORPG esperam que os servidores do jogo estejam constantemente disponíveis com pouco ou nenhum período sem serviço. Os autores propõem um esquema de espelhamento que pode ser usado para tratar de falha dos servidores: Um servidor reserva Sb é designado a cada servidor operacional So, que atua como servidor primário. É usado então um método similar ao utilizado no algoritmo de particionamento de região descrito anteriormente. Dessa vez, no entanto, Sb é assinante de todos os eventos de atualização em toda a região geográfica gerenciada por So. Assim, So anuncia todos os eventos a Sb antes de anunciá-los a qualquer outro assinante. Além disso, So informa Sb cada vez que ele adquire ou libera uma trava. Diferente do algoritmo de particionamento descrito anteriormente, os jogadores não são notificados da presença de Sb, a não ser que seja detectado queda de So. Neste caso, Sb, que possui todos os dados do jogo, executa o passo 5 daquele algoritmo. A transferência dos dados em si é O(Ls). Com um esquema adequado de detecção de defeitos, o atraso total entre o defeito do servidor primário e o servidor reserva estar completamente operacional pode ser tão curto quanto o atraso de uma comunicação entre servidores.

2.4 Avaliação do trabalho

A primeira grande restrição da arquitetura proposta por (**?**) é a de que a latência e a largura de banda disponível entre os servidores é muito menor que aquela entre os clientes e o servidor. Tal pressuposto baseia-se no fato de que as máquinas servidoras estão em rede local, em algum datacenter pertencente ao produtor do jogo. Pode não ser viável a solução proposta pelos autores em um cenário onde os servidores estejam geograficamente distribuídos e conectados através de um canal com capacidade mais limitada.

Além disso, os autores sugerem tratar da escalabilidade apenas aumentando o tamanho do mundo virtual, onde caberiam mais jogadores. No entanto, é necessário haver alguma forma de lidar com um grande número de jogadores em uma mesma região do ambiente

do jogo. A abordagem proposta para tratar estes hotspots sugere que o ambiente seja particionado recursivamente, enquanto houver novos servidores disponíveis, até que não haja mais sobrecarga. No entanto, há um limite para este reparticionamento, pois implica em mais comunicação entre servidores. É necessário, pois, avaliar onde está este limite e de que forma lidar quando se estiver próximo dele.

3 UMA ARQUITETURA MULTI-SERVIDOR PARA WALK-THROUGH VIRTUAL DISTRIBUÍDO

Neste trabalho, Beatrice Ng et al. apresentam a arquitetura do CyberWalk (**?**), um sistema de walkthrough virtual distribuído desenvolvida pelos autores. Ele permite que usuários em diferentes locais no globo compartilhem informação e interajam dentro de um ambiente virtual comum através de uma rede local ou da Internet. Para permitir boa performance do sistema, são utilizados múltiplos servidores e é empregada uma técnica de particonamento adaptativo dos dados que particiona dinamicamente o ambiente virtual inteiro em regiões. Todos os objetos dentro de cada região serão gerenciados por um servidor. Sob circunstâncias normais, quando um usuário está explorando uma região, o servidor daquela região será responsável por responder a todos os pedidos do usuário. Quando um usuário está cruzando a fronteira entre duas ou mais regiões, os servidores das regiões envolvidas estarão respondendo às requisições do usuário, já que ele pode visualizar objetos dentro de todas as regiões. Informações a respeito de objetos virtuais, incluindo suas localizações e formas, são mantidas em um servidor de banco de dados central.

3.1 Visão geral do CyberWalk

O sistema é implementado com baso no conceito de transmissão sob demanda, que possui algumas vantagens em relação a outros sistemas existentes: redução do custo de transmissão através de envio progressivo de modelos e a introdução dos escopos do observador e do objeto; redução do custo de renderização através da modelagem de objetos com múltiplas resoluções e aumento da interatividade e disponibilidade do sistema através de caching e prefetching.

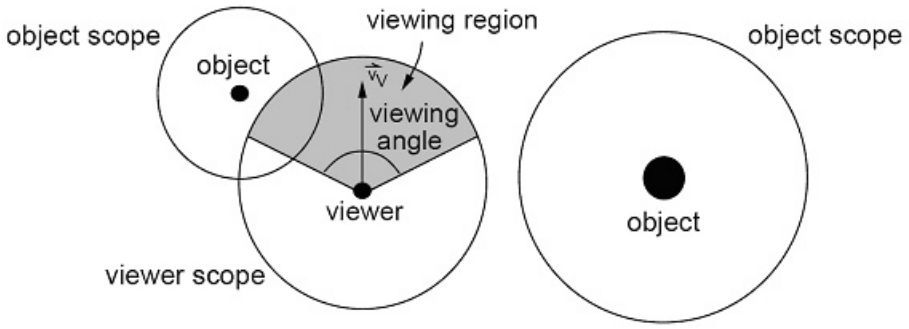


Figura 3.1: Escopo de objeto (object) e escopo de observador (viewer)

- Escopo de observador e escopo de objeto: para reduzir a quantidade de dados necessários para enviar, é generalizado o conceito de área de interesse para ambos observador e objeto, chamados de escopo de observador e escopo de objeto. Um escopo de observador indica quão longe o observador pode ver. Um escopo de objeto indica quão longe um objeto pode ser visto. Seu tamanho é proporcional ao tambanho do objeto. Em geral, um observador pode também ser considerado um objeto e ter um escopo de objeto a ele atribuído, além do próprio escopo de observador. Um objeto pode apenas ser visto por um observador quando os dois escopos se sobrepõem. Além disso, objetos cujos escopos não se sobrepõem com o escopo do observador não precisam ser transferidos para o cliente, de modo a economizar largura de banda de transmissão, processamento e memória necessária. Os autores definem um escopo como uma região circular caracterizada por um raio, como mostrado na Figura 3.1.

- Modelagem com múltiplas resoluções para envio e renderização progressivos: enviar os modelos completos apenas daqueles objetos visíveis à máquina do cliente sob demanda pode ainda causar uma possível longa pausa no walkthrough. Ao invés disso, o sistema proposto codifica cada modelo de objeto como uma malha progressiva. Durante a execução, a distância entre os centros do escopo do observador e o escopo do objeto determina a resolução da malha progressiva necessária à máquina do cliente. Como a visibilidade de um objeto geralmente muda apenas um pouco entre um quadro e outro, apenas um pequeno número de registros progressivos precisam ser transmitidos ao cliente entre quadros consecutivos.

- Caching com múltiplas resoluções: foi desenvolvida uma técnica de caching para permitir uma granularidade fina de caching e substituição de objetos. Um mecanismo de caching permite que um cliente utilize sua memória e armazenamento local para guardar objetos que estão visíveis e que provavelmente continuarão visíveis num futuro próximo. Um cache local também suporta um certo grau de operação desconectada quando a Internet estiver temporariamente indisponível.

- Prefetching: um mecanismo de prefetching permite que um cliente prediga objetos que provavelmente estarão visíveis no futuro e os obtenha antecipadamente para melhorar o tempo de resposta. Considerando o fato de que a maior parte dos usuários estaria utilizando um PC equipado com um mouse 2D para navegação 3D, foi desenvolvido um método de previsão de movimento para predizer o futuro movimento do observador, baseado no padrão de movimento do mouse 2D.

3.2 Arquitetura paralela do CyberWalk

Para tratar o problema da sobrecarga do servidor com um grande número de clientes, o CyberWalk emprega uma arquitetura paralela de múltiplos servidores. Uma arquitetura paralela é fundamentalmente uma arquitetura share-nothing (não compartilha nada). Como ilustrado na Figura 3.2, cada servidor gerencia sua própria memória e repositório local de objetos virtuais. Em arquiteturas share-nothing tradicionais, um dos servidores está frequentemente dedicado a ser um coordenador, que tem conhecimento completo dos dados gerenciados em cada um dos outros servidores. O coordenador recebe uma requisição de um cliente e a encaminha ao servidor que gerencia os dados que foram requisitados. No entanto, o coordenador poderia rapidamente tornar-se um gargalo quando

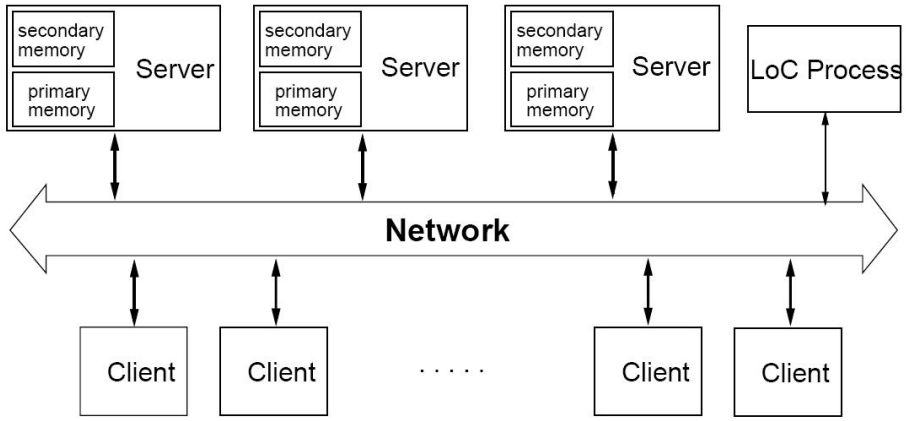


Figura 3.2: Arquitetura paralela do CyberWalk

a freqüência de submissão das requisições se tornar alta. Há outro problema com tal arquitetura: quaisquer mudanças feitas por um cliente podem ter que executar vários passos na comunicação da rede, até que o cliente receba uma resposta. Esse atraso extra pode afetar seriamente a interatividade do sistema, especialmente se os servidores não estiverem localizados na mesma rede local.

No CyberWalk, o ambiente virtual inteiro é particionado em regiões, formando uma matriz bidimensional. Objetos virtuais dentro de uma região serão gerenciados apenas por um servidor. Cada servidor estará respondendo pedidos apenas dos clientes que estiverem explorando objetos dentro da região por ele gerenciada. Sob circunstâncias normais, quando um observador V está explorando a região R1, o servidor de R1, S1, será responsável por responder a todos os pedidos de V. Quando V estiver cruzando a fronteira entre R1 e R2, ou seja, quando o escopo de observador de V tocar a fronteira de R2, os servidores de todas as regiões envolvidas estarão respondendo pedidos do observador pois ester poderia visualizar objetos dentro de todas essas regiões. Em outras palavras, tanto S1 como S2 estarão servindo pedidos de V. Para conseguir isto, quando o escopo de observador de V tocar a fronteira de R2, S1 enviará uma mensagem a S2 no formato <idv, locv, vv>, onde idv é o ID de V, que é inerentemente o endereço IP do cliente, locv é a localização de V, e vv é a direção de visualização de V. Uma vez que S2 recebeu tal mensagem de S1, ele pode determinar quais objetos em R2 estão visíveis a V e transmitir os correspondentes registros progressivos e/ou malhas base diretamente a ele. Este, então, irá manter canais de comunicação direta tanto com S1 quanto com S2, enquanto seu escopo de observador se sobrepuser a ambas regiões. Quando eventualmente mover-se para a região R2, V irá parar de se comunicar com S1 e irá comunicar-se apenas com S2.

Quando o número de clientes em uma determinada região tiver aumentado substancialmente, a um ponto em que a diminuição da performance é perceptível pelo observador, aquela região deve ser particionada e a nova partição deve ser alocada a um servidor vizinho menos ocupado.

Como mostrado na Figura 3.2, um processo, chamado de LoC (coletor de carregamento), é dedicado a coletar a informação de carregamento de cada servidor. Cada um deles irá periodicamente informar ao processo LoC a respeito de seu então nível de carga, e há a flexibilidade de determinar quão frequente será o envio desta informação ao coletor. Quanto mais rapidamente a carga de um servidor variar, maior será a freqüência de envio de informações ao LoC. Este, assim sendo, possui o mais atualizado conjunto de informações a respeito do carregamento de cada servidor. Quando um servidor está

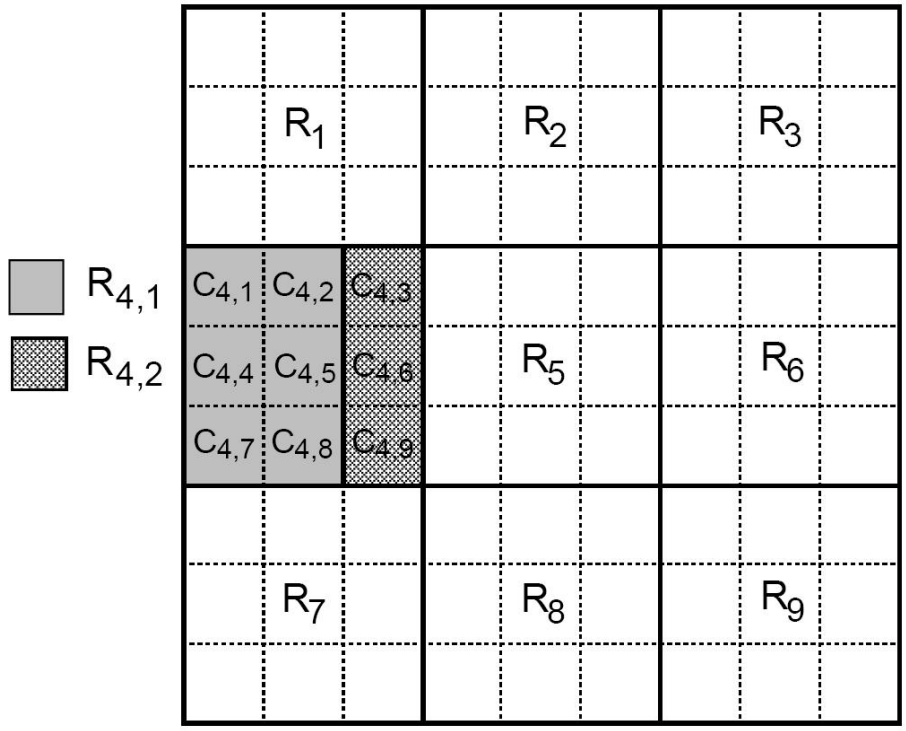


Figura 3.3: Particionamento em regiões

sobrecarregado, ele requisita informações de carregamento de seus vizinhos ao LoC. A partir daí, ele seleciona um de seus vizinhos e - o menos sobrecarregado - para absorver uma porção de sua região.

Para que esta abordagem realmente seja vantajosa, é necessário que:

1. Haja um mecanismo de medição do carregamento em cada servidor e um critério que divida a carga de maneira justa e eficiente, de forma que o servidor que a recebeu uma porção do trabalho do servidor sobrecarregado não seja penalizado com carga acima do que pode suportar.

2. Deve ser feito um particionamento do ambiente virtual, levando em conta que algumas áreas do mundo virtual atrairão mais usuários que outras. É necessário, pois, que o particionamento resulte em regiões que possuam cargas semelhantes. Isto seria o particionamento adaptativo de regiões.

3.3 Balanceamento de carga adaptativo

No sistema proposto, o ambiente virtual inteiro é regularmente subdividido em um grande número de células retangulares. Cada célula, Cij, contém um conjunto de objetos, ou seja, $Cij = \{Ocij1, Ocij2, ..., Ocij|Cij|\}$. O ambiente também é particionado em um conjunto de Nr regiões, ou seja, $ambiente = \{R1, R2, ..., RNr\}$, enquanto cada região contém um número inteiro de células, ou seja, $Ri = \{Ci1, Ci2, ..., Ci|Ri|\}$. A Figura 3.3 ilustra uma partição de 9 regiões, cada uma contendo 9 células e gerenciada por um servidor.

Há diversas maneiras de particionar o ambiente virtual em regiões. A mais simples é dividí-lo em regiões iguais. Cada região irá cobrir o mesmo tamanho geográfico do mundo virtual e conter o mesmo número de células, ou seja, $\forall$ Ri, Rj $\in$ ambiente, |Ri| =

|Rj|. No entanto, como os objetos virtuais podem não ser distribuídos uniformemente dentro do ambiente, algumas regiões podem conter mais objetos do que outras. Isso poderia ser tratado particionando o ambiente com base na "densidade de objetos", que é o número de objetos por célula no mundo virtual. Neste esquema, cada região irá conter aproximadamente o mesmo número de objetos, ou seja, $\sum_{k=1}^{|R_i|} |c_{i,k}| \approx \sum_{l=1}^{|R_j|} |c_{j,l}| \forall R_i, R_j \in \mathsf{VE}$. No entanto, cada região pode cobrir um tamanho geográfico diferente e, portanto, conter um número diferente de células. Esta abordagem busca uma carga uniforme entre todos os servidores, garantindo que irão tratar o mesmo número de objetos. Isto é baseado no pressuposto de que todos os objetos tem graus de interesse semelhante entre os observadores e, portanto, uma probabilidade próxima de serem acessados. Na prática, no entanto, algumas áreas e objetos atraem mais observadores que outras. Assim, não necessariamente o esquema baseado em densidade de objetos irá garantir uma distribuição uniforme de carga entre os servidores. Por este motivo, os autores propuseram um particionamente em regiões adaptativo, de acordo com a carga entre os diferentes servidores.

3.3.1 O esquema de particionamento adaptativo em regiões

No início do sistema, o ambiente virtual é divido sem levar em conta a carga dos servidores, pois não havia clientes conectados até então. Por isso, o particionamento inicial do ambiente é ou em regiões de tamanho igual, ou de número de densidade de objetos igual. Cada uma é atribuída a um servidor Si, que possui um indicador de carga wi. Quando wi indica que Si está sobrecarregado, a região Ri - gerenciada por Si - deve ser particionada. Um vizinho de Si com a menor carga será selecionado e a nova partição será entregue a ele.

Para determinar se uma região precisa ser particionada, cada servidor Si irá continuamente monitorar seu indicador wi, mantendo duas janelas de monitoração, chamadas janela de curta duração Ws(Si) e janela de longa duração Wl(Si). A primeira tem o objetivo de detectar repentinas chegadas de carga a Si, seja por congestionamento da rede, aumento repentino do interesse em Ri ou algum outro motivo. A janela de longa duração tem o objetivo de detectar sobrecarga contínua de Si.

Cada uma das janelas de monitoração avalia a carga do servidor baseado em dois fatores: comunicação e processamento. Se algum destes chegar a um nível acima de determinado limite, seja na janela de curta ou de longa duração, é disparado um particionamento da região gerenciada por aquele servidor. Tais limites podem ser pré fixados ou calculados dinamicamente, o que requeriria um algoritmo para cálculo dos mesmos.

Os autores sugerem um algoritmo para executar o particionamento de uma região Ri, de forma a reduzir a carga sobre o servidor Si dela encarregado:

1. Quando Si estiver sobrecarregado, ele entra em contato com o LoC, de modo a obter informações referentes à atual carga de seus vizinhos - um servidor vizinho de Si é aquele que gerencia uma área adjacente a Ri.

2. Após receber as informações a respeito de seus vizinhos, Si seleciona dentre les aquele que tiver a menor carga, que será chamado de St. Este receberá parte da carga de Si. Isto é feito particionando Ri e alocando a nova partição para St.

3. Si irá determinar a carga alvo, que é a quantidade de carga que deverá ser transferida para St.

4. O particionamento ocorre no nível das células. Cada servidor manterá um indicador de carga para cada célula. O particionamento de uma região Ri é conseguido

desalocando uma ou mais das células Cij da fronteira desta região, e agregando-as à região Rt, gerenciada por St: $Ri \leftarrow Ri - \{Cij\}$ e $Rt \leftarrow Rt \cup \{Cij\}$. Uma célula da fronteira de uma região Ri é aquela adjacente a uma célula pertencente a outra região, Rj. Por exemplo, na Figura 3.3, C4,3 é uma célula de fronteira de R4 e será agregada a R5.

5. Como muito provavelmente haverá mais de uma célula de fronteira, aquela com o indicador de carga mais próximo da carga alvo será alocada para St primeiro.

A razão para que a transferência de carga de um servidor seja apenas para seus vizinhos, ao invés de servidores arbitrários, é que quando o escopo de observador cruzar a fronteira entre dois servidores, ambos serão responsáveis por transferir modelos de objetos para o observador. Se um servidor descarregar a nova partição que criou em qualquer outro servidor, o observador poderá ter que se comunicar com um grande número de servidores, aumentando significativamente o sobrecusto de comunicação. Na figura D, se o servidor S4 descarregar as células C4,3, C4,6 e C4,9 em três servidores aleatórios, eles poderão ser diferentes. Quando um observador cujo escopo atravessar a fronteira entre as regiões R4,1 e R4,2, terá que se comunicar com vários servidores para que possa receber modelos de objetos dentro das três células. Por outro lado, se a transferência de carga é feita apenas para servidores vizinhos, todas essas três células serão descarregadas para o servidor S5. O observador terá apenas que se comunicar com dois servidores quando seu escopo atravessar a fronteira.

3.3.2 Alocação de sub-região

Quando um servidor está sobrecarregado, suas células de fronteira são transferidas para o mesmo servidor de destino até que a carga de trabalho do primeiro esteja abaixo de um determinado valor. Na figura D, se a transferência da célula C4,9 para R5 não pode reduzir a carga de trabalho da região R4,1 de forma que esteja abaixo do valor pré-determinado, outras células de fronteira, como C4,3 e C4,6, poderão também ser transferidas. Um problema em potencial, chamado de balanceamento de carga em cascata, pode ocorrer. Uma vez que um servidor vizinho é identificado como servidor alvo, ele recebe as células do servidor sobrecarregando, podendo ele mesmo ficar sobrecarregado, o que dispararia um novo processo de balanceamento.

Quando o efeito em cascata acontece enquanto o servidor Si descarrega uma nova sub-região a um servidor St, é possível que St identifique Si como servidor alvo e isso gere um efeito de troca contínua de carga. Desta forma, é necessário garantir que St não irá selecionar Si como servidor alvo imediatamente após Si ter lhe entregue uma partição.

3.4 Avaliação do trabalho

Uma forte limitação da proposta de (**?**) é que se considera o cenário de rede local para a rede de servidores. Tal premissa implica que haja pouca latência entre os servidores e a largura de banda disponível para intercomunicação dos mesmos seja a menor possível. Tais pressupostos não funcionam se é considerado um sistema distribuído geograficamente, em que os links entre as máquinas servidoras pode ser mais limitado.

No entanto, os autores propõem algumas soluções interessantes: uma delas é a utilização de diferentes resoluções para os dados a serem transmitidos, divididas em níveis de resolução. Isso é bastante útil para prover qualidade de serviço adaptativa aos recur-

sos disponíveis na rede: se há largura de banda disponível, envie o máximo de níveis de resolução possível. Caso contrário, envie apenas aqueles que for possível.

Outra idéia interessante é a do pre-fetch. Apesar de ser um conceito antigo e simples, a aplicação em jogos MMG pode ser bastante útil: clientes começam a comunicar-se, se provavelmente irão se ver num futuro próximo (o "provavelmente" pode ser calculado através de alguma heurística).

Por último, é deixado em aberto qual será o limite de subdivisão das regiões. A proposta de um novo método para calcular tal limite pode ser um trabalho interessante a desenvolver.

4 UM ESQUEMA DE DISTRIBUIÇÃO DE CARGA ESCALÁVEL PARA SISTEMAS DE AMBIENTE VIRTUAL DISTRIBUÍDO EM MULTI-SERVIDOR COM DISTRIBUIÇÃO DE USUÁRIOS ALTAMENTE ASSIMÉTRICA

Neste trabalho, (**?**) propõem um esquema dinâmico de distribuição de carga para os servidores de um sistema multi-servidor de ambiente virtual, levando em conta que os usuários podem estar distribuídos através deste mundo de maneira não uniforme. De acordo com o esquema proposto, um servidor sobrecarregado inicia a distribuição de carga selecionando um um conjunto de outros servidores para fazerem parte da distribuição, adptando-se dinamicamente ao nível de carga destes servidores selecionados. Após completar a seleção de servidores, o servidor que iniciou o processo reparte as regiões dedicadas ao grupo resultante de servidores utilizando um algoritmo de particionamento de grafo, de forma que os servidores envolvidos tenham carga final de trabalho semelhante. Após decidir-se quem fica com que trabalho, os servidores envolvidos migram suas cargas entre si de maneira par-a-par.

Essa distribuição de carga é necessária, pois assume-se - e, de fato, é a situação mais comum - que os jogadores não estejam distribuídos no mundo virtual uniformemente. Assim, algumas áreas do mapa terão mais jogadores que outras e os servidore associados a áreas mais populosas terão maior carga de trabalho. Servidores sobrecarregados não conseguiriam manter os jogadores a eles associados atualizados em tempo satisfatórios. Estes, então, experimentariam atraso em sua interação com o ambiente virtual, prejudicando significativamente a experiência de jogo.

Uma questão que foi considerada para resolver este problema foi a da escolha de utilização de informações locais (servidor e seus vizinhos) ou globais (todos os servidores estarão envolvidos). A primeira apresenta pouco overhead, mas pode não resolver o problema de maneira eficiente em poucos passos, já que servidores sobrecarregados tendem a estar adjacentes. A segunda alternativa é capaz de dividir a carga de trabalho o mais equilibradamente possível, mas sua complexidade cresce exponencialmente com o número de servidores envolvidos. A solução apontada pelos autores é de utilizar apenas um subconjunto de servidores, sendo que a quantidade de servidores envolvidos varia de acordo com a necessidade (se os vizinhos do iniciador estiverem também sobrecarregados, são selecionados mais servidores). Dessa forma, tem-se um pouco mais de informação do que a abordagem local, mas sem o problema inerente de complexidade da abordagem global.

4.1 Projeto

Nesta seção, será apresentado o esquema dinâmico de distribuição proposto por (**?**). Discute-se o que deve ser levado em conta para projetar tal esquema e, em seguida, é apresentada a proposta dos autores.

4.1.1 Considerações de projeto

O principal objetivo da distribuição da carga entre os servidores do multi-servidor é manter a qualidade de interação dos usuários em um nível aceitável, reduzindo a carga de servidores sobrecarregados e delegando-a a outros, menos carregados, mantendo o nível de carregamento uniforme, ainda que a distribuição dos usuários no ambiente virtual não o seja. Se um servidor lida com muitos usuários além de sua capacidade computacional, ele não pode enviar mensagens de atualização de estado em tempo satisfatório; como resultado, a interação dos usuários é degradada. Distribuição dinâmica de carga pode evitar tal problema transferindo algumas porções de uma região (células e usuários) de servidores sobrecarregados para servidores menos carregados. No entanto, a migração de células e usuários impõe um novo sobrecusto nos servidores porque a informação de estado atualizada dos usuários que migraram deve ser replicada dos velhos servidores para os novos; consequentemente, a própria distribuição dinâmica da carga também pode degradar a qualidade do jogo. Portanto, um esquema dinâmico de distribuição de carga deve reduzir o número de jogadores prejudicados por servidores sobrecarregados, assim como minimizar o número de migrações necessárias.

4.1.1.1 Escalabilidade

Um esquema dinâmico de distribuição de carga deve funcionar bem independentemente do teamanho do sistema. Em termos de sobrecusto, a abordagem local é adequada para sistemas de larga escala porque seu sobrecusto é limitado à comunicação do servidor com seus vizinhos, sem ser afetado pelo tamanho do sistema. No entanto, a abordagem local - apesar de funcionar bem com pequeno número de servidores - não funciona de maneira eficiente no caso de um desbalanceamento da carga de trabalho entre os servidores, que pode ser altamente assimétrica. A abordagem global apresenta grande eficácia independentemente do tamanho do sistema, pois todos os servidores no sistema pooperam para balancear suas cargas de trabalho. Porém, a abordagem global não é escalável, pois tem um sobrecusto que aumenta exponencialmente com o número de servidores no sistema multi-servidor. Logo, deve ser encontrada uma nova abordagem que funcione de maneira eficiente, com pequeno sobrecusto, independente do tamanho do sistema.

4.1.1.2 Adaptatividade

Para prover tal solução escalável, o esquema dinâmico de distribuição de carga deve ser executado adptando-se dinamicamente ao estado de sobrecarga dos servidores. Infelizmente, os esquemas existentes - local e global - não se adaptam ao estado de sobrecarga dos servidores. Na abordagem local, servidores sobrecarregados apenas consideram servidores vizinhos como seus parceiros para redistribuição da carga, de forma que não pode haver distribuição da sua carga excessiva de forma eficiente, se todos seus vizinhos estão sobrecarregados também. Por outro lado, a abordagem global reparte o ambiente virtual inteiro com informações de todos os servidores. Isto é desnecessário quando apenas um pequeno subconjunto dos servidores pode resolver de forma satisfatória o problema de desbalanceamento de carga do sistema. Sendo assim, um subconjunto de servi-

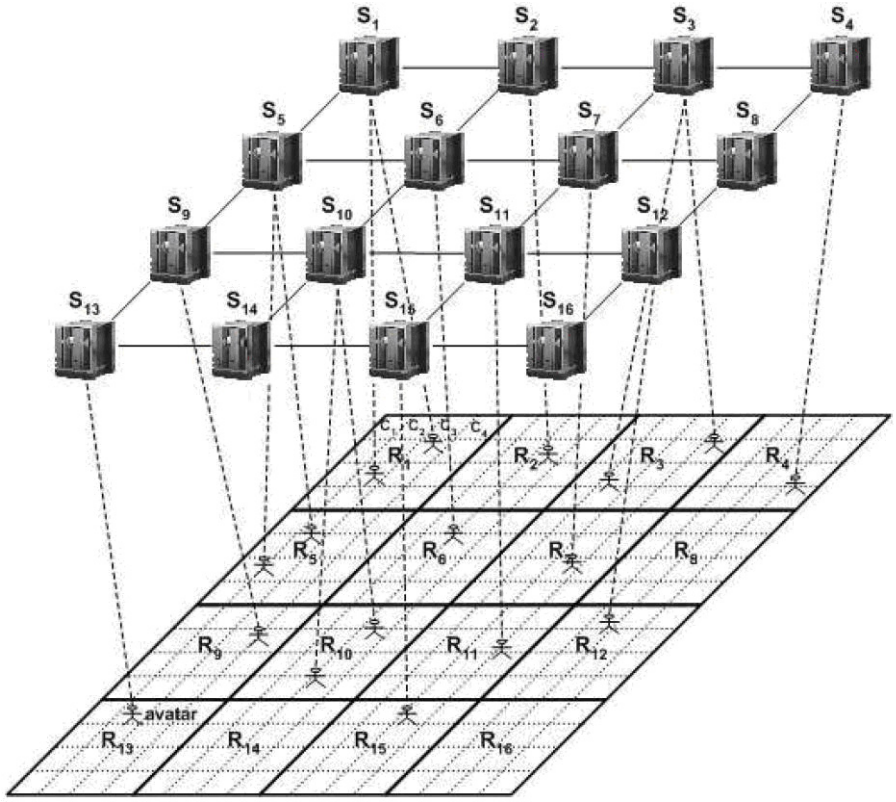


Figura 4.1: Um modelo multi-servidor para ambiente virtual distribuído

dores do sistema deve ser determinado, considerando o estado de sobrecarga dos outros servidores.

4.1.2 Modelo do sistema

É considerado um sistema multi-servidor de ambiente virtual distribuído, consistindo de Ns servidores: S1, S2, ..., SNs. O ambiente virtual que o sistema mantém é espacialmente subdividido em Nc células retangulares: C1, C2, ..., CNc, sendo que Ns $<<$ Nc. As células são agrupadas em Nr regiões e cada região é gerenciada por um servidor (i.e., Ns = Nr). Cada servidor mantém atualizadas informações de estado dos usuários e lida com as interações entre usuário na região a ele dedicada. Cada usuário envia e recebe atualizações de estado através do servidor que gerencia a região na qual ele está jogando. A Figura 4.1 ilustra o modelo de um sistema multi-servidor consistindo de 16 servidores. O ambiente virtual é dividido em 256 células, que são agrupadas em 16 regiões.

Duas células são ditas adjacentes (ou vizinhas) se elas compartilham uma fronteira em comum. Analogamente, duas regiões (e seus respectivos servidores designados) são ditos adjacentes se uma região contém uma célula que pertença à outra região. Um usuário representado por um avatar circula livremente de região a região e interage com outros usuários no ambiente virtual.

Define-se a carga de trabalho de uma célula, denotada por w(C), como o número de usuários presentes naquela célula. Assumindo que todos usuários atualizam seus estados na mesma frequência, a carga de processamento (computação e comunicação) que uma célula impõe a um servidor é proporcional ao número de usuários naquela célula. Similarmente, a carga de trabalho de uma região e seu servidor designado, denotada por w(R), é definida como a soma das cargas de trabalho das células que compõem aquela região.

Cada servidor periodicamente avalia sua carga de trabalho e troca informações de carga com os servidores vizinhos. Assume-se que estes servidores estejam conectados através de uma rede de alta velocidade. Dessa forma, o sobrecusto de trocar informações de sobrecarga entre vizinhos é limitada e considerada negligenciável, se comparada com outros custos da distribuição de carga.

A capacidade de processamento de um servidor é limitada; isto é, uma grande quantidade de carga, além da capacidade de um servidor aumenta o tempo de processamento das mensagens de atualizações de estado dos usuários. Define-se como capacidade de um servidor, representada por CP, o máximo número de usuários que o servidor pode suportar sem prejudicar a performance da interação entre os usuários. Considera-se que todos os servidores têm a mesma capacidade.

4.1.3 O esquema de distribuição de carga dinâmico proposto

Um servidor inicia a distribuição de carga quando sua carga excede sua capacidade. O servidor iniciador primeiro seleciona um conjunto de servidores para se envolver com a distribuição. Após completar a seleção dos servidores, o servidor que iniciou reparte as regiões que eram dedicadas a ele entre os servidores envolvidos, de forma que eles terão aproximadamente a mesma carga. Então, os servidores envolvidos migram suas celulas e usuarios entre si de maneira par-a-par, de acordo com o resultado do reparticionamento. Nas sessoes seguintes, será descrito em detalhe como isso é realizado.

4.1.3.1 Seleção adaptativa de servidor

Diferente dos esquemas global e local, um servidor iniciador no esquema proposto seleciona um conjunto de servidores para se envolver com a distribuição de carga dinamicamente se adptando ao status dos outros servidores. O servidor iniciaddor primeiro escolhe o menos carregado dentre seus vizinhos e pede que ele participe da distribuição. O vizinho escolhido rejeita o pedido se ele já participa ou executa outra distribuição de carga; caso contrário, ele participa da distribuição de carga respondendo ao servidor iniciador com a informação de carga de seus servidores vizinhos. Se o servidor vizinho que está partcipando não for capaz de absorver a carga de trabalho excedente do servidor iniciador, a seleção é executada novamente entre os servidores vizinhos de não apenas o servidor sobrecarregado, como também os vizinhos do vizinho escolhido na primeira fase. A seleção continua até que a carga de trabalho excedente do primeiro servidor possa ser absorvida - isto é, a carga de trabalho de todos os servidores selecionados torna-se menor que o limite. Pode-se definir este limite como 90% do CP, de forma a evitar o imediato reinício da distribuição de carga.

O procedimento para determinar o conjunto de servidores envolvidos é descrito da seguinte forma:

1. Um servidor iniciador é inserido a SELECIONADOS, que é o conjunto de servidores envolvidos na distribuição de carga. Os servidores vizinhos do iniciador são adicionados como CANDIDATOS, que é o conjunto formado pelos servidores candidadtos a seleção.

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

   (a) Se o servidor escolhido não está envolvido em outra distribuição de carga, ele responde ao servidor iniciador com a carga de trabalho de seus vizin-

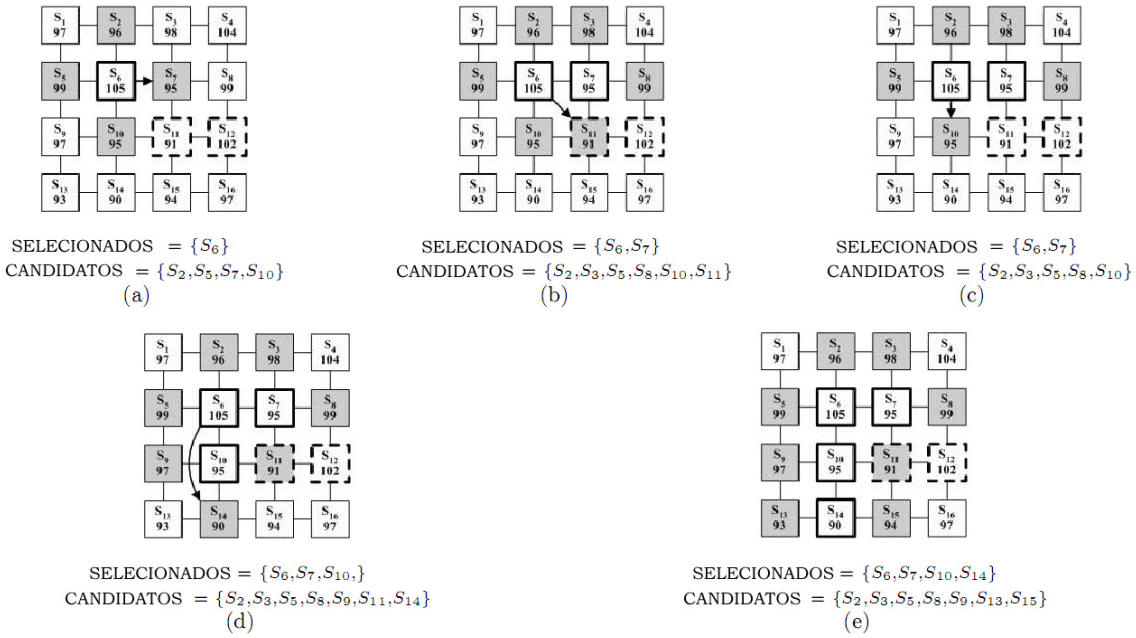


Figura 4.2: Um exemplo do algoritmo proposto para seleção de servidor

hos. Quando o servidor iniciador recebe esta resposta, o servidor escolhido é inserido em SELECIONADOS e seus vizinhos são inseridos em CANDIDATOS, se eles já não estiverem dentro de SELECIONADOS ou de CANDIDATOS.

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

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

Para exemplificar o funcionamento do algoritmo, pode-se observar a Figura 4.2. Todos os servidores têm a mesma capacidade de 100 - isto é, CP = 100. Primeiro, o servidor iniciador, S6, é inserido em SELECIONADOS e seus vizinhos (S2, S5, S7 e S10) são adicionados a CANDIDATOS (Figura 4.2(a)). Então, S7, que têm a menor carga de trabalho dentre os servidore em CANDIDATOS, é selecionado e convidado a participar da distribuição de carga. Quando S7 responde a S6 com a informação de carga de seus vizinhos (S3, S6, S8 e S11), S7 é inserido em SELECIONADOS e seus vizinhos, exceto S6, são adicionados a CANDIDATOS (Figura 4.2(b)). Agora, S11, que tem a menor carga de trabalho dentre os servidores em CANDIDATOS, é selecionado e convidado a participar da distribuição de carga. Porém, S11 rejeita o convite, pois já está envolvido em outra distribuição, iniciada por S12. Assim, S11 é removido de CANDIDATOS e S10 é selecionado porque tem agora a menor carga de trabalho dentre os servidores em CANDIDATOS (Figura 4.2(c)). Até que a carga de trabalho média dos servidores em SELECIONADOS se tornar menor que 0,9xCP = 90, o procedimento acima continua (Figura 4.2(d) e Figura 4.2(e)).

4.1.3.2 Particionamento de região

Uma vez que o servidor iniciador seleciona um conjunto de servidores para se envolverem na distribuição de carga, ele reparte as regiões a ele dedicadas com os servi-

dores envolvidos. Depois de reparticionar as regiões, todos os servidores envolvidos terão aproximadamente a mesma carga de trabalho. É utilizada a técnica de particionamento de grafos, que é extensivamente utilizada em computação paralela e de alta performance.

Um grafo G = (V, E, P) é construído utilizando informações detalhadas, isto é, a carga de trabalho de cada célula. Um vértice Vi pertencente a V representa a célula Ci. O peso do vértice Vi é ajustado como a carga de trabalho da célula Ci - isto é, w(Ci). Uma aresta Eij representa que duas células Ci e Cj são adjacentes. Os vértices são agrupados em partições $P = \{P1, P2, ...P|selected|\}$ que representam as regiões selecionadas. O grafo construído é reparticionado utilizando um algoritmo de particionamento de grafo, de forma que cada partição tem um subconjunto de vértices de peso total aproximadamente igual - ou seja, cada servidor terá aproximadamente a mesma carga de trabalho.

4.1.3.3 Migração de usuário e célula

Com o término do reparticionamento da região, o servidor iniciador dissemina o resultado do reparticionamento para os servidores envolvidos. O resultado inclue a informação de que células devem migrar para quais servidores. Os servidores que recebem o resultado executam a migração de células e usuários uns com os outros de maneira par-a-par.

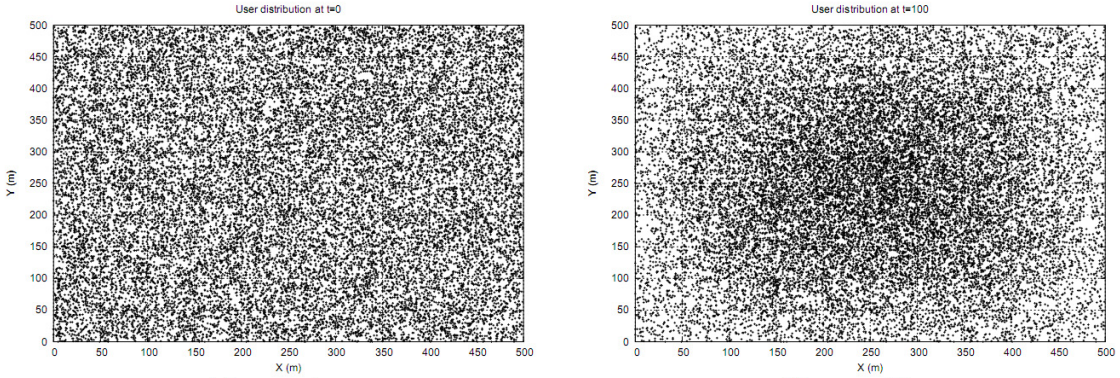
Primeiro um servidor replica a informação atualizada de usuários nas células migrantes para os servidores que as receberão. A informação que deve ser replicada varia de acordo com as características da apliação. Por exemplo, pode incluir apenas a localização dos usuários; ou pode conter informações mais detalhadas, como qual é o modelo 3D do avatar do usuário. Depois de completar a replicação, o servidor notifica os usuários nas células migrantes a respeito do novo servidor. Os usuários, então migram para o novo servidor; eles simplesmente deixam o servidor antigo e juntam-se ao novo.

4.2 Padrão de movimentação do usuário

Este artigo também propõe um modelo de movimentação de usuário, que baseia-se no *Random Waypoint Mobility Model*, que é extensivamente utilizado para avaliação de performance em redes sem fio ad-hoc. Usuários estão aleatoriamente distribuídos no ambiente virtual. Eles têm suas próprias localizações de destino traçadas, que são escolhidas aleatoriamente no mundo virtual. A cada passo t no tempo, cada usuário se move em direção ao seu destino através de uma linha reta com velocidade escolhida aleatoriamente entre 0 e max_speed. A velocidade máxima dos usuários, max_speed, é escolhida de forma que um usuário possa ir da esquerda para a direita do ambiente virtual em 100 passos de tempo. Por exemplo, se o tamanho do ambiente virtual é de $500x500(m^2)$, a velocidade máxima dos usuários é de 500/100 = 5 (m/passo de tempo). Quando um usuário chega à sua posição de destino, ele começa a se mover novamente de acordo com a mesma regra. Figura 4.3(a) mostra a distribuição inicial de usuários no ambiente virtual no instante t=0 e a Figura 4.3(b) mostra a distribuição de usuários no instante t=100. A distribuição de usuários mostrada nas figuras está de acordo com os resultados da distribuição espacial do *Random Waypoint Mobility Model*: os usuários tendem a se aglomerar na área central, ao invés da área das bordas.

4.3 Avaliação do trabalho

Tal como os trabalhos vistos anteriormente, (**?**) contém uma proposta de suporte a ambientes virtuais distribuídos (onde se encaixam jogos maciçamente multijogador),



(a) $t = 0$. (b) $t = 100$.

Figura 4.3: Distribuição dos usuários com o modelo de movimentação proposto

baseada na distribuição do servidor do ambiente, porém, considerando uma rede local e que não há atraso significativo na comunicação entre os nodos servidores.

De qualquer forma, este trabalho traz um algoritmo interessante para reparticionamento do ambiente virtual do jogo, de maneira dinâmica e adaptativa, buscando repartir a carga de trabalho com servidores menos sobrecarregados, seguindo uma heurística definida pelos autores.

Além disso, é proposto um modelo simples de movimentação dos usuários, baseado no *Random Waypoint Mobility Model*, que pode vir a ser útil numa futura simulação em algum trabalho futuro.

5 BALANCEAMENTO DE CARGA OTIMISTA EM UM AMBIENTE VIRTUAL DISTRIBUÍDO

Neste trabalho, (**?**) investigam a arquitetura de um ambiente unificado onde o mundo virtual online não é particionado com fronteiras rígidas, mas de acordo com um paradigma adaptativo. Como é difícil desenvolver um algoritmo de balanceamento de carga ótimo para um ambiente unificado, propõe-se um esquema otimista que converge rapidamente. O custo de migrações frequentes é reduzido seguindo um modelo de troca de dados push/push. Analisa-se também o custo de tempo computacional de tal sistema.

A grande questão deste trabalho é que o ambiente virtual é unificado, e não desconectado. Seria trivial criar um ambiente onde as regiões não tivessem nenhuma relação umas com as outras, e, ao invés de ter um grande ambiente com um grande suporte de rede, ter-se-ia vários pequenos ambientes, com soluções triviais para seu funcionamento.

Uma questão crucial a ser resolvida ao se criar um esquema para esse tipo de ambiente é o atraso da rede. Se houver um atraso na comunicação entre os jogadores, ou entre o servidor e o jogador, maior que o máximo tolerável, estes jogadores irão perceber movimentações inusitadas e não naturais de si mesmos e de seus parceiros de jogo. É necessário, pois, manter uma qualidade de serviço mínima para a simulação do ambiente virtual distribuído. Como serviços integrados e diferenciados não estão amplamente implementados no suporte atual da Internet, os autores empregam roteamento e políticas para manutenção de qualidade de serviço no nível da aplicação. Cada provedor de serviço de Internet pode prover um ou mais nodos gateway para os quais seus clientes conectar-se-iam para participar do ambiente virtual distribuído. Figura 5.1 ilustra dois gateways em dois provedores diferentes, que conectam-se ao mesmo núcleo servidor (ou cluster de servidores). Os gateways podem atuar como pontos de agregação de dados de usuários. Além de prover acesso rápido ao servidor, os gateways podem ser pontos de filtragem. Por fim, os gateways podem atuar como sincronizadores para garantir que todos os clientes receberão atualizações aproximadamente ao mesmo tempo, de forma a garantir a fairness.

Criar e gerenciar tais gateways é vantajoso apenas se o núcleo servidor puder suportar um número extremamente grande de clientes. Clientes ficarão relutantes em pagar taxas de assinatura se o servidor permanecer maior parte do tempo muito ocupado e ele não puder participar. Além disso, o núcleo servidor deve ser um cluster de máquinas que executam balanceamento inteligente de carga para compartilhar a carga do cliente de uma maneira escalável. É empregado um modelo de troca de dados push/push para reduzir os custos de migração de clientes entre os servidores.

Este paper fornece uma arquitetura e algoritmos para um ambiente virtual distribuído similar ao sistema ilustrado na Figura 5.1. O sistema permite que os clientes se movam com direções e velocidades selecionadas, assumindo-se que o percurso no eixo vertical

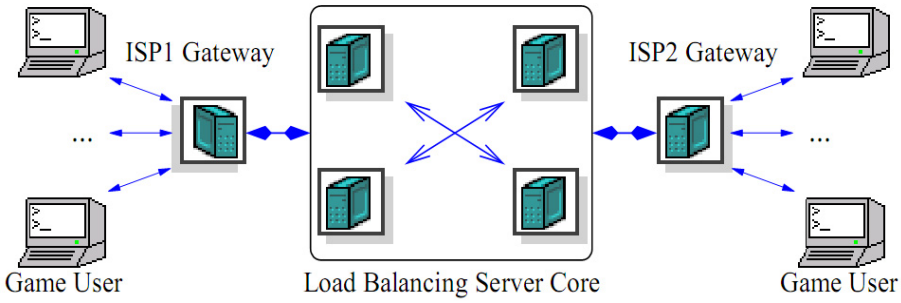


Figura 5.1: Arquitetura do sistema

não é relevante, significando que o ambiente pode ser decomposto em um mapa de apenas duas dimensões. Isto é adequado para a maior parte dos sistemas interativos no mercado atualmente. É projetada e avaliada a performance de um novo esquema adaptativo de balanceamento de carga para o núcleo servidor que explora o fato de que os clientes tendem a se aglomerar ao redor de pontos de interesse.

5.1 Visão geral do sistema

Clientes de ambientes virtuais distribuídos não se movem tipicamente de acordo com um caminhar aleatório: sempre há alguma estrutura para a posição geral dos clientes. Neste trabalho, (**?**) exploram o fato de que os clientes tendem a se aglomerar ao redor de pontos de interesse, que variam de um ambiente virtual para o outro. Os pontos de interesse podem ser dinâmicos e desconhecidos com antecedência, tornando difícil, senão impossível, particionar o espaço de jogo com antecedência. Assim sendo, um balanceador de carga altamente dinâmico é necessário. Um bom balanceador de carga deve satisfazer as seguintes propriedades:

1. Alcançar distribuição de carga uniforme;
2. Lidar de maneira eficiente com ambientes esparsos e
3. Permitir pontos de interesse dinâmicos.

Figura 5.2 ilustra um ambiente distribuído virtual com oito clientes. Os círculos pontilhados ao redor dos clientes representam o alcance de seu campo de visão. As setas representam os vetores diretores do cliente. Dados três servidores, é impossível construir regiões retangulares que dividam o processamento dos clientes igualmente entre os servidores. Foram escolhidos retângulos porque é rápido executar operaçãoes geométricas sobre eles, se comparados com outros polígonos. As regiões de cobertura dos retângulos podem mudar de tamanho com a movimentação dos clientes, mas enquanto os clientes permanecerem nas proximidades do ponto de interesse, não há necessidade de mudar as associações de cliente e servidor. Até que as coberturas dos servidores se interceptarem ou estiverem num campo de visão de um cliente, os servidores não precisam interagir de maneira extensiva para determinar se os clientes no seu domínio podem ver outros clientes.

Claramente, particionamento perfeito nem sempre pode ser mantido. Clientes tipicamente permanecem próximos a um ponto de interesse por algum tempo, mas então decidem mover-se para outro ponto, causando sobreposição. A Figura 5.3 ilustra um cenário

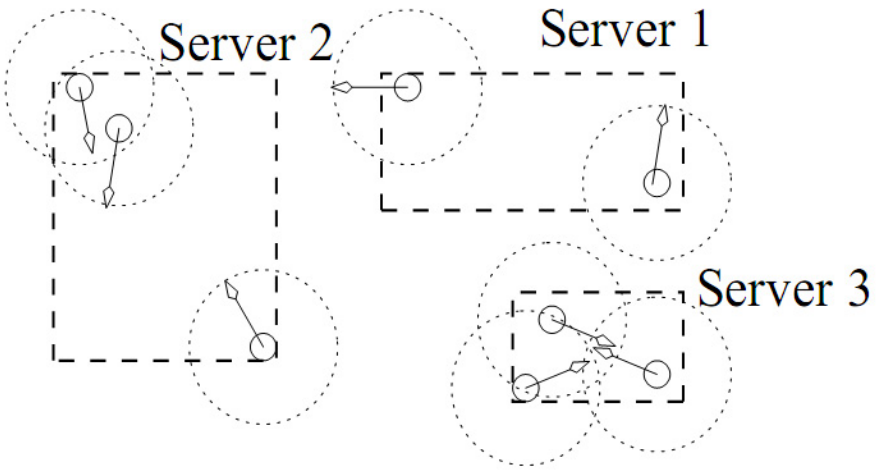


Figura 5.2: Particionamento ideal dos clientes

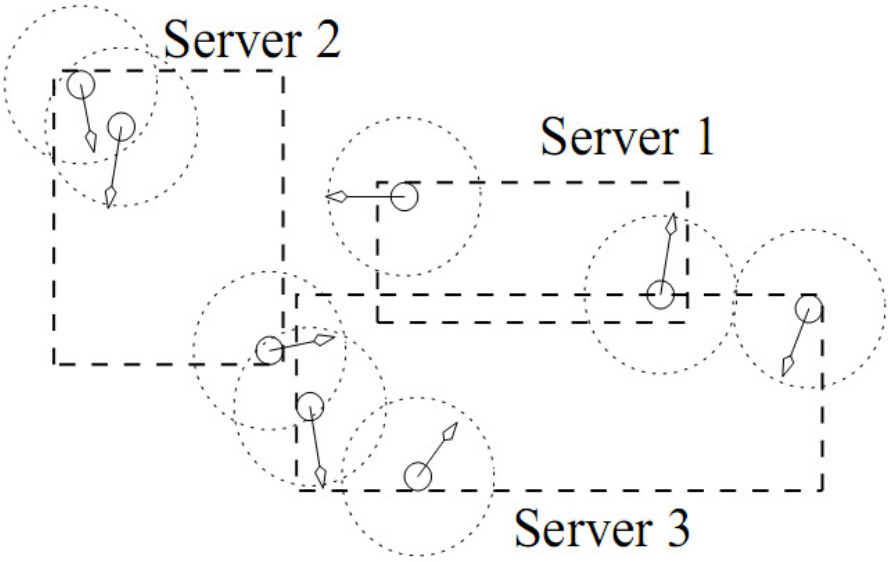


Figura 5.3: Sobreposição de cobertura devido ao uso de retângulos como fronteira

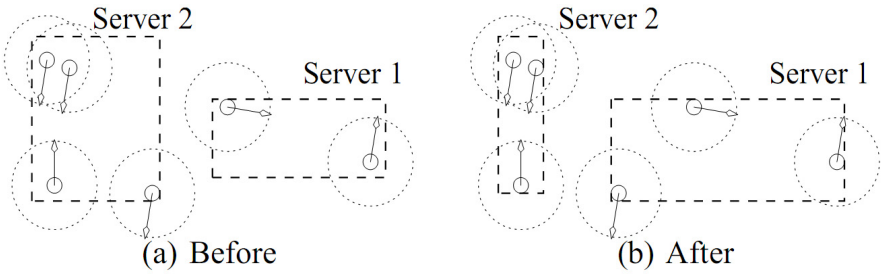


Figura 5.4: Migração de um cliente quando o número de clientes de um servidor excede o client_threshold de 3

em que um cliente no Server2 pode ver um cliente no Server3 e vice-versa. Isto pode ser facilmente checado observando quais campos de visão saem da área de cobertura. O caso em que o Server1 e Server3 se sobrepõem é mais problemático. Buscas devem ser conduzidas dentro das áreas de cobertura, e isto pode se tornar caro se o número de clientes é grande ou se sobreposições são frequentes. Este problema é melhor discutido mais adiante.

A seguir, será descrita por completo a arquitetura do algoritmo de balanceamento de carga, núcleo servidor e gateways.

5.1.1 Balanceador de carga

Indivualmente, os servidores no núcleo servidor executam ações locais gulosas para resolver problemas de sobreposição como os da Figura 5.3. O otimismo no sistema deriva do fato de que podemos rapidamente alcançar um bom estado global que nós prevemos ficará estável por vários ciclos. O balanceador de carga tem dois modos de operação. No primeiro modo, o balanceador tenta balancear a carga. O segundo modo é necessário para resolver qualquer questão de sobreposição.

Cada servidor usa um client_threshold para determinar o número de clientes que ele pretende servir. Se client_threshold é superado, o servidor tenta migrar parte de sua carga para um servidor próximo. Em certas situações, o servidor pode aceitar mais clientes que o especificado por client_threshold, mas ele não irá aceitar mais nenhum quando max_clients for atingido. Figura 5.4 demonstra a migração em ação. O número de clientes em Server2 excede o client_threshold, que tem o valor de 3, e o Server2 inicia a migração. O servidor seleciona outro servidor, que esteja mais próximo e que não tenha alcançado ainda sua capacidade máxima. Então, o cliente que estiver mais próximo da área gerenciada por aquele servidor será transferido. Colocou-se uma condição na transferência, para evitar sobreposição excessiva: a transferência não irá ocorrer se a nova área do servidor exceder max_area, que é configurável. Isto mantém o sistema verificado, de forma que um servidor não pode começar uma expansão rapida e maciçamente sobrepositora.

Migrações e movimentações de clientes vão, inevitavelmente, produzir casos onde as áreas de cobertura se sobrepõem. A Figura 5.5 ilustra um cenário onde um cliente do Server1 se move até a área de cobertura do Server2. Para resolver esta situação, cada servidor examina os outros servidores, que se sobrepõem com ele e tenta transferir todos os seus próprios clientes que estão na área sobreposta para o outro servidor, contanto que a capacidade máxima destes não tenha sido alcançada. Uma heurística que se aplicou neste caso é uma busca aleatória dos servidores que se intersectam. Isto é importante em situações onde a mesma região está na área de cobertura de mais que dois servidores.

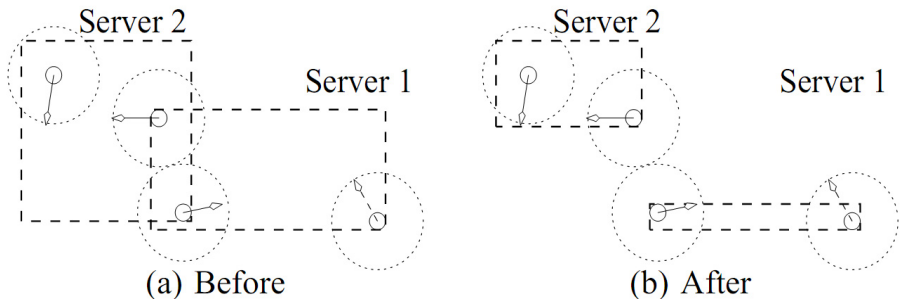


Figura 5.5: Resolução do problema de interseção de áreas de cobertura

Busca aleatória garante que várias combinações de soluções para intersecção são levadas em conta até que uma boa solução seja encontrada.

Para reduzir ainda mais a sobreposição de cobertura, introduz-se uma regra adicional. Se a área de cobertura de um servidor excede max_area, então o client_threshold é setado a 0, de forma que migrações irão ocorrer. Isto é necessário para evitar casos em que a área de cobertura do servidor X começa a crescer, causando uma crescente sobreposição, e os servidores que não eram vizinhos de X anteriormente passam a fazer transferências para ele, piorando ainda mais o problema.

Os parâmetros do algoritmo então permitem um tradeoff entre redução na variação de carga e sobreposições. Se o valor de max_clients estiver próximo de client_threshold, a variação da carga entre os servidores será pequena; no entanto, isto irá deixar pouco espaço para resolução do problema da sobreposição de áreas de cobertura, resultando em um maior número de sobreposições.

5.1.2 Núcleo servidor

O núcleo servidor serve como um ponto de agregação de todos os dados dos clientes, e emprega o algoritmo de balanceamento de carga descrito acima. Além disso, o núcleo servidor pode processar dados dos clientes, rodando uma versão limitada do motor do ambiente virtual distribuído com todas as rotinas gráficas desabilitadas. Executando este motor no núcleo servidor, trapaceadores podem ser detectados, pois os cálculos do cliente e do servidor irião apresentar discrepâncias. Em uma abordagem alternativa, o núcleo servidor pode executar o ambiente virtual distribuído e os clientes apenas processarem entrada e funções de exibição gráfica. Isto é útil em situações onde os clientes t em capacidade limitada de CPU, como clientes em dispositivos portáteis.

Para reduzir o custo da migração de clientes dentro do núcleo servidor, emprega-se um modo stateless, significando que servidores individuais não guardam informações dos clientes localmente. Isto elimina a necessidade de migração de complexas estruturas de dados dos clientes. Isso também elimina atraso na migração. A abordagem stateless proposta é adequada para casos em que uma porção significativa do estado do cliente irá mudar a cada atualização, e o estado pode ser comptactadamente representado (desde que os tipos e regras do ambiente virtual distribuído são conhecidas). O servidor mantém um pool de vagas de cliente para preencher com informação, à medida em que ocorrem atualizações e inserções de novos clientes. A vaga no pool elimina problemas de fragmentação de memória. É importante observar, no entanto, que se a taxa de atualização de estado for baixa, o modo stateless pode ser indesejável, já que o volume de tráfego será muito alto. Isto pode ser mitigado utilizando diferentes níveis de detalhe, de tal forma que o estado completo, até quando possível, não seja necessário.

O núcleo servidor comunica-se com gateways que executam roteamento no nível da aplicação. Conexões com estado são evitadas: pacotes UDP são roteados simplesmente com base no identificador único do servidor (ID). Roteamento eficiente no nível da aplicação pode ser conseguido por um software roteador modular ou um processador de rede. Há relatos de que aumentar o número de micro-motores nos processadores de rede irá permitir lidar com classificação específica da aplicação e roteamento na velocidade da linha.

Os gateways, primeiramente, fazem um push dos dados de atualização de estado dos usuários para os servidores. Cada cliente tem um ID de gateway e um ID de servidor associados a ele. Se uma migração de cliente ocorre, o núcleo servidor irá fazer um push dos novos dados em direção aos gateways refletindo a nova relação entre servidor e cliente. Isto forma a base dop modelo de troca de estados stateless push/push.

5.1.3 Gateways

Cada gateway atua como um ponto de conexão de alta velocidade ao núcleo servidor, a partir de um provedor de serviços de Internet em particular. Um gateway pode ser um servidor rápido ou uma coleção de processadores de rede que são otimizados para lidar especificamente com as questões do ambiente virtual distribuído. O gateway sincroniza-se com o núcleo servidor, garantindo que os clientes utilizam o mesmo tempo global. Isto é necessário para compesar os efeitos de atraso quando se utiliza vetores de predição e direção. O gateway pode também atrasar pacotes destinados a clientes para garantir que todos os clientes recebam os pacotes ao mesmo tempo. Isto previne casos em que alguns clientes podem ver eventos antes que outros clientes possam. Apesar de que os clientes podem aderir a ciclos iguais de tempo, o gateway não deve ser forçado a esperar que todos os dados dos clientes cheguem antes dele fazer o push para o núcleo servidor, de forma a evitar demoras devidas a clientes com performance pobre. Isto pode resultar em atualizações incompletas, mas enquanto os clientes não perderem seus tempos de envio agendados com muita frequencia, este problema não será perceptível.

Cada gateway recebe dados do núcleo servidor apenas a respeito dos clientes que pertencem a ele e que eles podem ver. Isto filtra de forma significativa informação irrelevante, reduzindo custos de processamento e de largura de banda. O gateway deve também filtrar os dados que são relevantes a cada clientes, já que seria ineficiente encaminhar o conjunto inteiro de dados para todos os seus clientes. A filtragem pode ser executada com base no campo de visão do cliente. Se o cliente não tem como ver aquele objeto, então não há necessidade de encaminhar-lhe aquela informação.

5.2 Análise de tempo de computação

Primeiro, é considerado o caso de um único servidor, então uma abordagem simples é utilizada pra múltiplos servidores, e então é apresentada uma abordagem otimizada para o problema.

5.2.1 Servidor único

No caso de um único servidor que utiliza o modelo de gateway descrito anteriormente, o servidor deve computar a física do ambiente virtual distribuído para cada cliente, e então computador quais clientes podem se ver uns aos outros de forma que ele possa reduzir a informação que os gateways receberam. Seja N o número de clientes. Seja P_i o tempo necessário para processar a física e a detecção de trapaça para o $Client_i$. Seja $C_{i,j}$ o tempo

necessário para computar se o $Client_i$ e o $Client_j$ podem se ver e se eles colidiram. Em um sistema simples de um único servidor executado comparações de par em par, o tempo total, T_s, pode ser computador como $T_s = \sum_{i=0}^{N} P_i + \sum_{i=0}^{N} \sum_{j=0}^{N} C_{i,j}$. Se assumir-se que $Pi = P$ para todo i, já que cada operação $C_{i,j}$ tem tempo constante, obtém-se $T_s = NP + N^2$.

5.2.2 Múltiplos servidores

Considere-se um sistema com M servidores e N clientes. Assuma-se que N_i é o número de clientes do $Server_i$, e (como acima) que P é o tempo requerido para processar a física e a detecção de trapaça para qualquer cliente. O custo das tarefas a serem executadas pelo servidor i, T_i, inclue:

1. N_iP para processar a física e a detecção de trapaça para seus próprios clientes.
2. N_i^2 para buscar interações locais de clientes.
3. $M-1$ para determinar os K retângulos adjacentes e sobrepostos (dentro do campo de visão do cliente) que são gerenciados por outros servidores.
4. $N_i \sum_{k=0}^{K} N_k$ para determinar os clientes que potencialmente podem interagir com seus clientes.
5. $N_i + K$ para determinar o cliente que é o mais próximo do servidor mais próximo se migração for requerida.
6. KN_i para determinar quais clientes atuais estão dentro da área de cobertura de outros servidores de forma que eles possam ser transferidos para lá, levando em conta que a capacidade máxima daquele servidor não tenha sido alcançada.

O tempo total do sistema é computador como $T_{total} = max(T_0, ..., T_M)$.

5.2.3 Custo otimizado

A análise simples acima mostra que se K é grande ou a distribuição de carga dos clientes (N_i é grande para algum i) é não-uniforme, o sistema multi-servidor irá ter um desempenho pior do que um sistema de servidor único. K é altamente dependente do número de retângulos soprepostos, porque se não havia sobreposição e todos os servidores cobriam espaçoes disjuntos, haveria no máximo 8 retângulos vizinhos a checar. Assim sendo, K depende da distribuição dos clientes e da performance do balanceador de carga.

O custo da busca de proximidade dos clientes pode ser reduzido para um pior caso de $O(N2N^{0.5})$ se uma quadtree for usado. Também leva $O(NlnN)$ para construir a árvore se inserção aproximadamente aleatória for assumida. Daí então, seria mais rápido construir uma árvore do zero e fazer buscas de proximadade do que fazer comparações de par em par. Modificando os passos na seção anterior, agora obtém-se a seguinte equação para o tempo de cada servidor: $T_i = N_iP + 2N_i^{1.5} + N_i lnN_i + M - 1 + (2N_i \sum_{k=0}^{K} N_k) \times (N_i \sum_{k=0}^{K} N_k)^{0.5} + (N_i \sum_{k=0}^{K} N_k) ln(N_i \sum_{k=0}^{K} N_k) + N_i + K + KN_i$.

A complexidad computacional é, então: $O(N_i^{1.5} + (N_i \sum_{k=0}^{K} N_k)^{1.5})$ se assumir-se tempo constante P. Quando for utilizada a mesma otimização (e novamente assumindo tempo constante P), a complexidade de um sistema com um único servidor é $O(N^{1.5})$. Assim, fica claro que é *essencial* minimizar K, e manter Ni tão próximo de N/M quanto possível. Para um valor grande de N, o custo principal é guiado pelo algoritmo de interação de clientes, e *não* pelo processamento da física.

5.3 Avaliação do trabalho

Este trabalho considera que os servidores, também formando um sistema distribuído, estarão instalados na forma de um núcleo servidor, dentro do qual não há atraso de rede ou gargalo de comunicação considerável. Assume-se que cada cliente está conectado a um provedor, que possui um gateway de acesso ao núcleo servidor, de forma a minimizar o custo de comunicação entre o cliente e o núcleo servidor do ambiente virtual distribuído.

Há também alguns pontos fracos neste trabalho, pois os autores consideram que o limite de clientes por servidor é uma constante, que poderia ser ajustada dinamicamente, com base na situação corrente daquele servidor. Mesmo assim, é interessante a idéia de considerar um sistema heterogêneo, onde cada servidor teria uma capacidade diferente. Ainda assim, esta capacidade poderia ser medida dinamicamente, baseada em parâmetros como atraso médio na comunicação com os clientes, ou largura de banda disponível para comunicar-se com eles.

Outro ponto questionável é o do algoritmo utilizado para reduzir a sobreposição das áreas de cobertura dos diferentes servidores. Para evitar que um servidor atinja uma área excessivamente grande, quando a sua área atingir ou exceder o valor max_area, o client_threshold é ajustado para 0, disparando o algoritmo de migração . E, mais uma vez, o valor max_area deveria ser calculado dinamicamente.

Não obstante, é interessante a análise de complexidade feita pelos autores, o que permite ter uma noção mais exata do tempo que é gato pelo algoritmo.

REFERÊNCIAS