PONG ’4’ 4:
Jogo Multi-Utilizador, distribuido e tolerante a faltas

Grupo 10

Francisco Calhanas
27905
i27905@alunos.di.fc.ul.pt

Luís Morais
29417
lfmorais@alunos.di.fc.ul.pt

José Miguel Rangel
28345
rangel.miguel@gmail.com

Abstract

Os sistemas distribuidos ganharam uma importância extrema ao longo dos últimos anos, tendo a fasquia da tolerância a faltas aumentado exponencialmente, já que as exigências do utilizador final em termos de fiabilidade e rapidez aumentaram com o desenvolvimento deste tipo de sistemas. Neste trabalho iremos tornar o ***Pong ’4’ 4*** *(lê-se pong for four), um jogo multi-utilizador baseado no Pong da Atari, num jogo multi-utilizador, distribuido e tolerante a faltas.*

1. Introdução

Nos últimos anos assistimos a uma evolução dos sistemas computacionais centralizados, em que apenas havia um interlocutor, para sistemas distribuidos, em que temos vários interlocutores que podem estar a aceder de diversos pontos. Com o tempo foi necessário melhorar a prestação dos serviços distribuidos, já que estes se tornaram insuficientes para responder aos requisitos cada vez mais estritos dos utilizadores em termos de tempo de resposta. Para tal introduziram-se conceitos como a descentralização, replicação, paralelização e equilibrio da carga computacional.

Para além da descentralização é também fundamental, hoje em dia, que alguns sistemas respondam num tempo estrito, muito bem definido pelo utilizador. Este tipo de sistemas, também conhecido como sistema de tempo-real, assenta no pressuposto que a sua computação é bem definida assim como o tempo em que esta é fornecida, tendo, para isso, em conta os diversos factores que podem influenciar esse tempo, assim como o sistema operativo, o escalonador e a rede.

O aparecimento das arquitecturas de Sistemas Distribuidos de tempo-real, ganhou uma nova área de oportunidades para novas aplicações. Mas tal como enuncia uma das leis de Murphy, ”Novos sistemas geram novos problemas” e é aí que a tolerância a faltas distribuída ganha um grande ênfase, na procura de construção de sistemas distribuidos cada vez mais robustos e fiáveis.

Pretendemos com este trabalho desenvolver o **Pong ’4’ 4**, um jogo multi-utilizador baseado no Pong da Atari, num jogo multi-utilizador, distribuido e tolerante a faltas. Pretende-se que caso um jogador ou um servidor falhe, o jogo não termine e que tenha mecanismos de recuperação dessas faltas.

Ao longo deste relatório falaremos sobre as decisões tomadas na implementação do programa. Vamos abordar os vários problemas, as várias soluções e quais as escolhas efectuadas pelo grupo para resolver o problema proposto.

2. Trabalho: Pong ´4´ 4

2.1. O jogo

O jogo é composto por um terreno de jogo quadrado, quatro paddles e uma bola. Os paddles deslocam-se lateralmente na sua parede, a bola ressalta nos paddles ou na parede. No inicio a bola encontra-se ao centro no terreno de jogo, são necessários pelo menos dois jogadores para iniciar um jogo. A pontuação inicial de cada jogador é zero, este controla um dos paddles, e tenta evitar que a bola embata na sua parede, deslocando o paddle ao longo desta. Caso a bola embata na parede de um jogador, é contabilizado um ponto para esse jogador, a bola é reposta em

jogo a partir do paddle desse jogador. O jogo termina quando algum dos jogadores tiver atingido 20 pontos, é considerado vencedor aquele que contabilize menos pontos.

2.2. Arquitectura

No ambito deste trabalho optámos por utilizar uma arquitectura descentralizada em detrimento de uma arquitectura centralizada. Optámos por esta arquitectura visto que pretendemos que jogo corra em tempo real e pretendemos também que este seja tolerante a falhas.
Numa arquitectura centralizada existe um servidor que gere todos os utilizadores. Este tipo de arquitecturas tem duas desvantagens flagrantes tendo em conta o sistema em causa. A primeira prende-se com a carga computacional do servidor neste tipo de arquitecturas, o que compromete a performance, o que no nosso sistema, que se pretende que seja de tempo-real não é de todo aceitável. A segunda desvantagem prende-se com a possibilidade de ocorrências de falhas no servidor, o que comprometeria todo o sistema uma vez que este é fundamental para o funcionamento do sistema.

Numa arquitectura distribuída o papel do servidor numa arquitectura centralizada é dividido pelos vários utilizadores, sendo da responsabilidade destes a manutenção de um estado do sistema e a computação de todos os dados possiveis antes de proceder a uma actualização com os outros participantes.

Nesta tipo de arquitectura temos que lidar com o problema da consistência, visto que o resultado da computação de cada utilizador pode (e muitas vezes é) diferente da dos restantes. Apesar de apresentar a consistência como uma desvantagem em relação a sistema centralizado, uma arquitectura descentralizada ganha em termos de performance, visto que o trabalho desenvolvido pelo servidor centralizado é distribuido por todos, e ganha também em tolerância a faltas, já que um utilizador pode falhar e todos os outros poderão continuar a sua execução sem problemas.

Para resolver o problema da consistência dos dados é necessário introduzir o conceito de *Coordenador*. O *Coordenador* é a entidade responsável por tornar os dados consistentes, ou seja, ele recebe os dados enviados pelos outros utilizadores e toma a decisão de quais os dados que todos deverão seguir, enviando-os de volta para os utilizadores finais.

Neste projecto o coordenador será sempre o cliente mais antigo do sistema, assumindo este a responsabilidade de coordenar todos os outros clientes. Quando o coordenador falha este é substituido pelo cliente mais antigo dos clientes em jogo. Isto é conseguido através da manutenção de uma lista de chegadas no servidor, quando um utilizador falha, basta atribuir as funções de coordenador ao primeiro dessa lista.

2.3. Comunicação em grupo

Um Sistema de gestão de grupos fornece duas funcionalidades aos participantes:

- A capacidade de criar ou tornar membro de um grupo;
- Fornecimento de informação actualizada sobre a disponibilidade actual dos outros membros(vista de grupo);

Um Serviço de comunicação em grupo permite a troca de informação entre membros do grupo.

Um conjunto de protocolos que ofereça o Sistema de gestão e o Serviço de comunicação em grupo é considerado uma Plataforma de Grupos.[DSSA]

Neste projecto para concretizar a comunicação será utilizada a plataforma de grupos **Appia**, que fornece a sincronia da vistas, que permite aos clientes conhecer os outros jogadores, facilita a gestão de entrada e saída de jogadores bem como a detecção de falhas dos mesmos. O Appia como plataforma de grupos facilita a troca de mensagens e permite a ordenação total das mesmas pelo que a actualização de estado dos membros do grupo é simplificada.

2.4. Mensagens

As mensagens trocadas no sistema são extremamente importantes, já que estas influenciam directamente a performance da aplicação distribuída. Numa aplicação de tempo-real é necessário garantir que todos os intervinientes partilham a mesma vista, ou seja, o mesmo estado num determinado instante. Para garantir esta propriedade é necessário garantir um tempo de latência uniforme e curto, dentro dos possível.

Para tal é necessário minimizar o tráfego na rede para evitar estrangulamentos, tanto na rede como nos utilizadores que, caso contrário, têm que lidar com demasiada informação que vai gerar atrasos na computação.

Nesta aplicação o utilizador tem apenas que enviar informação ao restante grupo em duas ocasiões distintas:

- Quando ocorre uma movimentação no seu paddle.
- Quando a bola bate na parede ou no seu paddle.

Optámos por este tipo de implementação para reduzir ao máximo e também ao essencial o tráfego na rede. Deste modo uma vista contém apenas a posição dos 4 paddles assim como a pontuação de cada jogador. A posição da bola é actualizada apenas quando esta bate numa das paredes ou no paddle, altura em que é enviada a sua posição e a sua velocidade, cabe depois a cada utilizador calcular localmente a trajectória desta. Conseguimos também através do segundo tipo de mensagens saber quando foi ou não golo já que nesta mensagem vai também a indicação se a bola bateu na parede ou num paddle.

Esta divisão provém do facto de a movimentação da bola ser uma accção determinável, visto que a trajectória da bola permanecerá igual até esta encontrar um obstáculo, e podemos assim poupar recursos de rede e de tratamento de incoerências visto que cada utilizador é capaz de efecturar os cálculos sem que ocorram erros demasiado flagrantes.

No caso das movimentações dos paddles estamos perante accções não determináveis, dai a necessidade de uma constante actualização de vistas entre os utilizadores, para mantermos assim um estado coerente partilhado por todos os utilizadores.

2.5. Faltas

”Novos sistemas geram novos problemas”

O uso de arquitecturas de sistemas distribuídos põe em evidência alguns factores que comprometem o funcionamento correcto do sistema. O facto de os intervenientes se encontrarem separados, e comunicarem por meio de uma rede é um dos factores que leva à ocorrência de faltas. Neste tipo de arquitectura existem dois tipos de faltas, assertivas e omissivas.

As faltas omissivas são caracterizadas por um componente não realizar uma interacção quando especificado para o fazer, subdividem-se em faltas temporais, omissão e paragem. Uma falta temporal ocorre quando um componente se atrasa numa acção, uma falta por omissão acontece quando uma acção é omitida, é um subconjunto das faltas temporais onde o grau de latência tende para o infinito. A falta por paragem é a mais grave das três, pois neste caso o componente pára. No nosso sistema, cada componente estima o estado do jogo a cada momento, no entanto é realizada uma troca de mensagens para actualizar o estado do jogo de tempos a tempos. Caso seja detectada uma falta por paragem, o componenete em causa é retirado do jogo. As faltas assertivas são caracterizadas pela interacção de um componente ser realizada de uma forma não especificada, podem ser classificadas como semânticas ou sintáticas.

As faltas sintáticas é caracterizada por um erro de sintaxe, por exemplo, as cordenadas da bola serem (a,b), onde são esperados algarismos. As semânticas occorem quando uma mensagem indica algo que não está de acordo com o domínio da aplicação, tomando como exemplo novamente as coordenadas da bola, estas indicarem um ponto fora dos limites do terreno de jogo. Todas as mensagens recebidas são inspeccionadas e verificada a sua validade, caso não cumpram os requisitos, são descartadas, não sendo assim os erros propagados aos outros intervenientes.

2.6. Inconsistência de Estados

Como foi mencionado anteriormente o tipo de arquitectura aqui presente padece de um problema comum a todos os sistemas distribuidos, a consistência dos dados.

No decorrer do jogo existiram algumas ocasiões em que teremos algumas inconsistências, explicaremos por isso como vamos lidar com todas elas. O responsável por ”arbitrar” as inconsistências será o *Coordenador* que tem a responsabilidade de receber as várias vistas e comunicar uma vista única a todos os outros utilizadores.

Como foi dito em no capítulo 4, teremos accções indeterináveis de utilizadores no decorrer do jogo. Neste tipo de acções enquadra-se a movimentação dos paddles. Ao movimentar o seu paddle, um utilizador(utilizador 1) envia nova posição do seu paddle, mas este utilizador(1) não sabe se entretanto um outro utilizador(utilizador 2) movimentou também o seu paddle e envia na sua vista que o outro utilizador(2) não movimentou o seu paddle quando na realidade ele o movimentou. O outro utilizador(2) também não teve noção que o primeiro(1) movimentou o paddle e por isso envia na sua vista que o primeiro(1) utilizador não efectuou qualquer movimento. Neste caso o *Coordenador* receberá duas vistas distintas, uma que indicará que apenas o primeiro utilizador(1) e outra que indicará que apenas o segundo utilizador(2) efectuou um movimento. Neste caso o *Coordenador* terá que ver a proveniência de cada uma das vistas e dar mais importância, numa vista, à informação relativa á posição do paddle do utilizador que enviou a vista. Seguindo o exemplo anterior, ele verá que a primeira

vista, pertence ao utilizador 1, quem melhor conhece a sua posição, logo na vista única final que o coordenador irá calcular a posição do paddle respectivo ao utilizador 1 será retirada da vista enviada pelo utilizador 1. O mesmo se passa para o utilizador 2. Deste modo garantimos que permanece sempre o conhecimento de quem sabe mais da situação, ou seja, nada melhor que o próprio utilizador para dizer onde está.

Nos casos em que um utilizador não envia nenhuma vista e a sua posição no paddle não coincide em duas istas distintas, o novo valor será calculado por maioria, se esta não existir, é escolhido (ao acaso) um dos valores propostos. Seguindo o exemplo supra citado, imaginemos que o utilizador 1 e o utilizador 2 enviavam valores distintos respeitando à posição do paddle do utilizador 3, e que este não enviou nenhuma vista. Neste caso, se um utilizador 4 tivesse também enviado um valor igual ao enviado pelo utilizador 1, seria esse o novo valor, mas se esse utilizador 4 também não tivesse enviado nenhuma vista, não teríamos maioria, logo seria escolhido o valor de 1 ou de 2.

Para evitar inconsistências na difusão de posição da bola, apenas um utilizador será responsável por emitir a sua posição, evitanto assim que cada vez que bola bate numa parede ou num paddle tenhamos que lidar com inconsistências, já que poderiam ocorrer atrasos na computação da sua posição ou inconsistências (mínimas mas existentes) na posição dos paddles o que criaria dados inconsistentes já que para um utilizador a bola poderia não ter batido na parede e para outro ela poderia ter batido.
O utilizador responsável por difundir a nova posição da bola, e consequentemente se foi ou não golo, é o jogador em cujo paddle ou parede a bola beteu. Desta maneira temos novamente um estado geral do sistema mais fiável e mais credivél, visto que a informação que prevalece é a informação mais ”verdadeira” visto que está mais próxima da fonte.

3. Trabalho Relacionado

A problemática da tolerância a faltas num sistema distribuido do tipo **Pong ’4’ 4** foi estudada em anos anteriores, nomeadamente por:

- José Mocito, Liliana Rosa e nuno Almeida em 2000 [4]
- Fernando Felício, Suzana Guedes e Valter Conceição em 2003 [5]

A comunicação em grupo através de sincronismo de vistas foi anteriormente estudado por José Pereira, Luís Rodrigues e Rui Oliveira. [3]

4. Conclusões

O sistema descrito implementa, ainda que apenas na teoria, um jogo multi-utilizador, tolerante a faltas e destribuído.

Tivemos em consideração os factores mais importantes e mais condicionantes no sistema e atacámos todos os problemas, tanto de inconsistência como de performance.

Descrevemos um sistema em que não existe uma entidade centralizada responsável por toda a computação, mas sim réplicas independentes e responsáveis por calcular cada uma um estado que depois será uniformizado com resurso a um coordenador escolhido de uma dessas entidades.

Minimizamos ainda ao máximo a possibilidade de existência de vistas concorrentes, e descrevemos um algoritmo que permite resolver inconsistências nas vistas.

References

[1] Rachid Guerraoui e Luís Rodrigues, ”Introduction to Reliable Distributed Programming”, Sprinter-Verlag, September 2005
[2] Paulo Veríssimo e Luís Rodrigues, ”Distributed Systems for System Architects”, Kluwer Academic
[3] José Pereira, luís Rodrigues e Rui Oliveira, ”Reducing the Cost if Group Comunication with Semantic View Synchrony”
[4] José Mocito, Liliana Rosa e nuno Almeida, ”Pong a 4 jogadores, distribuido e tolerante a faltas”, 2000
[5] Fernando Felício, Suzana Guedes e Valter Conceição, ”Jogo Multi-Utilizador Em Tempo-Real: 4Pong”, 2003
[6] http://appia.di.fc.ul.pt