Pong a quatro jogadores com servidores tolerantes a faltas

Carlos Segura
i29258@alunos.di.fc.ul.pt

João Pereira
i29259@alunos.di.fc.ul.pt

Tiago Honorato
i29157@alunos.di.fc.ul.pt

Grupo 03

Abstract

O Pong é um jogo multi-utilizadores (4 jogadores), em que todos os jogadores comunicam com um único servidor. Logo, se o servidor falhar, origina uma falha global do sistema e os respectivos jogadores não poderão continuar o seu jogo. Este artigo apresenta uma solução para este problema. A solução passa por replicar o servidor, onde cada réplica guarda o estado do jogo e comunica com as restantes usando comunicação em grupo. Assim, se um servidor falhar, os jogadores poderão continuar o jogo.

1. Introdução

O problema dos jogos multi-utilizador utilizando servidores replicados não é novo e já existem inúmeras propostas de resolução para este problema. Jogos hoje em dia muito populares e com uma incontável legião de fans tornaram necessário o desenvolvimento de arquitecturas que suportem de modo eficiente este tipo de jogos. Nos sistemas distribuídos quando um servidor falha, pode provocar a falha total do sistema em que está inserido. Muitas aplicações distribuídas já não usam um modelo básico cliente-servidor, onde as interacções só ocorrem entre duas entidades, usam sim uma aproximação com aplicações multi-participantes, onde os dados precisam de ser enviados para um grupo. É neste momento, quando se tem um grupo de entidades, que o problema se complica. Pretende-se garantir que, ao falhar uma componente, todas as outras continuarão a funcionar. Para se atingir este objectivo, usa-se a replicação de dados entre os servidores, onde cada um destes, utiliza a comunicação em grupo e guarda o estado actual do sistema, e se um falhar, os restantes continuarão a funcionar. A solução que iremos apresentar utilizará a plataforma de comunicação APPIA a qual nos irá disponibilizar vários protocolos que irão facilitar a comunicação, especialmente entre os vários servidores de modo a manter a informação coerente de cada um, em comparação com os outros. Com o auxílio do APPIA esperamos conseguir tornar os servidores tolerantes a faltas e permitir que mesmo que falhe um servidor, e mesmo que esse servidor seja o servidor com o qual os clientes (jogadores) estejam a interagir, o jogo continue. Um dos principais objectivos deste projecto é mostrar que utilizando ferramentas já disponíveis (ex: APPIA) podemos poupar trabalho e aumentar eficácia na resolução deste tipo de problemas. Este artigo está organizado da seguinte forma. A secção 2 descreve a arquitectura do nosso sistema. A secção 3 trata da comunicação fiável em grupo. Na secção 4 é apresentado o tipo de ordenação de mensagens utilizado no nosso projecto. A secção 5 descreve como é realizada a eleição do coordenador e para que serve. A secção 6 apresenta a pilha protocolar do cliente e do servidor, com uma breve descrição de cada camada nelas contida. Na secção 7 são descritos os mecanismos de detecção e recuperação de falhas. A secção 8 apresenta o trabalho relacionado. Por fim a secção 9 apresenta as conclusões relativas a este projecto.

2. Arquitectura

Este sistema distribuído apresenta uma aplicação multi-utilizador, logo existe uma grande probabilidade de ocorrerem falhas (de um ou mais servidores). Quando uma falha deste tipo ocorrer é suposto o sistema tolerá-la, proporcionando assim a continuação do funcionamento do jogo. A solução passa por utilizar os protocolos oferecidos pelo Appia para garantir comunicação em grupo e sincronia virtual [1]. Quando o cliente começa a execução pede uma lista de endereços dos servidores. Irá tentar efectuar a sua ligação ao primeiro servidor da lista recebida. Cada servidor vai aceitar no máximo duas ligações. Se outro cliente se tentar ligar ao servidor já lotado, este irá contactar outros servidores, de modo a obter o endereço de um que não esteja lotado, para sugerir ao cliente. Optámos por esta solução porque assim evitamos a ligação de todos os clientes ao mesmo servidor, de modo a distribuir a carga entre os servidores.

3. Comunicação fiável em grupo

O Appia foi criado para suportar vários modelos de comunicação, entre os quais protocolos de comunicação

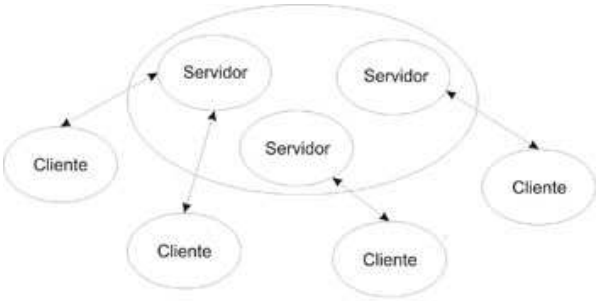

Figura 1. Esquema da arquitectura do sistema

em grupo. Estes foram desenvolvidos para fornecer todos os mecanismos necessários para garantir a sincronia virtual. Esta foi a solução adoptada para a comunicação em grupo. Sincronia virtual é um modelo de comunicação em grupo, que afirma que todos os participantes no grupo vêm os membros de um grupo na forma de vista, e que todos vêm as mesmas mudanças na vista, pela mesma ordem, existindo ordem total em todas as vistas. Afirma também que todas as mensagens enviadas numa vista são entregues nessa vista, e se for necessário uma mudança de vista então todas as mensagens da vista corrente são entregues antes da mudança de vista ser executada. Por isso, para garantir uma mudança correcta de vista são necessários três passos:

- O grupo tem de ser bloquado para garantir que o 2º passo vai terminar.

- Todas as mensagens de grupo recebidas por qualquer membro, são entregues a todos os membros antes da nova vista.

- A nova vista é entregue a todos os membros.

As mudanças de vista ocorrem quando um membro do grupo falha e deixa o grupo, enquanto novos membros se podem juntar a este mesmo grupo[1].

4. Ordenação das mensagens

Para garantir a coerência do estado do jogo, temos de assegurar que todas as mensagens enviadas são entregues pela mesma ordem pela qual foram enviadas. A ideia de ordem total é que todas as mensagens enviadas pelo grupo têm de ser recebidas pela mesma ordem pela qual foram entregues[4]. Optámos pela ordem total em detrimento da ordem FIFO e da causal, de modo que, em conjunto com a sincronia virtual, alcançamos a coerência desejada durante a execução do jogo, nos vários servidores.

5. Coordenador

Existe uma necessidade de consenso no modelo que apresentamos em que os clientes estão distribuídos pelos vários servidores. Esse consenso pode ser atingido por vários algoritmos, mas optou-se pela eleição de um coordenador que toma a decisão e a comunica aos restantes membros do grupo quando necessário. Este coordenador é sempre o primeiro elemento da vista actual (resolvendo deste modo o problema da falha do servidor coordenador). Por exemplo, se num jogo um dos servidores decide que a bola bate no paddle, e outro servidor decide que bate na parede (incrementa um na pontuação do jogador), torna-se necessário a existência de consenso de modo a tornar os servidores coerentes.

6. Pilha Protocolar

Uma pilha protocolar é uma pilha de protocolos. Cada combinação de layers (camadas) na pilha oferece um conjunto de propriedades e pode ser considerada como o serviço oferecido pela pilha. As propriedades que resultam de cada combinação e os protocolos utilizados na pilha são permutáveis e são chamados Qualidade de Serviço ou QoS (Quality of Service). Uma QoS é uma descrição estática de um conjunto ordenado de protocolos. Um canal é uma instância dinâmica de uma QoS. Neste sistema definimos dois QoS um para os servidores e outro para os clientes[2].

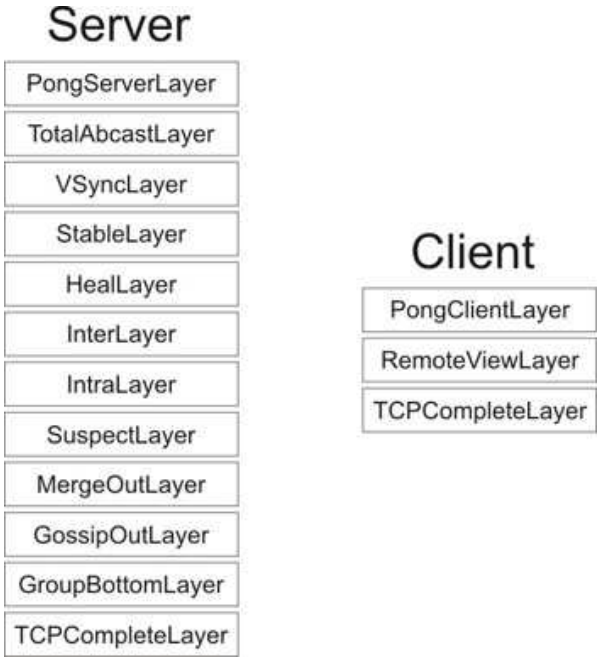

Figura 2. Pilhas protocolares

6.1. Pilha protocolar para o cliente[1]

PongClientLayer - Camada de suporte à interface do jogo.

RemoteViewLayer - É a camada responsável por obter o id do grupo e um array de endereços dos membros desse grupo, através do envio, à camada superior, de um RemoteViewEvent[3].

TcpCompleteLayer - Responsável pela comunicação entre os processos usando o protocolo TCP.

6.2. Pilha protocolar para o servidor[1]

PongServerLayer - Garante a ligação entre a pilha de comunicação em grupo e o servidor.

TotalAbcastLayer - Garante a ordem total na entrega das mensagens à camada acima.

VSyncLayer - Verifica se a mudança de vista está correcta; bloqueia os membros do grupo, e verifica se todos os membros receberam as mesmas mensagens antes da mudança de vista.

StableLayer - Entregar as mesmas mensagens a todos os membros vivos - Reliable Broadcast.

HealLayer - Detecção de existência de vistas concorrentes.

InterLayer - Efectua o merge entre duas vistas concorrentes do mesmo grupo

IntraLayer - Interage com outras camadas para garantir correcta mudança de vista

SuspectLayer - Detecta falhas - Propaga a suspeita pelos outros membros do grupo

MergeOutLayer - Fusão de uma nova vista com as outras. View merge é executado pelo InterLayer

GossipOutLayer - (Gossip) Servidor externo para detecção de vistas concorrentes. - Responsável pela comunicação entre o gossip e a pilha de Comunicação.

GroupBottomLayer - Interface entre as camadas de comunicação em grupo e camadas point-to-point; tem de ser a camada mais baixa das camadas da comunicação em grupo

TcpCompleteLayer - Responsável pela comunicação entre os processos usando o protocolo TCP.

7. Tolerância a faltas

Existe a possibilidade de um servidor falhar, o objectivo deste projecto é tolerar essas falhas de modo a transferir os clientes que estavam a utilizar o servidor que falhou para um que esteja a correr sem que o jogador se aperceba de tal mudança. Contamos com a plataforma de comunicação APPIA.

7.1. Detecção de falhas

Com a plataforma de comunicação APPIA conseguimos detectar a falha de um servidor dentro do grupo por intermédio das vistas[1]. Uma vista é um conjunto com todos os membros do grupo (neste caso o grupo serão todos os servidores). Quando um membro do grupo falha, ele é retirado da vista e a nova vista passa a ser a vista actual (todos os elementos do grupo passam a ter conhecimento da nova vista e deste modo tambem da falha). Isto é assegurado pela QoS que propomos para os servidores.

7.2. Recuperação de falhas

Quando um servidor falha, os clientes a ele ligado têm de se ligar a outro servidor de modo a poderem continuar a jogar. Isto é feito através da detecção da falha (o servidor deixa de responder) e é feita uma ligação a um novo servidor. Como todos os servidores têm guardado o estado do jogo (pontuações de todos os jogadores e ip:port de cada cliente), é possível mudar o servidor ao qual o cliente estava ligado, mantendo o estado do jogo.

8. Trabalho relacionado

Depois de consultar vários documentos referentes ao APPIA, apresentados mais à frente nas referências, incluindo relatórios disponiblizados na página da cadeira e livros que se focam na resolução do problema de tolerância a faltas, chegámos à nossa solução. Consiste em ter um grupo de servidores com o mesmo estado de jogo, um grupo de clientes, cada qual com uma ligação a um dos servidores, tendo cada servidor no máximo duas ligações. Quando um cliente tenta efectuar uma ligação a um servidor que já tem o número máximo de ligações, o servidor comunica ao cliente que não pode aceitar mais ligações, mas sugere um novo servidor ao qual o cliente se deverá ligar.

9. Conclusões

Neste artigo foi apresentada uma solução para um jogo multi-utilizador em que é utilizada a comunicação em grupo para replicar o servidor do jogo (o jogo está implementado numa base servidor-cliente), de maneira que os clientes possam continuar a jogar mesmo que um dos servidores falhe. Nesta solução quando um servidor falha os clientes ligados a este servidor, reconhecem a falha, e ligam-se a um novo servidor (dentro do grupo), este ou aceita a ligação, ou no caso de já ter o numero de ligações maximo, sugere um novo servidor (tambem dentro do grupo), ao qual o cliente se pode ligar. O cliente começa por enviar um evento do tipo RemoteViewEvent[3]. Este evento é usado para pedir uma lista de endereços de um determinado grupo (grupo

dos servidores) ao GossipServer[1]. Este último envia o id do grupo e a lista de endereços dos membros desse mesmo grupo. A partir daqui se o servidor ao qual o cliente está ligado falhar, este já sabe a quais se poderá ligar. Se o cliente se ligar a um servidor que já tem o número de ligações máximo este informa o cliente que a ligação não vai ser possível e sugere um novo servidor ao qual o cliente se deve tentar ligar. Assim, evitamos a ligação de todos os clientes ao mesmo servidor, de modo a distribuir a carga entre os servidores.

Referências

[1] A. Pinto, *Appia Group Communication*, October 2005

[2] H. Miranda, A. Pinto, N. Carvalho, L. Rodrigues, *Appia Protocol Development Manual*, October 2005

[3] URL: *http://appia.di.fc.ul.pt/docs/javadoc/*

[4] R. Guerraoui, L. Rodrigues, *Introduction to Reliable Distributed Programming*, Springer-Verlag, September 2005