Tolerância a Faltas Distribuidas: Sistema Multi-Cliente com Servidor Replicado

Bruno Rico Jorge Coutinho Miguel Sanches

October 24, 2005

Abstract: Devido ao grande crescimento dos sistemas computacionais na actualidade, podemos observar um aumento no número, e complexidade, dos sistemas distribuídos. Estes sistemas fornecem uma variedade de serviços aos seus utilizadores e como tal convém que se mantenham funcionais mesmo que uma falha ocorra. A este tipo de mecanismos chamamos tolerância a faltas distribuídas e têm como objectivo garantir que caso um dos componentes do sistema falhe, o sistema continue a fornecer todos os serviços para os quais foi desenvolvido. Neste artigo são explorados vários aspectos de uma técnica através da qual a tolerância a faltas é garantida cujo nome é replicação.

1 Introdução

Actualmente observamos que, nos sistemas distribudos, o sistema de comunicação é extremamente linear, ou seja, um cliente envia um pedido pela rede para um dado servidor, o qual o processa e devolve o valor deste processamento de volta para o cliente. Este paradigma tem o nome de modelo Cliente-Servidor. O grande problema desta arquitectura é que caso o servidor falhe, o cliente não tem ninguém com quem comunicar, o que implica a que falha também. Num sistema distribuído tolerante a faltas isto já não acontece porque se garante que o sistema não falhe, fornecendo hipóteses, ao cliente, de enviar o pedido para um outro servidor que lhe forneça o mesmo serviço.
Utilizamos um protótipo de um jogo muito famoso no mundo dos jogos de computador obviamente modificado para incluir vários jogadores em simultâneo, para se introduzir a tolerância a faltas num sistema distribuído. O conceito do jogo e alterações efectuadas são descritos na secção 2.
A técnica que vamos apresentar neste artigo é a da replicação. Esta técnica tem como conceito principal a criação de várias instâncias do servidor, iguais, para que caso uma delas falhe, o pedido do cliente seja redireccionado para uma das outras que esteja em funcionamento para que seja resolvido. Os diferentes modelos de comunicação utilizados no sistema são explicados na secção 3, incluindo a comunicação entre as réplicas e a comunicação com os clientes. As réplicas têm de ser coerentes e a coerência pode ser mantida através de diferentes técnicas. A aproximação que vamos seguir é explicada de forma mais detalhada na secção 4.

2 Pong: O Jogo

O jogo no qual o trabalho é baseado é conhecido no mundo dos jogos de computador como Pong. As regras deste jogo são extremamente simples. O jogo é composto por um tab-

uleiro de forma quadrangular, dois jogadores e uma bola. Os jogadores são representados por dois *paddles* e têm como objectivo impedir que a bola toque na parede atrás de si. Para tal, os *paddles* possuem capacidade de movimento, controlado pelo utilizador, vertical, estando os utilizadores posicionados em campos opostos. Caso a bola toque numa parede atrás de um jogador, o outro recebe um ponto e vence o jogador que atingir um limite de pontos em primeiro lugar.
Para este trabalho foram introduzidos mais dois jogadores que irão ocupar as duas paredes do tabuleiro de jogo que se encontravam vagas. A mecânica de jogo é a mesma, ou seja, cada jogador move um *paddle* de forma a evitar que a bola que se encontra em movimento no tabuleiro toque na parede que se encontra atrás dele. Caso a bola toque na parede, os restantes três jogadores recebem um ponto. O objectivo do jogo também foi alterado, ou seja, caso a bola atinja uma parede, ao defensor dessa é atribuido um ponto. No final do jogo o vencedor é aquele que tiver menos pontos.

3 Comunicação

3.1 Cliente-Servidor

Relativamente a este mecanismo, o cliente e o servidor irão comunicar através de um canal TCP[1] para um único servidor, ou seja, iremos ter uma ligação ponto-a-ponto, de forma a utilizar todos os mecanismos oferecidos por este protocolo. Este protocolo cria um canal lógico entre o cliente e o servidor, garante a ordem FIFO[2] na entrega das mensagens e a fiabilidade na entrega das mesmas. Assim podemos manter uma lista de mensagens enviadas e caso o servidor ao qual o cliente se encontra ligado falhe, este poderá repetir o envio dos pedidos que ainda não foram processados a uma nova réplica em funcionamento.
Se o servidor falhar, o cliente irá ligar-se a uma outra réplica, como já mencionámos, através de uma lista de endereços que lhe foi fornecida aquando da sua primeira ligação. A ordem à qual ele irá efectuar a nova ligação será completamente aleatória, podendo ligar-se a qualquer servidor da lista.
Caso o servidor detecte que o cliente falhou, o jogo continua normalmente, com o *paddle* deste cliente parado enquanto que os restantes se encontrarão em funcionamento normal.
Caso seja introduzido um novo jogador num jogo que já se encontre em execução, este irá receber um estado completo da situação do jogo como por exemplo: pontuação, posições dos *paddles*, etc.

3.2 Servidor-Servidor

Entre servidores iremos utilizar a comunicação em grupo através do uso de um protocolo de comunicação por datagramas designado por UDP[3]. Este protocolo não garante ordem nem fiabilidade na entrega das mensagens que têm de ser verificados com auxílio de outros mecanismos, como por exemplo, confirmações timeouts, etc. A mais valia deste protocolo é relativamente ao facto de ser mais rápido na comunicação e permitir-nos efectuar o envio de mensagens em *multicast*. Esta última característica é o ponto fulcral na comunicação entre os servidores que entre si irão formar um grupo, para o qual se efectua o multicast.
Este grupo é mantido e gerido por uma camada a desenvolver, a ser implementada sobre

[1] Transmission Control Protocol - RFC 793

[2] First in First out - as mensagens são entregues na ordem por que foram enviadas

[3] User Datagram Protocol - RFC 768

uma plataforma já existente: o **APPIA**[4]. Esta plataforma é-nos fornecida pelos docentes para a concretização deste trabalho.

Só nos falta introduzir um conceito no sistema que é o de **coordenador**. Esta entidade não é nada mais nada menos que uma réplica que mantém um estado coerente do sistema em qualquer altura e todos os outros servidores se vão sincronizar por este e reger pelas suas decisões em caso de duvida (como por exemplo, devido a pequenas falhas de sincronismo, alguns servidores considerarem uma situação de ponto e outros não). Para evitar percas de tempo com a eleição do **coordenador**, este é automaticamente escolhido pelo seu ID, ou seja, a réplica com o ID mais baixo é o **coordenador** e se este falhar, o próximo mais baixo será eleito novo **coordenador**.

3.3 Outras considerações

Este trabalho tem as suas nuances em relação aos estados das várias réplicas: quando actualizar? Aqui, no nosso entender, encontra-se um dos pontos fulcrais deste trabalho porque se actualizarmos demasiado depressa, o estado é coerente mas o tráfego na rede torna-se demasiado grande, por outro lado, se actulizarmos demasiado devagar temos o outro lado da moeda, ou seja, poucas mensagens e réplicas demasiado incoerentes, o que pode implicar falhas na sincronia dos clientes e servidores, e um grave problema no sistema caso uma das réplicas falhe. Para tal, dividimos esta actualização em duas partes: actualizar o movimento da bola e actualizar o movimento dos *paddles*.

No primeiro caso (movimento da bola) decidimos que seria mais indicado que fosse actualizado quando esta embatesse num *paddle* ou numa parede, assim, não haveria excesso de mensagens a circular pela rede e ao mesmo tempo a sincronização de estado seria frequente o suficiente para manter todas as réplicas e clientes com estados muito semelhantes.
No segundo caso (movimento dos *paddles*), ao executar um movimento o cliente envia essa informação para o servidor ao qual se encontra ligado que por sua vez informa as outras réplicas por difusão e esta informação é enviada a todos os clientes que se encontrão a jogar. Só após esta difusão da informação de movimento do *paddle* é que o jogador que moveu o *paddle* é presenciado com o movimento do mesmo no seu ecrã.

[4]framework de comunicação por camadas implementada em Java - http://appia.di.fc.ul.pt

4 Trabalhos Relacionados

Para a execução deste trabalho foram consultados alguns artigos que à semelhança deste abordam o problema da replicação.
De este conjunto destacam-se:

Jogo Multi-Utilizador Em Tempo-Real: 4Pong
por *Fernando Felício, Susana Guedes* e *Valter Conceição*

Pong a quatro jogadores, distribuído e tolerante a faltas
por *José Mocito, Liliana Rosa* e *Nuno Almeida*

Tolerância a faltas em jogos de computador multi-utilizador
por *M. João Monteiro* e *Sandra Teixeira*

Arquitectura de um Sistema de Chamadas a Procedimentos Remotos a Servidores Replicados
por *Pedro Vicente* e *João Martins*

Appia Group Communication
por *Alexandre Pinto*

5 Conclusões

O desenvolvimento de projectos como este é de grande importância pois cada vez mais é necessario recorrer a sistemas distribuidos por motivos como segurança, capacidade de resposta e processamento, e perante este cenário torna-se necessário garantir a fiabilidade dos mesmos pois um sistema distribuido está tão ou mais exposto a falhas como qualquer outro. A fiabilidade é mantida recorrendo a métodos de replicação e sincronismo entre réplicas como os mencionados anteriormente. A grande dificuldade encontra-se em construir um modelo suficientemente rápido e estável de modo a que a replicação esteja completamente "escondida" dos utilizadores e a performance do sistema não sofra devido a todos os mecanismos inerentes.

6 Bibliografia

The Not So Short Introduction to LaTeX2
por *Tobias Oetiker, Hubert Partl, Irene Hyna* e *Elisabeth Schlegl*

Introduction to Reliable Distributed Programming
por *Rachid Guerraoui* e *Luís Rodrigues*

Distributed Systems for System Architects
por *Paulo Veríssimo* e *Luís Rodrigues*