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 3

LISTA DE FIGURAS 4

RESUMO 5

RESUMO 6

1 INTRODUÇÃO 7

2 UMA ARQUITETURA DISTRIBUíDA PARA MMORPG 8
2.1 Definições 8
2.1.1 Eventos 8
2.1.2 Ações 9
2.1.3 Área de interesse 9
2.2 Design 9
2.2.1 Mecanismos de travamento 10
2.2.2 Mecanismo de anúncio de eventos 10
2.2.3 Eventos próximos à fronteira do servidor 12
2.2.4 Abortando eventos 14
2.3 Escalabilidade 14
2.3.1 Hotspots 14
2.3.2 Escalabilidade 15
2.3.3 Tolerância a falhas 15
2.4 Avaliação do trabalho 15

REFERÊNCIAS 17

LISTA DE ABREVIATURAS E SIGLAS

MPI	Message-Passing Interface
SMP	Symmetric Multi-Processor

LISTA DE FIGURAS

Figura 2.1: O conceito de área de interesse com múltiplos servidores 9
Figura 2.2: Dois jogadores em servidores diferentes interagindo entre si 11

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

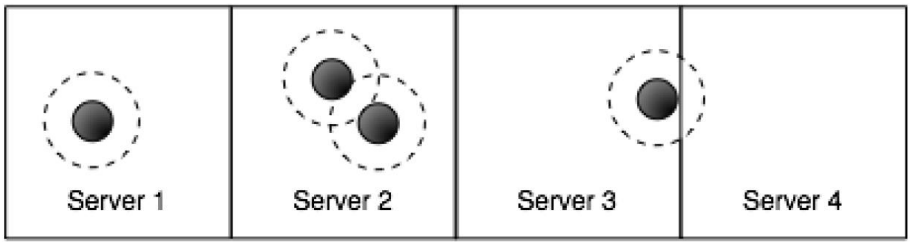

Figura 2.1: O conceito de área de interesse com múltiplos servidores

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

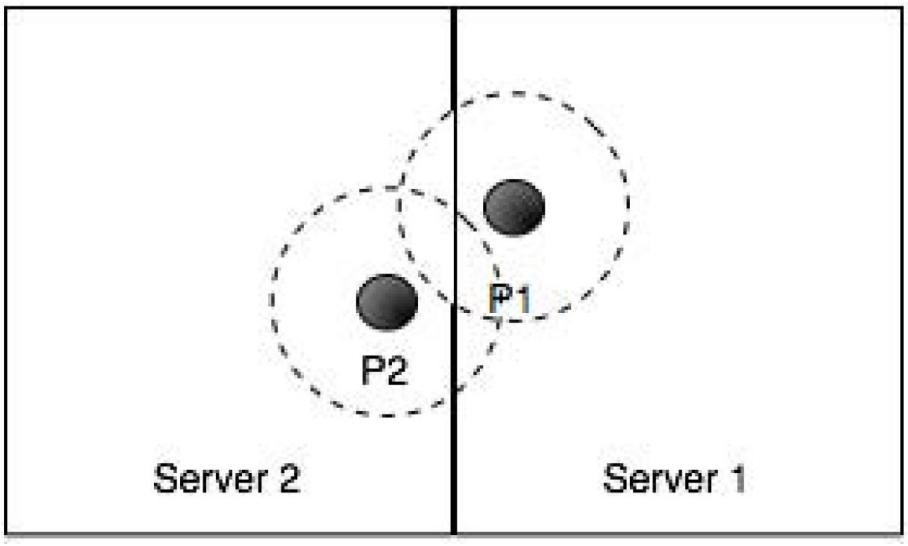

Figura 2.2: Dois jogadores em servidores diferentes interagindo entre si

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.

REFERÊNCIAS