Pong a quatro, tolerante a faltas distribuídas num ambiente multi-jogadores
TFD05
23 Outubro 2005

Ângelo Costa - i29441
i29441@alunos.di.fc.ul.pt

Emanuel Teixeira - i30422
i30422@alunos.di.fc.ul.pt

Fernando Ribeiro - i30296
i30296@alunos.di.fc.ul.pt

Abstract

O Pong a quatro é um jogo simples, onde múltiplos jogadores têm como principal objectivo proteger a sua parede. Este, é jogado sobre uma arquitectura distribuída e tolerante a faltas. Sendo um sistema tolerante a faltas a aplicação ficará mais fiável e robusta, garantindo o bom funcionamento do jogo.

1 Introdução

O Pong consiste num jogo interactivo entre utilizadores. Cada utilizador estará associado a uma parede e como só temos quatro paredes, só quatro jogadores podem jogar numa só sessão[1]. Uma sessão encontra-se ligada a um servidor[2] - isto vai-nos levar a falar do conceito de arquitectura Cliente-Servidor.

Ainda sobre o jogo, um jogador irá ter completo controlo sobre um paddle[3] (ou bloco) que se pode movimentar para a esquerda e direita (ou vice-versa) ou cima e baixo (ou vice-versa) dependendo de que paddle o jogador fica. O controlo sobre o paddle pode ser através do rato ou das teclas de direcção do teclado.

Voltando ao conceito Cliente-Servidor, os jogadores são clientes de uma ou mais máquinas denominadas servidores. Com este sistema Cliente-Servidor, os clientes comunicam com um servidor usando a comunicação ponto-a-ponto. Para garantir uma comunicação sem faltas, iremos ter replicação de servidores. Os elementos do grupo de réplicas, comunicam entre si usando comunicação em grupo. ≪Se alguma falha puder acontecer, e se essa possibilidade de falha for negligenciada, então a falha irá certamente ocorrer da pior maneira e na pior altura possível.≫ A única forma de contrariar esta afirmação é considerar todas as possibilidades de falha e tomar medidas adequadas, para tal teremos que tolerar faltas. [1]

No caso particular do Pong, uma das técnicas utilizadas para tolerância a faltas como mencionado anteriormente é a replicação de servidores, no qual o sistema APPIA[4] [5] fornece os serviços necessários para desenvolver esta aplicação. Sendo assim a falha do servidor não inviabiliza todo o sistema.

O APPIA é um núcleo de protocolo que oferece uma forma limpa e elegante de expressar restrições entre canais. Esta funcionalidade é obtida como uma extensão das funcionalidades dos sistemas correntes.

O APPIA tem uma boa flexibilidade e um design modular que permite que as pilhas da comunicação sejam compostas e reconfiguradas em tempo de execução. [2]

1.1 Motivação

Sabendo que hoje em dia os sistemas distribuídos continuam em plena evolução, a exigência quanto à tolerância a faltas sobre estes sistemas é cada vez maior. Pois as faltas num sistema podem trazer vários problemas indesejáveis, desde a perca de informação (critica, e.g. bancos) até a não existência de serviços vitais para a realização de tarefas na nossa vida quotidiana.

A elevada exigência de um sistema distribuído tolerante a faltas, leva-nos a pensar de uma forma inteligente com o objectivo de resolver problemas nos quais todos dias investigadores e profissionais tentam encontrar soluções para uma melhor eficiência deste tipo de sistemas.

Isto coloca-nos num mesmo patamar de um lote limitado de investigadores que pensam em resolver este tipo de problemas, que têm por hábito uma solução simples e inovadora.

Iremos também, demonstrar que a nossa resolução, à tolerância a faltas, não será mais uma solução às já existentes, pois o nosso dever é criar soluções robustas e coerentes.

[1] Entende-se por sessão um jogo com quatro jogadores

[2] Sistema de software computacional que fornece um ou mais serviços para um outro software computacional denominado cliente

[3] Bloco que o jogador movimenta para proteger a sua parede

[4] Framework de comunicação implementado em java

1.2 Tema

O tema deste artigo é a concretização de um servidor que nos oferece um dado serviço. Esse serviço é um jogo chamado Pong a quatro, na qual quatro jogadores podem jogar entre si.

Para evitar todo o tipo de falhas, iremos replicar o servidor, de modo a tolerar faltas. Sendo assim, os jogadores podem continuar a jogar sem perda de transparência, i.e., sem se aperceber que houve uma paragem ou falha de um ou mais servidores replicados, existindo sempre pelo menos um servidor a funcionar.

1.3 Estrutura do artigo

O artigo encontra-se organizado da forma seguinte. Na secção 2, temos o Trabalho onde se vai descrever a arquitectura, a pilha protocolar e os respectivos algoritmos a implementar. Na secção 3, iremos ter a Avaliação, que se irá fazer uma breve apreciação do trabalho a desenvolver. Na secção 4 descreve o Trabalho Relacionado que se compara (sita) a trabalhos já realizados. Na secção 5 temos as Conclusões deste paper. Na última sessão (6) temos a Bibliografia, no qual se referencia documentos consultados para a elaboração deste artigo.

2 Trabalho

2.1 Descrição da arquitectura

Estamos perante um sistema que visa a comunicação entre cliente e servidor. O grupo decidiu que o sistema que vamos implementar não vai incluir coordenador, iremos ter no máximo quatro clientes e a carga do servidor não será tão grande que justifique uma divisão de tarefas entre as réplicas.

O cliente transmite para o servidor o seu estado actual (os movimentos do paddle).

O servidor recebe o estado de um jogador e transmite em modo broadcast para todas as réplicas existentes, e de seguida o servidor actualiza todos os clientes. Esta actualização do cliente implica o envio de uma mensagem ponto-a-ponto a cada cliente.

O servidor terá no máximo quatro estados, um para cada paddle.

Todos os servidores estão contidos no mesmo grupo. Cada um deles, com ajuda do Gossip, consegue obter varias informações sobre o grupo, entre elas, todos os elementos que entram e saem do grupo.

Para facilitar a comunicação entre processos, vamos utilizar o Appia que nos permite concentrar todos os esforços nos mecanismos de tolerância a faltas distribuídas.

A comunicação entre o servidor e o cliente é baseada numa pilha protocolar (explicaremos mais adiante o seu funcionamento), que nos garante a Qualidade de Serviço (QoS) necessária ao bom funcionamento do jogo.

2.2 Interface gráfica

Cada jogador tem uma interface gráfica que lhe permite visualizar e actuar sobre o jogo.

Esta interface comunica com um canal do Appia de onde recebe eventos gerados por outros jogadores e para onde envia os seus próprios eventos.

A interface gráfica é disponibilizado pelos docentes, no qual o grupo não necessita de alterar pois é adequada, para a elaboração do projecto.

2.3 Comunicação fiável em grupo

Todos os servidores que fazem parte do sistema replicado de dados pertencem a um grupo. Para a comunicação entre os elementos do grupo, serão utilizadas as camadas do Appia referentes a comunicação em grupo. [6]

O Appia oferece ainda uma funcionalidade que podemos utilizar: a sincronia virtual. [4] [7]

O modelo de sincronia virtual é adequado ao nosso sistema, facilitando em larga medida a gestão da sincronia dos estados entre os vários servidores replicados e permite ainda detectar falhas nos diversos elementos do grupo.

Na sincronia virtual, sempre que um novo servidor entra no grupo, todos os servidores recebem uma mensagem contendo uma nova vista actualizada, onde o novo servidor recebe os estados de cada cliente de um outro servidor.

Quando um servidor sai do grupo, devido a falhas de paragem (assumindo que os processos só saem porque morrem), os restantes servidores são notificados que esse elemento já não pertence ao grupo - enviando uma nova vista.

2.4 Pilhas de protocolos

Na nossa aplicação, as propriedades adoptadas da pilha protocolar são muito semelhantes às de um sistema com sincronia virtual (onde as replicas comunicam entre si através de uma pilha protocolar), onde todos os membros (correctos) de um grupo observam o grupo em forma de vistas, observam a mudança de vista pela mesma ordem, todas as mensagens enviadas durante uma determinada vista são entregues nessa mesma vista e se uma mudança de vista é necessária, todas as mensagens são entregues antes da mudança da vista.

A Pilha Protocolar terá as seguintes camadas:

ServApplLayer
TotalAbcastLayer
VSyncLayer
LeaveLayer
StableLayer
HealLayer
InterLayer
IntraLayer
SuspectLayer
MergeOutLayer
GossipOutLayer
GroupBottomLayer
UDPSimpleLayer - TCPCompleteLayer

Todas as camadas á excepção do ServApplLayer são fornecidas pelo Appia.

A camada UDPSimpleLayer irá ser usado na comunicação ponto-a-ponto entre servidor e clientes, enquanto que a camada TCPCompleteLayer será utilizada na comunicação entre as várias réplicas do grupo.

Usamos o protocolo UDP[5] porque permitimos tolerar a perca de algumas mensagens. Isto porque a diferença temporal entre as actualizações dos estados do cliente não é suficientemente grande para se perder a noção do jogo.

Iremos utilizar o protocolo TCP[6] na comunicação entre o servidor e as replicas, devido ao facto de se poder perder mensagens com actualizações de estados. Tendo em conta este facto temos que prevenir que os estados de todos os servidores se mantenham incoerentes entre si.

A nossa sugestão de pilha protocolar baseia-se numa pilha utilizada para resolver o problema de Pong a 4 jogadores usando também a plataforma Appia como suporte para a comunicação.

2.5 Tolerância a faltas

Como já foi expresso no artigo, é importante ter um sistema distribuído tolerante a faltas. Pois, caso estas não sejam tratadas iremos ter um sistema imperfeito, no qual pode ocorrer uma falha de paragem ao servidor (por diversos factores) e o cliente não deverá ficar contente, ao saber que o servidor se encontra operacional por momentos.

Para tolerar f falhas precisamos de 2f +1, maquinas [3]. Esta citação leva-nos ao conceito da replicação de servidores.

Como temos que tolerar um determinado número de faltas, vamos possuir um dado número de servidores replicados, onde irão contrabalançar todo o sistema com as possíveis faltas. Eles irão responder tal e qual como o servidor anterior, pois cada réplica terá os estados actualizados, para a qualquer momento poder assumir o controlo da comunicação com o cliente, isto tudo será abstracto para o cliente.

Como no trabalho iremos arrancar só com um servidor, teremos que ter no mínimo duas réplicas, como na citação acima, 2 * 1 + 1 = 3, temos três servidores correctos e quando ocorre uma falha temos 3 -1 = 2, dois servidores operacionais. Um destes servidores irá assumir o controlo da comunicação com os vários clientes.

2.5.1 Detector de falhas

No Pong a quatro, iremos ter um detector de falhas. Este detector tem o dever de verificar que falhas poderão existir, enquanto o ambiente se encontra em execução.

O detector de falhas vai permitir que as réplicas correctas possam ter noção de como se está a comportar o ambiente (grupo de réplicas). Caso algo de anormal aconteça, no servidor todas as réplicas estarão prontas para agir, de forma a permitir o desenrolar do jogo Pong a quatro. Tendo em conta este princípio a escolha de uma réplica para tomar o lugar do servidor é aleatória.

2.6 Coerência dos estados

O estado de um jogo é constituído por tudo o que caracteriza o jogo, isto é, a posição da bola, a posição do paddle de cada jogador e as respectivas pontuações.

Como sabemos que a comunicação numa rede de computadores tem características que condicionam a transmissão de uma dada mensagem (como por exemplo, um dado nó se encontrar sobrecarregado), temos um problema de coerência dado que, podemos ter duas ou mais mensagens a chegarem ao seu destinatário numa ordem diferente, em relação aos outros destinatários.

A solução para que este problema, encontra-se pela utilização de um protocolo de ordenação total. Assim, todas as mensagens enviadas serão entregues pela mesma ordem.

Esta característica garante-nos coerência dos estados entre processos.

3 Avaliação

A aplicação tem uma funcionalidade atractiva quanto a tolerância a faltas pois tratamos dos problemas que nos parecem importantes para viabilizar todo o jogo.

As soluções para os problemas apresentados foram analisadas de forma detalhada, de modo a que fique claro para o leitor as ideias apresentadas, podendo estes reaproveita-las em projectos futuros.

[5] User Datagram Protocol

[6] Transmission Control Protocol

4 Trabalho Relacionado

A comunicação em grupo através de sincronismo de vistas foi anteriormente estudada por José Pereira, Luís Rodrigues e Rui Oliveira. [7]

Existem também estudos nesta área efectuados por alunos desta cadeira[7]. [8]

5 Conclusões

Através deste artigo descrevemos as diversas problemáticas do jogo Pong a quatro, descrevendo uma solução possível para cada uma.

Uma das opções tomadas foi a utilização dos protocolos de comunicação em grupo já existentes no Appia, pois tratam-se de implementações mais robustas e testadas. O mecanismo de vistas facilita bastante a integração de novos elementos no jogo e por isso adequa-se perfeitamente à nossa estrutura.

Uma sugestão para trabalho futuro seria ter clientes em proximidades diferentes, em performances diferentes ao nível do tempo de latência das mensagens, permitindo dessincronizações nos estados. Tal como haver a possibilidade de ter múltiplas sessões do Pong a quatro, ou seja, permitir a existência de vários jogos a decorrer ao mesmo tempo.

5.1 Bibliografia

Referências

[1] http ://osc.di.fc.ul.pt/sii/Docs/SII-02-confiab.pdf

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

[3] Introduction to Reliable Distributed Programming, Rachid Guerraoui and Luís Rodrigues

[4] Distributed System for System Architects, Paulo Veríssimo and Luís Rodrigues, Kluwer Academic Publishers

[5] Appia Protocol Development Manual; Hugo Miranda, Alexandre Pinto, Nuno Carvalho, Luís Rodrigues, Outubro 2005

[6] Pinto, Appia Group Communication Manual, 2001

[7] José Pereira, Luís Rodrigues e Rui Oliveira, the Cost of Group Comunication with Semantic View Synchrony

[8] http://www.di.fc.ul.pt/ ler/docencia/tfd/trab.html

[7] Tolerância a faltas distribuídas