FTP - Fault-Tolerant Pong
Grupo 08

José Côrte-Real 29037
Leonel Duarte 25887
Miguel Figueiredo 28152

23 de Outubro de 2005

Resumo

Ao longo do tempo, as aplicações informáticas têm vindo a evoluir de modelos de computação centralizada, para modelos de computação distribuída. Tal evolução foi acompanhada por um acréscimo de complexidade na implementação de abstracções e algoritmos relativos às aplicações distribuídas. Neste panorama é importante a utilização de técnicas de tolerância a faltas para garantir a disponibilidade de um serviço

No âmbito da disciplina de Tolerância a Faltas Distribuídas foi proposto aos autores deste projecto que concretizassem o jogo Pong para múltiplos jogadores , com tolerância a faltas.

1 Introdução

Pong consiste num jogo da autoria de Nolan Bushnell lançado pela Atari em 1972 e que se tornou no primeiro grande sucesso comercial da - naquela altura - emergente indústria dos jogos de computador. O jogo consiste na adaptação de um jogo de ténis de mesa a uma plataforma de jogo de computador:

- A cada jogador é fornecida uma *paddle* ou raquete que usa para proteger a sua parede contra a colisão de uma bola que se encontra em movimento no tabuleiro do jogo;
- A bola do jogo descreve um movimento linear que é alterado aquando de uma colisão com uma parede ou com uma raquete de um jogador;
- Um jogador perde um ponto - ou sofre um golo - sempre que a bola colidir com a parede que protege.

Originalmente o jogo apenas suporta dois jogadores num tabuleiro rectangular e o controlo de ambos efectua-se localmente no computador que executa o jogo.

Por razões académicas e de modo a tornar este simples jogo mais interessante e tecnicamente mais complexo, propõem-se duas modificações:

- Suporte para mais de dois jogadores
- Suporte à computação distribuída fiável

O jogo a implementar no âmbito da disciplina de Tolerância a Faltas Distribuídas consiste numa área de jogo quadrangular em que cada jogador controla uma *paddle* que utiliza para impedir que uma bola que se movimenta dentro da área de jogo, embata na *sua* parede. Cada jogador inicia o jogo com 0 pontos e por cada golo (colisão com a sua parede) que sofrer, acrescenta um ponto à sua pontuação (número de golos sofridos).

Este jogo embora seja relativamente simples, captura requisitos essenciais para outras aplicações distribuídas , sendo que as aproximações utilizadas para resolver este problema podem servir de trabalho futuro para problemas mais práticos e complexos.

O resto deste artigo apresenta-se organizado da seguinte forma: Na secção 2 apresenta-se o problema e a ferramenta a utilizar para o resolver, na secção 3 a relação entre clientes e servidores, a secção 4 refere-se a conceitos importantes para a resolução do problema, na secção 5 enumeram-se duas arquitecturas possíveis para resolver o problema , na 6 apresenta-se trabalho a realizar no futuro, e finalmente na 7, conclui-se o artigo.

2 Descrição do problema

O jogo a implementar suporta apenas uma lotação máxima de 4 jogadores simultaneamente, com uma lista de espera para jogadores adicionais.

O projecto seguirá o modelo cliente-servidor, tendo em conta a preocupação de tornar o servidor tolerante a faltas. Para alcançarmos este objectivo, iremos recorrer á replicação do servidor, e ao consequente tratamento dos problemas que daí advêm.

2.1 Appia

O Appia[3] é uma *framework* para desenvolvimento de protocolos de comunicação por camadas, totalmente implementada em Java, que oferece abstracções que implementam inúmeros modelos de computação distribuída . No trabalho a implementar, pretende-se tomar partido das vantagens do Appia para desenvolver um sistema de comunicação em grupo, para os servidores.

3 Operações inter - componentes

3.1 Cliente - Servidor

De modo a participar no *Fault-Tolerant Pong* um cliente (representando um jogador) tem que se conectar a um servidor de jogo.

A interacção entre clientes e servidores deve, mormente, incidir sobre a capacidade de apresentar um estado de jogo ao cliente de modo coerente e eficiente (e.g. Ordenação FIFO de mensagens sobre o protocolo UDP).

Uma vez que na inicialização entre cliente e servidor, o jogador não se encontra efectivamente a jogar, considera-se este momento crucial para que existam trocas de mensagens de auxílio à tolerância a faltas.

3.2 Servidor - Servidor

A interacção entre servidores de jogo é fundamental para garantir a tolerância a faltas de um servidor: sem este tipo de interacções torna-se difícil garantir o correcto funcionamento do jogo, tendo em conta os requisitos fundamentais do enunciado proposto.

Aos servidores compete apresentar, principalmente, garantias de continuidade de serviço aos seus clientes. O problema reside em conseguir oferecer garantias em detrimento de coerência e eficiência do funcionamento do jogo.

4 Conceitos relevantes

4.1 Noção de grupo

Por grupo[1] entende-se um sistema de filiação que permite acrescentar/retirar elementos de um grupo e saber que elementos pertencem num dado momento a esse mesmo grupo (vista do grupo), bem como um sistema de comunicação que permite que todos os elementos do grupo comuniquem entre si.De modo a simplificar as interacções servidor - servidor no *Fault-Tolerant Pong* a utilização de ferramentas de comunicação em grupo é requisito essencial, nomeadamente as ferramentas oferecidas pela *framework* Appia.

4.2 Sincronia virtual

A sincronia virtual[2] garante a entrega e ordenação das mensagens e, por consequência, que todos os processos correctos têm a mesma vista sobre o grupo. Por vista compreende-se o conjunto de processos correctos que fazem parte de um grupo, sempre que exista uma alteração na filiação é entregue uma nova vista a cada elemento do grupo.

4.3 Noção de coordenador

Frequentemente, de modo a simplificar as interacções entre elementos de um grupo, é eleito um coordenador[2] do grupo ao qual poderão ser atribuídas funcionalidades exclusivas como, por exemplo: elemento primário de interacção com entidades exteriores ao grupo, poder de decisão em problemas de consenso, poder de decisão em problemas de replicação.

A utilização de um coordenador na implementação do *Fault-Tolerant Pong* está dependente do modelo arquitectural escolhido para a resolução do problema.

4.4 Ordenação de mensagens

A ordenação de mensagens assume, principalmente, duas facetas[1, Sec. 2.7]:

- Determinar *a posteriori* a ordem em que as mensagens foram enviadas de modo a assumir ou excluir relações de causa-efeito entre entre elas
- Garantir que as mensagens obedecem a critérios de ordenação estabelecidos por políticas definidas *a priori*

Existe uma série de modelos de ordenação que permitem aplicar uma ou ambas das facetas referidas: ordem FIFO, ordem causal, ordem total, etc...

Tais facetas e tais modelos devem-se reflectir na implementação do *Fault-Tolerant Pong*. É importante que os jogadores obtenham estados de jogo aproximadamente idênticos. Para tal, deve-se optar, cuidadosamente, por uma solução que permita aos clientes assimilar mensagens ordenadas em função do desenvolvimento do próprio jogo e sempre tendo em conta as noções de coerência e performance.

4.5 Modelo de replicação

Para além de garantir a continuidade de serviço o grupo de servidores tem de garantir a coerência do mesmo, e tal requisito é garantido através da replicação do estado do jogo pelos elementos (réplicas) do grupo de servidores. Existem três modelos de replicação a considerar para a resolução do problema[1, Sec. 7.6]:

1. Replicação Activa
2. Replicação Semi-Activa
3. Replicação Passiva

No modelo de replicação activa todos os servidores recebem actualizações de estado dos clientes e mantêm um estado de jogo idêntico de servidor para servidor. A disseminação de estados para o cliente é feita por todos os servidores em *simultâneo*. Cabe aos clientes, de seguida, assimilar o estado de jogo. Intuitivamente, percebe-se que a replicação activa apresenta custos de desempenho potencialmente insuportáveis ao bom funcionamento do *Fault-Tolerant Pong*.

O modelo de replicação semi-activa assemelha-se ao modelo de replicação activa pois todos os servidores continuam a receber os estados de jogo, cabendo a um dos servidores - o coordenador - actualizar as réplicas do estado correcto do jogo. Perante a falha do servidor coordenador um novo coordenador é eleito de entre os restantes elementos do grupo.

No modelo de replicação passiva cabe apenas a um servidor coordenador receber actualizações de estado vindas dos clientes; as restantes réplicas encontram-se em espera de actualizações periódicas - *checkpoints* - vindas do servidor coordenador. Perante a falha do servidor coordenador um novo coordenador é eleito de entre os restantes elementos do grupo.

5 Proposta de arquitectura

A título meramente indicativo apresenta-se de seguida uma arquitectura possível para implementar o programa *Fault-Tolerant Pong*:

- Arquitectura baseada em Coordenador e replicação passiva.

Nesta arquitectura, privilegia-se a simplicidade de implementação em detrimento de abordagens ao problema mais sofisticadas: Considera-se uma arquitectura típica cliente-servidor coordenador; o servidor coordenador encontra-se replicado; o estado do coordenador é propagado para as restantes réplicas - *checkpointing* - periodicamente.

O funcionamento do sistema baseado nesta arquitectura assenta nos seguintes pressupostos:

- Todos os clientes (no caso do *Fault-Tolerant Pong* de 4 jogadores) ligam-se ao coordenador;
- Se este falhar ligar-se-ão ao novo coordenador (eleito de entre as réplicas *vivas* no grupo de servidores);
- Durante o funcionamento do jogo o estado do mesmo é disseminado pelas réplicas, e se ocorrer uma falta no servidor coordenador em tempo de jogo, o cliente continuará a ter acesso

ao jogo enquanto existir coordenador no grupo de servidores (ou seja, enquanto existirem servidores).

Esta é uma arquitectura simples e adequada para o problema proposto, mas de um ponto de vista generalista, ela falha em não tomar partido da totalidade das réplicas (não existe balanceamento de carga entre as réplicas), e em não permitir um grande número de clientes.

5.1 Pilha protocolar

Tomando partido das caracteristicas do Appia, os autores irão desenvolver uma aplicação que assenta numa pilha protocolar, de seguida apresenta-se uma previsão dessa pilha para esta implementação.

Para a concretização do servidor, os autores planeam utilizar a seguinte pilha (sujeita a alteração futura):

PongServerLayer
VSyncLayer
LeaveLayer
StableLayer
HealLayer
InterLayer
IntraLayer
SuspectLayer
GossipOutLayer
GroupBottomLayer
TotalOrderLayer
UdpSimpleLayer

Essencialmente, esta pilha, baseada em protocolos [4] fornecidos pela *framework* Appia, oferece comunicação em grupo inter-servidor com ordenação total e sincronia virtual (todos os elementos do grupo têm a mesma vista).

Para o cliente, visto não ser membro do grupo, tem-se uma pilha mais simples:

PongClientLayer
FifoLayer
TcpCompleteLayer
UdpSimpleLayer

Repare-se que as pilhas protocolares apresentadas deverão ser complementadas por camadas adicionais que os autores considerem úteis para a resolução do enunciado proposto.

6 Trabalho futuro

Outra arquitectura possível para resolver o enunciado proposto, apesar de potencialmente mais complexa, consiste em:

- Arquitectura baseada em memória partilhada e registos atómicos

Com o objectivo de aplicar os conhecimentos adquiridos na disciplina de Tolerância a Faltas Distribuídas , pretende-se desenvolver uma arquitectura mais generalista que potencie a utilização dos recursos disponibilizados ao sistema, ou seja, uma arquitectura que fizesse proveito, activamente, do maior número de réplicas servidor possível e simultaneamente permitisse escalar o número de clientes que o sistema serve. Para concretizar esta arquitectura, a aplicação de um protocolo de balanceamento de carga entre as várias réplicas tem de ser considerado. Assim, é de ponderar a concretização de um sistema com memória partilhada entre as réplicas e com registos atómicos, de modo a que cada réplica tenha a possibilidade de servir clientes concorrentemente com as restantes, e, mesmo assim, mantendo um estado de jogo coerente entre todas as réplicas.

A não consideração, por parte dos autores, desta arquitectura como prioritária deve-se, essencialmente, a duas questões:

- Para um problema como o proposto, será necessário tanta complexidade? (nomeadamente a implementação de memória partilhada)

E, principalmente:

- Será que uma arquitectura destas apresentará níveis performance suficientes (qualidade de serviço) para satisfazer o jogo *Fault-Tolerant Pong* em *tempo-real*?

Como tal, o planeamento de implementação passa por uma concretização inicial da primeira arquitectura - mais simples - e depois, se for possível, uma arquitectura alternativa baseada na aqui descrita, sendo possível ao utilizador, no trabalho final, escolher que arquitectura usar.

7 Conclusão

A criação de um sistema distribuído permite oferecer garantias mais fortes de funcionamento que

um sistema centralizado, como por exemplo: maior disponibilidade de serviço, maior confiabilidade e, por vezes, maior performance que um sistema centralizado. No entanto, o advento dos sistemas distribuídos veio acrescentar mais complexidade ao desenvolvimento de sistemas informáticos e novos desafios aos mesmos. É neste contexto que se insere a utilidade de *frameworks* - como o Appia - e a implementação de sistemas distribuídos tolerantes a faltas - como o que aqui nos propomos a concretizar.

Referências

[1] Paulo Veríssimo, Luís Rodrigues, 2001. *Distributed Systems for System Architects*, Kluwer Academic Publishers.

[2] Rachid Guerraoui, Luís Rodrigues, 2005. *Introduction to Reliable Distributed Programming*, Preliminary Draft, Springer-Verlag.

[3] http://appia.di.fc.ul.pt

[4] Alexandre Pinto, 2005. *Appia Group Communication*