Pong a 4 jogadores Distribuído e Tolerante a Faltas

TFD09

Fernando Real
Departamento de Informática
i26526@alunos.di.fc.ul.pt

Miguel Valente
Departamento de Informática
i28799@alunos.di.fc.ul.pt

Sérgio Brotas
Departamento de Informática
i29509@alunos.di.fc.ul.pt

Abstract

O Pong a quatro jogadores é um jogo simples cuja implementação é também ela simples se considerarmos que existe um servidor centralizado com o qual todos os processos comunicam. A falha no servidor origina assim a falha de todo o sistema inviabilizando a continuação do jogo. Este artigo descreve o Pong jogado a quatro, distribuído e tolerante a faltas em que a informação do estado do jogo é replicada entre múltiplos servidores, utilizando protocolos de comunicação em grupo. Assim, a falha de um servidor não origina o fim do jogo, podendo os jogadores continuar a jogar sem que a mesma falha seja observada.

1 Introdução

Nos dias que correm existe uma necessidade crescente em toda a comunidade informática de tornar os sistemas distribuidos existentes em sistemas tolerantes a faltas. Esta necessidade ocorre não só em sistemas de alto risco tais como o controlo de tráfego aéreo ou de informação militar, como em sistemas de encaminhamento na Internet ou mesmo sistemas lúdicos.

A nossa arquitectura tenta aproveitar as vantagens dos sistemas distribuídos usando o Appia para tornar o sistema tolerante a faltas.Resumidamente, existe um conjunto de servidores que contêm uma cópia do estado do jogo e dos ips clientes, podendo a qualquer momento e em caso de falha do servidor principal (coordenador) tomar o seu lugar, podendo o jogo prosseguir sem precalços. Para aliviar a carga do servidor onde os clientes estão ligados existe um servidor responsável, não só por difundir cada nova vista pelos elementos do grupo, como também por difundir o estado do jogo actual, que lhe é fornecido pelo servidor coordenador.

Este artigo está organizado da seguinte forma: Na secção seguinte descrevemos a interface gráfica da aplicação cliente. Na terceira secção é descrita a arquitectura do sistema que pretendemos implementar, focando especificamente como se processa a comunicação em grupo entre servidores, a definição de *estado de jogo*, e o método de propagação de mensagens. De seguida são descritas as falhas do sistema e as soluções que propômos, ainda os algoritmos de eleição que serão usados, a pilha protocolar usada para garantir a qualidade de serviço. Terminamos com a conclusão e a bibliografia.

2 Interface Gráfica

A interface gráfica é composta por uma bola e quatro plataformas rectangulares, correspondendo cada uma a um jogador, contidos numa janela que representa o campo de jogo. Cada jogador movimenta uma plataforma, mas apenas numa das arestas do campo de jogo, esta plataforma pode ser controlada pelo teclado (cursor) ou pelo rato. As movimentações efectuadas pelos jogadores são comunicadas ao servidor através do canal Appia criado para o efeito, por sua vez o servidor envia para cada cliente pelo respectivo canal as posições dos seus adversários.

A comunicação fiável vai permitir ao servidor garantir que quando envia uma mensagem a informar do novo estado do jogo, todos os processos clientes vão recebê-la, podendo por isso alterar também o estado da interface.

3 Arquitectura do sistema

Um dos objectivos propostos é ter um grupo com servidores replicados, onde caso haja uma falha num dos processos, é garantido que o jogo continua a decorrer. Na nossa arquitectura, descrita na figura 1, o grupo é formado por três tipos de servidores com funções distintas, que qualquer um destes pode assumir:

- *Coordenador* - servidor que interage com os clientes. Responsável por receber as ligações e pedidos dos clientes e reencaminhá-los para os outros clientes. A cada alteração do estado do jogo, envia dados para o nosso *Gossip server* sobre o novo estado.

- *Gossip Server* - servidor que terá uma dupla função. Será não só responsável por propagar as novas vistas do grupo, como também por receber mensagens do coordenador sobre o estado do jogo e reencaminhá-las para as réplicas, de modo a não sobrecarregar o coordenador com a difusão de mensagens.

- *Réplicas* - servidores auxiliares que se encontram em espera, mantendo uma réplica do estado do jogo. Caso ocorra uma falha no *Coordenador* ou no *Gossip server* encontram-se preparados para tomar o seu lugar.

3.1 Propagacao de Mensagens

Para o efeito de propagar o estado do jogo e os ips clientes utilizamos uma variação do algoritmo Best-Effort que consiste em: sempre que existe uma alteração no estado do jogo, o servidor coordenador envia uma mensagem para o servidor de gossip, que por sua vez propaga a mensagem para todos os elementos do grupo incluindo para si próprio. Este servidor é também responsável por propagar as novas vistas do grupo. Utilizamos esta solução por forma a minimizar o tráfego para e do servidor coordenador.

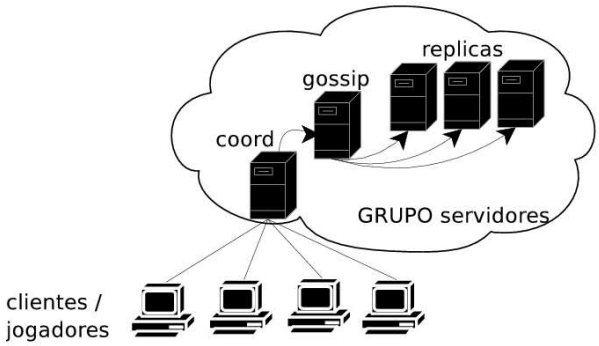

Figura 1. Esquema da Arquitectura

3.2 Comunicacao em grupo

Para a comunicação em grupo baseamo-nos no conceito de sincronia virtual, utilizando os protocolos já existentes no Appia para suporte à comunicação em grupo. O modelo de vistas é adequado ao nosso sistema, facilitando em larga medida a gestão da sincronia dos estados das várias réplicas. A sincronia virtual é um paradigma da comunicação em grupo que estipula que:

- todos os participantes de um grupo vêem a sua constituição em forma de vistas.
- os participantes de um grupo vêem as mesmas mudanças de vista pela mesma ordem.
- todas as mensagens de uma vista são entregues nessa vista.
- antes de uma mudança de vista, todas as mensagens entregues a um membro são entregues aos restantes membros.
- as mudanças de vista ocorrem porque se espera que os grupos sejam dinâmicos, isto é, que alguns membros falhem ou saiam do grupo enquanto novos membros se juntam ao mesmo.

No Appia não existe uma operação que permita a um membro entrar num grupo, no entanto existe um mecanismo de junção de vistas. Assim, cada vez que um processo deseje juntar-se a um grupo apenas tem

de criar uma vista desse grupo com um único membro, ele próprio. Depois, o protocolo de junção tratará de unir essa nova vista a uma vista possivelmente já existente o que resultará na entrada do novo membro. O mecanismo de união está dividido em duas partes:

- detecção da nova vista.
- união das vistas. A união das vistas está a cargo do protocolo *Inter* implementado pela *InterLayer* e correspondente *InterSession*.

Por outro lado, a detecção da nova vista é efectuada pelo protocolo *Heal*, implementado pela *HealLayer* e correspondente *HealSession*. O multicast é efectuada usando um servidor externo, *GossipServer* que recebe todas as mensagens do servidor coordenador retransmitindo-as para todos os processos conhecidos, isto é, todos os processos dos quais já recebeu mensagens.

3.3 Estado do jogo

Foram necessárias certas alterações à noção de estado de jogo que nos foi fornecida no protótipo. Segundo a concretização dos docentes, um estado é constituído por: posição da bola, posição dos paddles dos jogadores e as respectivas pontuações. Para podermos concretizar o trabalho segundo a nossa concepção, sentimos necessidade de acrescentar ao estado do jogo uma lista de IPs, dos clientes que estão ligados naquele instante. Deste modo, o coordenador pode enviar as mensagens com o estado do jogo para o gossip server e este difunde-as pelas réplicas e assim todos os servidores saberão o IP de todos os clientes.

4 Possíveis falhas e soluções

Nos casos que descrevemos, a falha é detectada por meio da camada Suspect (SuspectLayer) disponibilizada pelo *Appia*. A sua ocorrência e recuperação deverá ocorrer de maneira imperceptível para o jogador.

- o *Coordenador* falha - após detecção da falha, através do algoritmo de eleição é escolhida uma réplica que altera o seu estado para Coordenador. Este novo coordenador anuncia-se aos clientes para que estes passem a comunicar com ele. O Gossip Server, como parte do grupo, sabe que foi eleito um novo coordenador, com quem passa a comunicar.

- o *Gossip Server* falha - após detecção da falha, como no caso anterior é escolhida uma das réplicas através do algoritmo de eleição, a qual altera o seu estado para Gossip Server. Ao receber a nova vista, o coordenador passa a comunicar com o gossip server eleito e este difunde o estado do jogo para as réplicas. Para as réplicas, esta alteração é transparente, pois continuam a receber mensagens sobre o estado do jogo, continuando em stand-by à espera que sejam necessárias.

5 Algoritmos de Eleição

Para eleger o Coordenador ou o Gossip Server, deparámo-nos com dois possíveis algoritmos de eleição, cada um com os seus benefícios e contrariedades:

5.1 Menor Atraso

Quando o grupo recebe uma nova vista em que o coordenador falhou, todas as réplicas que estão em repouso (que têm os IPs dos clientes contidos no estado do jogo), fazem ping a todos clientes. No caso de falha do Gossip Server as réplicas fazem ping ao Coordenador. Nos dois casos, aquele servidor que receber uma resposta a um ping com o valor mais baixo é o eleito. A vantagem deste método é que aumentamos a probabilidade de eleger o servidor que está menos sobrecarregado e cujos canais de comunicação apresentam maior eficiência naquele instante. A desvantagem é que vamos perder algum tempo a fazer os pings das réplicas para os clientes e a aguardar a resposta do mais rápido.

5.2 Sequencial

Este será o modelo mais simples, de mais fácil implementação, em que as réplicas estão ordenadas sequencialmente. Ao receber uma nova vista, o grupo apercebe-se de que um dos servidores (coordenador ou gossip) falhou. A réplica que esteja no primeiro lugar da sequência é eleita. A vantagem deste método é que não vamos perder tempo a fazer pings aos

clientes, sendo a eleição feita mais rapidamente. A desvantagem é que podemos escolher uma réplica que esteja na altura com tráfego intenso ou com muitos processos a correr, o que poderá trazer atrasos significativos ao nível da jogabilidade.

6 Pilha Protocolar

A Qualidade de Serviço (QoS) que define as propriedades do canal de comunicação, é baseada nos protocolos do Appia que são utilizados e na forma como interagem. Definimos uma pilha protocolar que tem as propriedades habituais de um sistema com sincronismo virtual acrescidas da camada ClientLatencyLayer implementada por nós:
- todos os membros de um grupo observam o seu grupo sobre a forma de vistas.
- todos os membros de um grupo observam a mudança de vista pela mesma ordem. Existe uma ordem total das vistas.
- todas as mensagens enviadas durante uma determinada vista são entregues nessa mesma vista.
- se uma mudança de vista é necessária, todas as mensagens são entregues antes da mudança da vista.
Outra propriedade importante é a oferecida pela ordenação total de mensagens que, como já foi referido, garante que as mensagens são entregues pela mesma ordem a todos os processos intervenientes. Com este conjunto de propriedades garantimos que os estados dos vários processos estão sincronizados, e garantimos também a tolerância a faltas provocadas por falhas de processos. Usando o modelo de propagação *Best-Effort*, a pilha protocolar terá o seguinte aspecto:

ApplSupportLayer
BestEffortLayer
ClientLatencyLayer
TotalAbcastLayer
VSyncLayer
StableLayer
HealLayer
InterLayer
IntraLayer
SuspectLayer
MergeOutLayer
GossipOutLayer
GroupBottomLayer
TCPCompleteLayer

7 Conclusão

O jogo Pong a 4 jogadores Distribuído e Tolerante a Faltas descrito neste artigo será uma aplicação distribuída e fiável, de modo a conseguir tolerar falhas tanto a nível de clientes como de servidores. Do ponto de vista da implementação, o nosso sistema irá utilizar comunicação em grupo com o auxilio da plataforma Appia, em que o nosso grupo será o conjunto de todos os servidores: coordenador, gossip server e réplicas. Através do auxilio do Appia procuramos que o nosso sistema seja tolerante a faltas, de modo que qualquer falha que aconteça seja imperceptível para os jogadores.

8 Bibliografia

[1] Paulo Verissimo e Luís Rodrigues, "Distributed Systems for System Architects", Kluwer Academic Publishers ISBN 0-7923-7266-2
[2] Alexandre Pinto,"Appia Group Communication", 2005
[3] Rachid Guerraoui e Luís Rodrigues."Introduction to Reliable Distributed Programming"(Preliminary Draft), Springer-Verlag 2005
[4] Nuno Carvalho e Luís Rodrigues "Implementing Reliable Broadcast Protocols in Appia - A Brief Tutorial V1.1"2003