Jogos online de elevada disponibilidade

Jorge Lima
i23918@alunos.di.fc.ul.pt

Eduardo Andrade
i26048@alunos.di.fc.ul.pt

Rui Malhado
i28804@alunos.di.fc.ul.pt

Grupo TFD06

Abstract

Os serviços online representam um mercado emergente cada vez mais significativo para a economia portuguesa e mundial. Tendo em conta o crescente nível de exigência do consumidor médio, faz sentido estudar formas de aumentar a disponibilidade dos serviços que estes procuram.

Este artigo descreve maneiras de conseguir essa disponibilidade no contexto dos jogos interactivos em tempo real, nomeadamente o Pong a quatro jogadores, utilizando pilhas protocolares e servidores replicados para garantir ao consumidor a qualidade desejada.

1. Introdução

Com o aparecimento do computador pessoal de baixo custo surgiram novas maneiras de pensar e agir no contexto dos sistemas de informação com requisitos elevados de desempenho e fiabilidade. Onde antigamente se usava um sistema centralizado baseado num único computador altamente especializado (mainframe), começaram a aparecer soluções que reuniam o poder computacional de muitos computadores baratos para atingir os mesmos objectivos. Por terem sido planeados com o objectivo de reduzir custos, e para usos pouco exigentes, usam frequentemente componentes lógicos e físicos com uma probabilidade de falharem mais elevada, pelo que se torna necessário implementar sobre eles uma camada de fiabilidade, que explora a redundancia para que o sistema como um todo possa tolerar um certo numero de faltas e continuar a funcionar de forma transparente. Chama-se "disponibilidade" (availability) a esta propriedade de um sistema manter o seu funcionamento, continuando a prestar correctamente os serviços que oferece face a diversos tipos de problemas. Uma técnica frequentemente usada para garantir a disponibilidade de um sistema é a replicação, que trata de distribuir o estado do sistema por várias máquinas, para que o sistema possa migrar os clientes de umas máquinas para outras à medida que estas forem falhando.

1.1. Domínio aplicacional

Como exemplo de um jogo online multi-utilizador com requisitos de interactividade em tempo real, utilizamos uma variante de um jogo muito antigo chamado "Pong". Este jogo, inventado no ínicio da década de 1970 pela Atari, foi o primeiro jogo de vídeo comercialmente viável, consistindo de uma adaptação para computador do tradicional "Ping-Pong" (ténis de mesa). Cada jogador tem uma raquete rectangular que pode mover de um lado para o outro, procurando acertar numa bola que circula continuamente pelo campo de jogo, como objectivo de evitar que esta toque na parede do seu lado. Cada jogador que falhar a bola, não conseguindo impedir a colisão da mesma com a sua parede é penalizado com um ponto. O jogador com menor número de pontos vence.

1.2. Modelos de replicação

Ao longo dos anos foram surgindo vários modelos de replicação para várias finalidades, cada um com as suas vantagens e desvantagens.
O modelo de replicação activa[1] consiste na execução em paralelo de cópias exactas da mesma tarefa, produzindo exactamente o mesmo resultado. Após a terminação da tarefa pode ser utilizado qualquer um dos resultados. Este modelo tem como desvantagem o facto de que só pode ser usado em sistemas deterministas. No modelo de replicação passiva[1] apenas uma réplica, designada primária, executa os comandos recebidos, enquanto as outras réplicas armazenam todas as mensagens recebidas e só entram em execução após falha da réplica primária. Embora se possam introduzir "checkpoints" (pontos de sincronização intermédios) tem sempre como desvantagem uma certa latência na transição de uma réplica para outra.
Num sistema que se pretende interactivo é necessário minimizar esta latência. No limite, poder-se-iam realizar estas sincronizações para todos os comandos, mas num sistema não-determinista isto significaria que as réplicas poderiam divergir. O modelo de replicação semi-activa[1],

também chamada de “líder-seguidor”, resolve os problemas dos dois modelos anteriores delegando todas as decisões não-deterministas para a réplica primária, ou líder.
Existe ainda um modelo de replicação “preguiçosa” (lazy)[1] que mistura conceitos dos modelos de replicação activa e semi-activa. Este modelo tem a particularidade de aproveitar as características de cada aplicação individual para tomar decisões sobre a necessidade ou não da ordenação de mensagens para a manutenção da coerência.

1.3. Arquitectura

No caso particular do Pong, a única fonte de indeterminismo ocorre no momento da colisão. Entre cada duas colisões a bola segue um movimento linear perfeitamente previsível, sendo que o cálculo de uma nova direcção e pontuação depende estritamente do ângulo e tipo de colisão que ocorreu. Tendo como inspiração os modelos acima descritos pode ser projectada uma arquitectura em que a simulação seja executada por todas as réplicas em paralelo, tendo como pontos de sincronização os momentos em que ocorrem colisões. A réplica primária será considerada autoritária na manutenção da coerência do estado em todas as réplicas, e caso falhe, será substituída pela réplica seguinte numa hierarquia pré-definida. A execução simultânea do jogo em todas as réplicas possibilitará uma baixa latência de recuperação em caso de falha da réplica primária, e se restringirmos a sincronização aos momentos de colisão minimizamos a troca de mensagens na rede.

2. Concretização

2.1. Eleição do Coordenador

Caso haja uma falha no coordenador, a primeira réplica da vista é escolhida como novo coordenador. Podia-se melhorar o algoritmo de eleição implementando um tipo de balanceamento de carga, em que o novo coordenador é escolhido com base na maior disponibilidade de processador/memória.

2.2. Modelo de Faltas

Iremos usar o modelo de faltas ”fail-stop”, em que um servidor ao falhar não recupera, e a falha é detectada.

2.3. Appia

O Appia[2] é um sistema modular de comunicação, usando alguns protocolos já definidos, cada um utilizando os serviços do anterior para permitir maior flexibilidade na definição do protocolo de comunicação. Para tornar o sistema tolerante a faltas, usando replicação a nível do servidor, teremos que adicionar camadas à pilha de protocolos fornecida inicialmente no trabalho.

A construção de uma aplicação distribuída, difere de outras pela necessidade de garantir diversos níveis de funcionamento e de ajustamento dinâmico às alterações de rede. O Appia é um sistema modular de comunicação, oferendo a implementação de protocolos específicos que podem ser combinados da forma desejada, de forma a atingir uma determinada qualidade de serviço (QoS).

O Appia distingue vários tipos de abstracções : Camada, QoS, Sessão, Canal e Evento. Uma camada é uma descrição estática dum protocolo e uma sessão é uma implementação de uma camada. Um QoS descreve um conjunto ordenado de protocolos, bem como a comunicação entre cada um deles. Uma camada é uma implementação do QoS e as sessões utilizam um canal para comunicarem. A comunicação entre cada sessão é efectuada através de eventos que são transmitidos no canal.

2.4. Pilha de protocolos

A pilha de protocolos usada no jogo fornecido usa apenas duas camadas: o cliente usa uma camada PongClientLayer para tratar as mensagens que recebe do servidor e da interface gráfica, usando depois a camada TcpCompleteLayer para enviar os dados com garantia de entrega. Para o servidor acontece o mesmo, uma camada PongServerLayer para tratar os pedidos dos clientes, e mover a bola quando recebe um evento temporizador, e a camada TcpCompleteLayer para enviar os dados.

De forma a atingir os objectivos primordiais deste trabalho, é necessário introduzir algumas propriedades que não foram inicialmente implementadas. Assim, para tornar o sistema tolerante a faltas, usando replicação a nível do servidor, teremos que adicionar camadas à pilha de protocolos.

No caso do cliente, vamos apenas adicionar uma camada intermédia entre a PongClientLayer e a TcpCompleteLayer, que vai esconder do nível da aplicação as mudanças de coordenador. Esta camada será responsável por efectuar uma mudança de ligação do cliente para um novo coordenador, de forma transparente para a aplicação do cliente.

No caso do servidor vamos adicionar as seguintes camadas fornecidas no Appia[3], que juntas, fornecem comunicação em grupo e sincronia virtual:

VSyncLayer	Sincronia Virtual
LeaveLayer	Remove um membro do grupo
StableLayer	*
HealLayer	Detecção de novas vistas
InterLayer	União de vistas
IntraLayer	*
SuspectLayer	Detector de falhas
GossipOutLayer	Comunicação de vistas para o grupo
GroupBottomLayer	*

Nas camadas assinaladas a asterisco (*) não existe informação suficiente para determinar o seu propósito, porém são incluídas por serem necessárias para obter a sincronia virtual (VSyncLayer).

Em cima destas iremos construir uma camada de gestão da replicação, que irá salvaguardar o estado do jogo do coordenador nas réplicas, juntamente com a identificação dos clientes ligados, para que, caso o coordenador corrente falhe, seja escolhido um novo, que irá comunicar com a camada CoordinationHiderLayer dos clientes informando-os que é o novo coordenador.

Para o cliente iremos ter:
PongClientLayer
CoordinationHiderLayer
TcpCompleteLayer

Para o servidor(coordenador e réplicas):
PongServerLayer
CoordinationLayer
VSyncLayer
LeaveLayer
StableLayer
HealLayer
InterLayer
IntraLayer
SuspectLayer
MergeOutLayer
GossipOutLayer
GroupBottomLayer
TcpCompleteLayer

Esta pilha de protocolos talvez possa ser melhorada a nível de eficiência(menos mensagens trocadas pela rede) usando Nack-FIFO em UDP, em vez de TCP.

2.5. Eventos

Quanto a novos eventos, a camada CoordinationHiderLayer terá que tratar dos eventos de mudança de coordenador, mudando o destinatário das mensagens para o novo coordenador, para evitar que a camada PongClientLayer se aperceba da mudança de coordenador e continue a funcionar como anteriormente. Esta mudança será gerida com base em eventos gerados na eleição do novo coordenador e este, após assumir o comando irá contactar os clientes, através de um evento, informando-os da sua posição.

A camada CoordinationLayer vai ter que realizar a eleição do novo coordenador quando o anterior desaparece da vista, bem como guardar os estados que o coordenador lhe envia periodicamente. A camada PongServerLayer terá que ser alterada para poder receber os estados enviados pela camada CoordinationLayer e alterar o seu estado.

3. Trabalho Futuro

Numa primeira aproximação, por questões de simplicidade e desenvolvimento incremental, projectou-se um sistema baseado em fortes assumpções sobre o meio existente. A restrição do sistema a faltas de paragem é pouco realista, não há qualquer garantia de que uma réplica falhe estritamente por paragem sem recuperar. Poder-se-ia estender o sistema de forma para ter em conta a recuperação, tolerar faltas de omissão ou mesmo de valor. Como foi explicado anteriormente, a natureza do Pong permite-nos optimizar preemptivamente a troca de mensagens em rede, mas de forma alguma se pode afirmar que a escalabilidade do sistema foi totalmente analizada, pelo que um futuro desenvolvimento deste sistema teria necessariamente como objectivo uma análise mais profunda da troca de mensagens efectuada pelo Appia e de combinações alternativas de camadas que permitiriam uma utilização mais eficiente da rede. Nomeadamente, para uma execução em rede local, poder-se-iam utilizar as funcionalidades de difusão inerentes ao protocolo UDP com o possível sacrifício de alguma fiabilidade no nível mais baixo da rede que poderia no entanto ser compensada pelas camadas superiores. Seria também interessante medir os atrasos de rede antes do início de um jogo, por forma a garantir que este execute a uma velocidade ajustada á velocidade da rede subjacente o que permitiria uma maior estabilidade do estado do jogo: Se a comunicação for demasiado lenta isso poderá provocar a ocorrência de artefactos estranhos como o transporte instantâneo da bola de um lado para o outro do campo de jogo, etc.

4. Conclusão

Através da replicação de servidores e de pilhas protocolares, é possível garantir uma maior fiabilidade e disponibilidade de serviços. Neste caso particular, trata-se de um jogo, contudo pode estender-se a qualquer aplicação que funcione de forma distribuída. Com este jogo simples que funciona numa arquitectura cliente-servidor, podemos observar com algum detalhe como é aumentada a fiabilidade e disponibilidade, sobre um sistema convencional, através do uso de software adequado.

Referências

[1] Paulo Veríssimo e Luís Rodrigues *Distributed Systems for System Architects*. Kluwer Academic Publishers ISBN 0-7923-7266-2

[2] URL: http://appia.di.fc.ul.pt/

[3] Pinto, Alexandre *Appia Group Communication Manual*. 2001