Arquitectura Tolerante a Faltas para Jogos Distribuídos Multi-jogador

Hugo Ortiz
nº 29631
ortiz@netcabo.pt

João Nogueira
nº 29415
jvnogueira@mail.telepac.pt

Rui Guerra
nº 25268
i25268@alunos.di.fc.ul.pt

Grupo TFD011

21 de Outubro de 2005

Resumo

Com a evolução crescente das capacidades computacionais dos PC's e consolas e da velocidade das redes de computadores, a indústria de entretenimento dos video-jogos acompanha esta tendência, apostando nos jogos multi-jogador através da Internet.

A arquitectura cliente-servidor é tipicamente utilizada neste tipo de produtos devido à sua natureza se adequar perfeitamente ao modelo em questão: existe um árbitro (o servidor) que garante que as regras do jogo são cumpridas e gere os jogadores (clientes) e as suas acções no sistema. A adopção desta arquitectura tem, por definição, um problema fundamental: se o servidor falhar, o jogo acaba, ou seja, é ponto único de falha e estrangulamento do sistema. Propomos então, uma arquitectura para jogos multi-jogador cujo servidor é tolerante a faltas.

O jogo escolhido para demonstrar este conjunto de paradigmas foi o clássico da Atari, Pong [1] numa versão renovada distribuída e com 4 jogadores.

1 Introdução

A indústria dos video-jogos tende a acompanhar a evolução do poder de processamento dos computadores pessoais e consolas e das capacidades das redes de computadores. Os jogos multi-jogador através da Internet, oferecem aos utilizadores uma experiência mais atraente e interessante do que os comuns jogos que correm num único PC ou consola e serão, provavelemente, o próximo grande passo desta indústria.

A arquitectura cliente-servidor já usada em grande parte das aplicações distribuídas é a mais forte candidata à base dos jogos multi-jogador pois encaixa-se perfeitamente nos moldes da natureza do problema: o sistema necessita de um árbitro - uma entidade coordenadora que garanta o bom funcionamento do jogo (o árbitro evita a batota) e a gestão dos seus jogadores (o árbitro decide quem joga e quando) - e dos jogadores em si que ditam a evolução do jogo dentro das regras pré-definidas. O árbitro é o servidor e os jogadores são os clientes.

Esta arquitectura tem, contudo, um problema: o árbitro é vital para o jogo, pelo que se este falha (e.g deixa de funcionar, atrasa-se, toma decisões erradas) o jogo pode ficar comprometido. Tal como nos jogos reais (os não video-jogos), uma solução passa por ter mais do que um árbitro; apesar de um falhar, os outros podem tomar a decisão correcta. Este paradigma tem o nome, em sistemas distribuídos, de replicação de servidores.

A solução apresentada descreve uma arquitectura cliente-servidor, onde existe replicação de servidores para tolerar possíveis faltas. Para esta funcionar são introduzidas várias estratégias que mantêm o conjunto de servidores coerente (os árbitros não podem tomar decisões contraditórias para uma mesma acção) e que distribuem a carga do processamento entre eles (se existem vários árbitros, vamos repartir o trabalho por todos, para acelerar as decisões).

O jogo usado para demonstrar esta solução é uma adaptação do clássico Pong [1] da Atari. Esta nova versão distribuída e com suporte até 4 jogadores irá assentar na arquitectura descrita pelo que, se houver falhas no sistema dentro do modelo previsto, aquele continuará a funcionar com degradação mínima.

Este documento está organizado da seguinte forma: no ponto 2 é descrito o jogo Pong, em 3 é apresentado

o modelo de faltas assumido para o problema, em 4 é dada uma visão global da arquitectura que descreve a solução proposta; o ponto 5 apresenta o funcionamento do sistema no caso de não haver falhas de componentes e 6 o seu funcionamento caso contrário e respectivas adaptações; o ponto 7 descreve algumas optimizações introduzidas para mitigar problemas de desempenho; em 8 é explicada a decisão da escolha da entidade encarregue da evolução do jogo. 9 e 10 apresentam algumas conclusões e propostas de trabalho fututo.

2 O Pong

O jogo usado para demonstrar a solução apresentada para uma arquitectura de jogos multi-jogador fiável é uma versão adaptada do clássico da Atari, o Pong [1].

Esta nova versão tem natureza distribuída e suporta até quatro jogadores. Cada jogador é um cliente do sistema que se liga a um servidor que serve de árbitro e coordenador.

O jogo consiste num tabuleiro de jogo, uma bola em movimento e, por jogador, uma baliza e um paddle que este movimenta para desviar a bola da sua baliza. Cada vez que a bola entra na baliza de um jogador este perde um ponto. Ganha o jogo o jogador que chegar por último ao número máximo de pontos definido (tipicamente 10).

A posição da bola e pontuações dos jogadores são decididas pelo servidor. Os clientes recebem estas informações e interagem com o jogo movendo os seus paddles. Para evitar batota, os pedidos de movimento do paddle por parte de um cliente são relativos à posição anterior e não absolutos (pode mover x unidades de onde estava, não pode mover o paddle para a posição x), sendo o movimento pedido arbitrado pelo servidor.

A primeira operação de um novo cliente é a de se ligar ao servidor. Isto não faz com que começe imediatamente a jogar, apenas fica a ver o jogo actual. Para começar a jogar deve fazer, agora, um pedido de início de jogo. Se existir uma vaga no jogo actual, ou seja, se houver menos de quatro jogadores activos, é-lhe atribuído um paddle e começa a jogar. Se não houver uma vaga, não lhe é atribuído um paddle e fica em fila em modo de visualização.

Se um cliente está a jogar, pode sair do jogo e passar de novo ao modo de visualização do jogo actual. Neste modo, pode voltar a jogar ou terminar a ligação com o servidor.

3 Modelo de Faltas

Antes de descrever soluções para quisquer problemas é necessário haver uma ideia clara de quais são os problemas e de que estes efectivamente existem. Neste caso os problemas são as faltas que podem ocorrer no sistema clientes-servidores e as soluções são os métodos propostos para as tolerar de forma a que o jogo continue operacional.

Em particular, estamos interessados nas faltas de interacção entre os diversos componentes. Um componente no grau de granularidade que nos interessa, é um interveniente do sistema como um todo, ou seja, um cliente ou um servidor. Portanto, e seguindo o modelo falta-erro-falha recursivo proposto em [2], uma falha de um componente gera uma falta no sistema. Se esta última for tolerada, o sistema não chega a um estado erróneo, ou seja, mantém o seu comportamento correcto apesar de um componente ter falhado. Recorrendo ao modelo de classificação de faltas [3], consideramos os seguintes tipo: temporais, de omissão e de paragem, do grupo das faltas omissivas; sintácticas e semânticas do grupo das afirmativas.

As faltas que nos interessam são, como já referido, todas de interacção entre componentes. Contudo, as sintácticas, semânticas e temporais podem ser transformadas em omissivas somente ignorando a interacção respectiva (i.e. descartando a mensagem ou evento. O modelo de faltas a tolerar inclui, portanto, as omissivas e de paragem resultantes de falhas de interacção ao nível dos componentes do sistema.

4 Arquitectura da solução proposta

Existem dois componentes distintos e fundamentais no sistema: o cliente (operado pelo jogador) e o servidor (de operação automática). Cada um destes tem também uma divisão essencial à separação de abstracções que identifica o módulo do jogo, por um lado, e o da comunicação por outro.

Os módulos do jogo, quer do cliente, quer do servidor, são os responsáveis pela dinâmica das interacções, gerando os seus pedidos e respondendo a outros; são, portanto, o jogo em si (uma descrição mais detalhada do jogo encontra-se no ponto 5). Para que esta arquitectura não se aplique exclusivamente à variante do Pong usada e seja utilizável num leque mais vasto de produtos, é importante que haja algum nível de transparência,

para o jogo, na distribuição e replicação de servidores. Mas quanto é ”algum nível”? Não deve ser total, devido à própria natureza do problema. É importante que quer os clientes, quer os servidores estejam de alguma forma cientes do facto de existir distribuição: os clientes devem saber que estão ligados a um servidor e os servidores devem saber que têm clientes ligados. Os clientes do jogo não estão, contudo interessados em saber quantos servidores existem ou com o qual devem comunicar. Isso não acrescenta nada ao jogo em si. Pela mesma razão, os servidores de jogo também não estão interessados em quantos outros existem e quais os clientes ligados. O ideal seria, portanto, atingir o nível de transparência descrito acima, e ter uma camada de comunicação que resolvesse todos os outros problemas da distribuição. É exactamente esta a solução proposta neste artigo.

Os servidores estão organizados num grupo, suportado por uma camada que garante comunicação em grupo fiável e sincronia virtual de vistas. Cada cliente liga-se a um servidor, por uma ligação ponto-a-ponto fiável; para este, é virtualmente transparente a existência de um grupo de servidores.

O número de servidores no grupo necessários para tolerar f faltas que se enquadrem no modelo especificado é de $2.f + 1$.

Antes de descrever os módulos de comunicação do cliente e servidor, é importante introduzir a framework que os vai suportar: o appia [4].

4.1 O Appia

O Appia [4] é uma framework de suporte à comunicação usando pilhas de protocolos por camadas. Define uma interface comum a todas as camadas e é independende dos protocolos. Isto facilita a divisão destes em micro-protocolos, a interligação de protocolos diferentes e a comunicação entre as camadas. A um outro nível, os mesmo autores do Appia criaram, sobre essa framework, uma pilha de protocolos de comunicação em grupo fiável [5] que nos vai servir como base de trabalho para a criação das camadas de comunicação da nossa solução ao problema da fiablidade dos jogos multi-jogador distribuídos.

Mais informação sobre a arquitectura do appia e protocolos implementados sobre esta plataforma pode ser encontrada em [4], [5], [6] e [7].

4.2 Os módulos de jogo

Existem dois módulos de jogo: um do lado do cliente e outro do lado do servidor.

O do lado do cliente, representado na parte de cima da figura 1, encarrega-se de transmitir as interações do utilizador recebidas através da interface com este (*ClientUI*) ao respectivo módulo de comunicação (parte de baixo da figura 1). As interações possíveis são:

- Inicialização ou finalização do módulo de comunicação (*init* e *term*);
- Ligação ao servidor e terminação desta (*join* e *leave*);
- Começar e parar de jogar (*start* e *end*);
- Interacção com o jogo: movimentação do paddle - no caso de estar a jogar (*movePaddle*).

A inicialização e finalização do módulo de comunicação e estas interações são feitas através da interface *ClientConnector*, o que garante à arquitectura uma independência entre os dois módulos principais e, consequentemente, a transparência da forma como é feita a comunicação para o jogo em si como pretendido e descrito em 4. A transparência no sentido inverso não é necessária, pelo que as informações passadas pelo módulo de comunicação ao de jogo podem ser feitos directamente sem passarem pela interface *ClientConnector*.

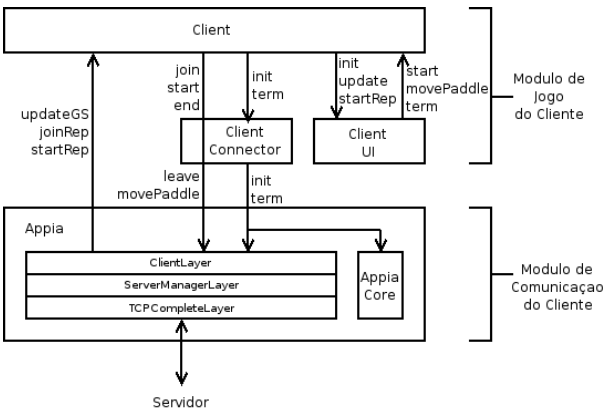

Figura 1: Diagrama da arquitectura do cliente

O módulo de jogo do servidor é passivo, ou seja, não toma a iniciativa de iniciar qualquer tipo de interacção, apenas responde a pedidos. Todos os pedidos são iniciados pelo módulo de comunicação, excepto os da sua

inicialização e finalização. A lista dos possíves é a seguinte:

- Inicialização ou finalização do módulo de comunicação (*init* e *term*);
- Pedido do seu estado do jogo actual (*requestGS*)
- Actualização do seu estado do jogo com uma informação mais recente (*updateGS*);
- Movimentação do paddle de um cliente (*movePaddle*).

Este módulo está representado na parte de cima da figura 2. A inicialização e finalização do módulo de comunicação, tal como no cliente, são feitas através de uma interface, neste caso *ServerConnector*, e os pedidos e informações passados ao módulo de jogo são também feitos directamente, pelas mesmas razões.

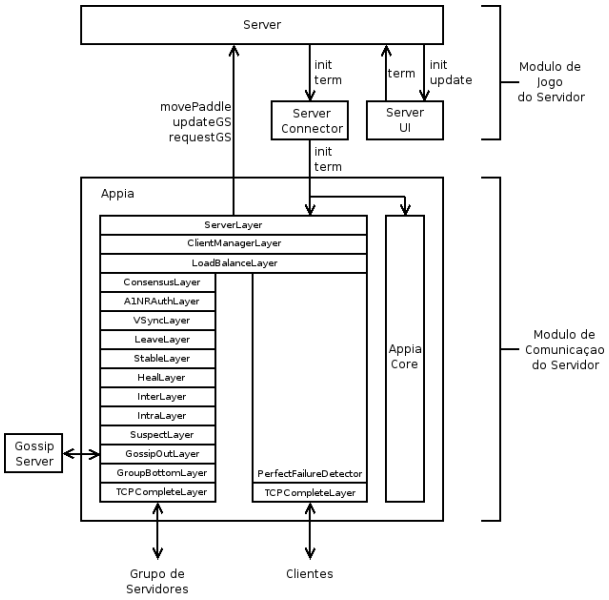

Figura 2: Diagrama da arquitectura do servidor

4.3 Os módulos de comunicação

Existem dois módulos de comunicação: um do lado do cliente e outro do lado do servidor. Este ponto faz uma descrição das funcões de cada um no sistema.

O módulo de comunicação do cliente tem como objectivos enviar os pedidos feitos pelo módulo do jogo (as interações do jogo em si) ao servidor e informar este módulo das actualizações do estado do jogo recebidas. O micro-protocolo que trata destas operações tem o nome de *ClientLayer* na pilha e opera sobre uma camada de comunicação ponto-a-ponto fiável. Este módulo faz também os pedidos de ligação ao servidor, de início de um novo jogo por parte do cliente e saída deste, através do micro-protocolo *ServerConnectLayer*. Como suporte ao balanceamento de carga feito pelos servidores, a operação de ligação inicial é dividida em duas fases. Na primeira fase, a camada *ServerManagerLayer* indica ao servidor que se quer juntar ao jogo. Na resposta, recebe o endereço de um servidor ao qual se deve efectivamente ligar. A segunda fase consiste na ligação a este último. Este módulo está esquematizado na parte de baixo da figura 1. Tem apenas um canal de comunicação ponto-a-ponto fiável para ligação ao servidor.

O módulo de comunicação do servidor tem quatro tarefas essenciais: a gestão dos clientes e balanceamento da carga em cada servidor do grupo, a manutenção da coerência do estado global do jogo entre os elementos do grupo, a actualização periódica do estado do jogo dos clientes e a gestão dos pedidos dos clientes. Este módulo está esquematizado na parte de baixo da figura 2.

A primeira tarefa consiste em indicar, aquando da recepção de um pedido de ligação da parte de um cliente, qual o servidor (de dentro do grupo) ao qual este se há de ligar. Esta decisão é tomada por acordo entre os membros do grupo e baseada num algoritmo de balanceamento de carga. O micro-protocolo encarregado desta decisão tem o nome de *LoadBalanceLayer* na pilha representada na figura 2 e opera com base noutros que resolvem o problema do consenso (*ConsensusLayer*) num grupo com comunicação fiável [8] (*VSyncLayer* até *GroupBottomLayer*).

A segunda tarefa mantém a coerência entre os estados de todos os servidores utilizando a abstracção de memória partilhada sobre os paradigmas de registo atómico (1,N) [9], comunicação em grupo fiável [8] para difusão de mensagens e consenso [10] para resolução de problemas de concorrência: o estado actual do jogo está guardado na memória partilhada acessível por todos os servidores do grupo. A memória partilhada é tornada possível pelo micro-protocolo *A1NRAuthLayer* que implementa um registo atómico (1,N) sobre um grupo de comunicação fiável [8] (*VSyncLayer* até *GroupBottomLayer*). Como já referido, o consenso entre elementos do grupo é oferecido pela camada *ConsensusLayer*.

A terceira tarefa consiste em enviar, periodicamente, a todos os clientes, uma cópia do actual estado do jogo.

A última tarefa consiste em fazer alguma validação dos pedidos dos clientes e enviá-los ao módulo de jogo para processamento e sua introdução no estado do jogo.

Aquelas tarefas são executadas ao nível dos micro-protocolos do topo da pilha *ServerLayer*, utilizando os serviços das camadas inferiores já referidas em cada um dos casos.

Este módulo de comunicação do servidor é composto por dois canais: um para ligação com os clientes que oferece serviços de comunicação ponto-a-ponto fiável e detecção de falhas e outro para ligação aos servidores que oferece serviços de comunicação em grupo fiável, sincronia virtual de vistas, registos atómicos com um escritor e múltiplos leitores e consenso. Os três micro-protocolos do topo da pilha (*ServerLayer*, *ClientManagerLayer* e *LoadBalanceLayer*) são comuns aos dois canais pois interagem tanto com os clientes como com o grupo de servidores.

4.4 Escolha do servidor para um cliente

Quando um novo cliente se pretende ligar ao sistema, envia um pedido deste tipo a um servidor do grupo que conhece. O grupo de servidores tem, agora, que decidir qual será o mais indicado para o cliente se ligar directamente. Isto é a primeira tarefa descrita em 4.3.

Para resolver o problema, cada servidor do grupo calcula, para si, um ranking de carga baseado num algoritmo de balanceamento de carga e nalguns parâmetros conhecidos (latências, número de clientes ligados actualmente). O grupo executa, agora, um protocolo de consenso para chegar a acordo sobre qual dos servidores deve aceitar a ligação do cliente, ou seja, o que tem o ranking de carga mais baixo. O servidor ao qual o cliente em questão se ligou originalmente responde agora ao cliente com o endereço do elemento do grupo ao qual aquele se deve efectivamente ligar.

4.5 Estado global do jogo coerente no grupo

É necessário manter coerente a informação do estado do jogo mantida por cada servidor do grupo. Esta é a segunda tarefa descrita em 4.3.

Para resolver o problema, a camada *A1NRAuth* oferece a abstracção memória partilhada através de registos atómicos com um escritor e múltiplos leitores [9]. Cada elemento do estado do jogo é escrito num determinado local desta memória partilhada. Para evitar corridas e problemas de concorrência, cada elemento do estado do jogo tem associado um servidor que fica encarregue de o actualizar: nenhum dos outros pode escrever nesta zona de memória. A um servidor encarregue de escrever uma determinada zona da memória partilhada é dado o nome de autoritário em relação a essa informação.

Por exemplo, no Pong, a posição de um paddle só pode ser alterada pelo servidor que tem o cliente que joga com aquele ligado. No caso da bola, a sua animação é feita (e escrita na memória partilhada) pelo servidor que tem o cliente jogador do paddle para onde aquela se dirige. A razão pela qual esta última opção foi escolhida, ao invés de haver um único servidor encarregue de mover a bola é explicada, de forma mais detalhada, em 8.

O módulo de jogo do servidor é completamente passivo em relação ao módulo de comunicação, ou seja, não toma a iniciativa de enviar qualquer tipo de informação que não seja pedida explicitamente. Esta opção foi tomada para aumentar a transparência do jogo em si em relação à comunicação e estratégias de tolerância a faltas. Deste modo, está a cargo do micro-protocolo *ServerLayer* de cada servidor com autoridade para escrever em cada zona da memória partilhada, inquirir periodicamente o módulo de jogo respectivo sobre o estado dos elementos a actualizar naquele registo. Esta informação é, depois, consultada por todo o grupo.

4.6 Actualização periódica do estado do jogo dos clientes

A terceira tarefa referida em 4.3 é a de informar periodicamente cada cliente do estado do jogo. A resposta autoritária (por oposição às informações não autoritárias dos elementos do estado do jogo resultantes das previsões feitas por cada servidor - mais informação sobre respostas autoritárias em 4.5 e sobre previsões em 7) deste estado é guardada na memória partilhada. O servidor responsável por transmitir esta informação a um determinado cliente é o que tem a ligação ponto-a-ponto directa com este. Existe, portanto, em cada servidor que tem um ou mais clientes ligados, um temporizador que indica periodicamente à camada *ServerLayer* do seu módulo de comunicação que este deve enviar o estado ao cliente. Esta camada faz então a leitura do

estado da memória partilhada e envia uma mensagem a cada cliente que tem ligado com esta informação.

4.7 Envio dos pedidos dos clientes ao módulo de jogo

Como quarta tarefa (também referida em 4.3), o módulo de comunicação deve fazer chegar os pedidos de cada cliente ao módulo de jogo do servidor encarregado de actualizar a respectiva zona da memória partilhada. Para o fazer, a camada *ServerLayer* recebe e interpreta os pedidos de interacção dos clientes (e.g. no Pong, mover o paddle de x unidades numa determinada direcção) e faz este pedido ao módulo de jogo. Este, por sua vez, faz as alterações necessárias no estado do jogo (e.g no Pong, posição do paddle e, possivelmente, a pontuação).

5 Descrição geral do funcionamento do sistema

Este ponto esquematiza e clarifica o funcionamento de todo o sistema proposto em caso de operação correcta (sem faltas). As situações possíveis estão descritas abaixo:

5.1 Ligação de um novo cliente ao sistema

Um novo cliente, para se ligar ao sistema, deve fazer um pedido de ligação a um servidor que conheça. Este servidor, recebendo um pedido deste tipo, vai iniciar um algoritmo de consenso entre os elementos do grupo para decidir a que servidor o cliente se deve efectivamente ligar. O acordo é baseado nos rankings de carga de cada servidor obtidos a partir de um algoritmo de balanceamento de carga. O resultado do consenso é comunicado ao cliente que se liga ao servidor decidido. O cliente fica em estado de visualização do jogo corrente. Não pode intervir. Todo este processo é executado nos módulos de comunicação do cliente e servidor: para o módulo de jogo do cliente só existe a primitiva para ligar e respectiva resposta, e para o módulo de jogo do servidor é transparente se tem clientes ligados ou não.

5.2 Ligação de um novo cliente ao jogo

Uma vez em modo de visualização, um cliente pode tornar-se um participante activo e começar a jogar, caso haja menos de quatro jogadores activos no momento. Para tal, faz um pedido desse tipo ao servidor. O servidor confirma que há menos de quatro jogadores activos no momento e vai iniciar um novo algoritmo de consenso entre o grupo. Caso este seja o único cliente a querer jogar no momento, o pedido é aceite e o cliente é informado de que começou a jogar. Caso haja mais clientes a querer jogar que posições disponíveis (e.g 3 candidatos para a posição de quarto paddle), o grupo de servidores vai chegar a acordo de quais daqueles vão ocupar as posições activas (no Pong, os paddles livres) restantes. Todo este processo é executado nos módulos de comunicação do cliente e servidor: para o módulo de jogo do cliente só existe a primitiva para começar a jogar e respectiva resposta e para o módulo de jogo do servidor é transparente se existem jogadores ou não - apenas trata pedidos de interacção (mover o paddle, no Pong).

5.3 Pedidos de um cliente activo

Uma vez em modo activo, o cliente faz pedidos de interacção (no Pong, mover o seu paddle) ao servidor. Este valida os pedidos e redirecciona-os para o módulo de jogo. A validação passa por verificar se vêm de um cliente que o pode fazer e se o seu conteúdo é correcto. Se a validação falhar, o pedido é ignorado.

5.4 Actualização do estado dos clientes

Cada servidor tem um temporizador que avisa o módulo de comunicação quando deve actualizar o estado dos seus clientes. Aquando desta indicação, este faz um pedido do estado do módulo de jogo e envia esta informação, numa mensagem, a cada um dos seus clientes.

5.5 Actualização do estado dos servidores

Cada servidor mantém uma cópia local do estado do jogo que vai actualizando em função dos pedidos dos seus clientes. Os servidores que não têm clientes directamente ligados não alteram, por si mesmos, o seu estado local. Têm também um temporizador que avisa o módulo de comunicação quando deve sincronizar a sua cópia local do estado do jogo com a cópia residente na memória partilhada. O facto de existir uma cópia local para além da da memória partilhada deve-se a motivos de optimização e aceleração do processo descritos em 7. A sincronização é feita da seguinte forma: se o servidor

tem autoridade para escrever num ou mais campos da memória partilhada (a noção de autoridade de escrita na memória partilhada, bem como a razão pela qual esta existe são descritas em 4.5), vai actualizá-los com a informação respectiva do seu estado local; os campos para os quais não é autoritário, vai copiar a informação da memória partilhada para o seu estado local. A sua memória está agora coerente com a do grupo.

6 Faltas do Pong e adaptação do sistema

Este ponto esquematiza e clarifica o funcionamento de todo o sistema proposto na ocorrência de cada tipo de falta previsto no modelo descrito em 3.

6.1 Faltas de omissão

O facto de basear toda a arquitectura em primitivas de comunicação fiável (quer ponto-a-ponto entre cliente e servidor, quer no grupo de servidores), faz com que as faltas de omissão sejam toleradas por estes protocolos de nível mais baixo ou, no pior dos cenários, dêem origem em falhas de paragem dos componentes (cliente ou servidor), ou seja, faltas de paragem do sistema segundo o modelo de [2].

6.2 Faltas de paragem

Uma falta de paragem do sistema indica uma falha de paragem de cliente ou servidor. Estas são sempre detectadas do lado do servidor.

6.3 Faltas de paragem dos clientes

Os servidores que têm clientes ligados tem um detector de falhas nesse canal de comunicação (figura 2). Sempre que o detector de falhas dá a indicação que um cliente falhou, o módulo de comunicação encarrega-se de o remover do jogo activo e, caso existam outros em fila de espera, dar a indicação ao próximo que pode começar a jogar.

6.4 Faltas de paragem dos servidores

Sempre que um servidor falha, a pilha de protocolos de comunicação em grupo identifica esta anomalia e entrega uma nova vista. Se o servidor que falhou não tinha clientes ligados, não há necessidade de fazer qualquer alteração. Caso contrário, o grupo vai correr o algoritmo de consenso baseado nas cargas (semelhante ao executado quando um novo cliente faz um pedido de ligação) e vai indicar, a cada cliente nesta situação, um novo servidor onde se deve ligar. Uma vez ligado ao novo servidor, o jogo fica reestabelecido para esse cliente e recomeça a operação sem que o utilizador se aperceba da falha do servidor. Isto é possível pois o estado dos clientes antes da falha do servidor é mantido durante este processo (se o cliente estava activo no jogo não perde o seu lugar passando à lista de espera).

7 Optimizações

A actualização da memória partilhada implica a troca de várias mensagens entre os elementos do grupo de servidores. Isto demora tempo não desprezível que pode comprometer a fluidez do jogo. A sua actualização com demasiada frequência pode também inundar a rede de mensagens.

Para optimizar esta solução e mitigar o problema, optou-se por manter, em cada servidor, uma cópia local do estado do jogo que funciona como uma cache. Deste modo, a cache só é actualizada periodicamente, minimizando o número de mensagens trocadas. Por outro lado, cada servidor pode fazer uma previsão da evolução do jogo (no caso do Pong, da posição da bola), aumentando a fluidez do mesmo. A sincronização das caches é feita por meio de autoridades sobre cada elemento do estado do jogo como já descrito no ponto 5.5.

8 Quem que decide a evolução do jogo

Já ficou claro quais são os servidores autoritários em relação aos campos de memória partilhada referentes ao estado de cada jogador (no caso do Pong, a posição do paddle e a sua pontuação). Quanto ao estado que é evoluído somente pelos servidores (no caso do Pong, a posição da bola), é necessário haver um critério de escolha do servidor que irá ser o autoritário em relação a esta informação (uma explicação da autoridade dos servidores sobre zonas da memória partilhada está em 4.5).

Um hipótese seria eleger apenas um (e.g. o coordenador de grupo ou aleatoriamente). Como foi referido em 7, pode haver, temporariamente, uma diferença entre os valores do estado de jogo de cada servidor e os

da memória partilhada. Se um servidor controla a bola e o outro controla o paddle de um jogador e a diferença entre estes valores nos seus estados locais for suficiente, pode haver uma incoerência na decisão se a bola passa pelo paddle ou se esta bate no paddle. Para resolver este problema, garantimos que, neste caso, o servidor que anima a bola é sempre o mesmo que anima o paddle, ou seja, quem anima a bola num determinado instante é sempre o servidor que tem o cliente jogador ligado que movimenta o paddle para onde aquela se dirige. Sempre que a bola muda de direção, o servidor que anima a bola naquele momento continua a animá-la ou, caso ela se dirija para um paddle que não controla, informa o novo servidor de que este vai passar a controlá-la.

Se o servidor que anima a bola falha, a decisão do novo animador é tomada de acordo com a decisão dos novos servidores para os clientes que estavam ligados ao que falhou (ver ponto 6.4), segundo o critério descrito acima.

9 Conclusões

A maior parte dos jogos multi-jogador têm uma enorme vantagem na utilização da bem conhecida arquitectura cliente-servidor. O principal problema de uma arquitectura destas é a possível falha do servidor, o que pode vir a comprometer o jogo. É, portanto, apresentada uma solução para este tipo de jogos em rede que visa resolver este problema através da replicação dos servidores. O servidor único, ponto de falha e estrangulamento do sistema, é substituído por um grupo de servidores que comunica e trabalha em conjunto de forma a, em caso de falha de algum componente do sistema, ajustar o seu funcionamento para que este continue a operar correctamente. Sendo de extrema importância manter algum nível de transparência do modelo de comunicação e tolerância a faltas para o jogo em si, esta encontra-se num módulo à parte, facilitando a futura adaptação a diferentes jogos ou outras aplicações distribuídas da mesma natureza.

10 Trabalho Futuro

Esta solução apenas visa tolerar faltas não maliciosas e não intencionais. É, contudo, um facto bem conhecido que em ambientes de redes públicas como a Internet, e mesmo em redes mais fechadas existe sempre a ameaça das faltas maliciosas que podem dar origem a intrusões nos sistemas. Este tipo não é tolerado pela arquitectura actual, pelo que o próximo passo será a introdução de mecanismos de segurança no sistema. Alguns exemplos são a assinatura da aplicação do cliente para gerar uma base de confiança, a utilização de canais de comunicação seguros quer entre clientes e servidores como no próprio grupo de servidores e mecanismos de autenticação para estabelecimento de confiança mútua entre elementos deste grupo. Outras formas de protecção externas à arquitectura também podem ser utilizadas, tais como anteparas de segurança e detectores de intrusões para minimizar o risco e a quantidade de ataques.

Outra evolução desta arquitectura proposta seria transformá-la numa framework com uma boa API que possa servir de produto de middleware sobre o qual podem contruídas as camadas de comunicação de futuros jogos multi-jogador em rede.

Referências

[1] Atari Pong Story [*http://www.pong-story.com/atpong2.htm*]

[2] Veríssimo, P., Rodrigues, L. (2001) *Distributed Systems for System Arquitects, sec. 6.1.1*

[3] Veríssimo, P., Rodrigues, L. (2001) *Distributed Systems for System Arquitects, sec. 6.1.4*

[4] Página oficial do Appia [*http://appia.di.fc.ul.pt*]

[5] Pinto, A. (2005) *Appia Group Communication*

[6] Rodrigues, L., Miranda, H., Pinto, A., Carvalho, N. (2005) *Appia Protocol Development Manual*

[7] Rodrigues, L., Carvalho, N. (2005) *Implementing Reliable Broadcast Protocols in Appia: A brief tutorial*

[8] Guerraoui, R., Rodrigues, L. (2005-draft) *Introduction to Reliable Distributed Programming, cap. 3*

[9] Guerraoui, R., Rodrigues, L. (2005-draft) *Introduction to Reliable Distributed Programming, sec. 4.3*

[10] Guerraoui, R., Rodrigues, L. (2005-draft) *Introduction to Reliable Distributed Programming, cap. 5*