Pong a quatro jogadores, distribuído e tolerante a faltas

José Mocito
i28289@alunos.di.fc.ul.pt

Liliana Rosa
i28351@alunos.di.fc.ul.pt

Nuno Almeida
i27841@alunos.di.fc.ul.pt

Grupo 07

Abstract

O Pong a quatro jogadores é um jogo simples cuja implementação é também ela simples se considerarmos que existe um servidor centralizado com o qual todos os processos comunicam. A falha no servidor origina assim a falha de todo o sistema inviabilizando a continuação do jogo. Este artigo descreve o Pong jogado a quatro e distribuído, onde cada jogador mantém uma cópia do estado do jogo e comunica com os restantes usando comunicação em grupo. Assim, a falha de um jogador não origina o fim do jogo, podendo os restantes jogadores continuar a jogar.

1 Introdução

Os sistemas tolerantes a faltas distribuídos são bastantes úteis quando é necessário garantir que quando um determinado componente falha, todos os outros se mantenham a funcionar. Existem várias técnicas desenvolvidas que pretendem solucionar este problema. Uma delas é a replicação da informação. Cada processo tem uma cópia do estado actual do sistema e por isso quando um processo falha os outros podem continuar o seu funcionamento.

Com base nesta técnica projectámos uma arquitectura para o Pong a quatro jogadores onde não é necessário a utilização de um servidor central. Cada jogador mantém uma cópia do estado do jogo e utiliza comunicação em grupo para comunicar com os restantes jogadores. Cada jogador é assim independente dos outros o que permite tolerar falhas de jogadores, podendo os restantes continuar a jogar.

Neste sistema o jogador (indivíduo) é aquele que actua sobre o jogo e o processo (*software*) é aquele que controla as entradas (movimentos do rato e pressão nas teclas do cursor) e produz as saídas (movimento do *paddle* e/ou da bola) consoante as alterações ao estado. No entanto, para simplificar, vamos considerar daqui em diante que jogador e processo referem-se ao processo que controla cada *paddle*.

Este artigo está organizado da seguinte forma. A secção 2 apresenta em resumo a motivação que conduziu à execução deste projecto. A secção 3 descreve de forma geral o Pong com quatro jogadores. A secção 4 descreve a arquitectura do nosso sistema. A secção 5 apresenta de forma sucinta a interface gráfica. A secção 6 trata dos aspectos relacionados com a comunicação fiável em grupo. A secção 7 apresenta os mecanismos de manutenção de coerência no estado. Na secção 8 são descritos os mecanismos de detecção e recuperação de falhas. Na secção 9 são apresentados alguns detalhes relativos à implementação em *software* da arquitectura descrita. A secção 10 contém uma descrição detalhada da pilha protocolar utilizada. Na secção 11 são apresentados os resultados obtidos nos testes efectuados à aplicação. A secção 12 apresenta algumas conclusões relativas à execução deste projecto. Finalmente, na secção 13 são referidos alguns aspectos interessantes de analisar em futuras apreciações do problema.

2 Motivação

Os sistemas distribuídos são hoje uma área em grande expansão nas Ciências da Computação. As suas aplicações são diversas e de grande utilidade, particularmente nos sistemas tolerantes a faltas.

Por forma a ilustrar algumas das suas potencialidades, projectámos uma arquitectura para o jogo Pong a quatro jogadores, distribuído e tolerante a faltas usando como plataforma de suporte à comunicação o **Appia** [3] [2].

Pretendemos com este projecto demonstrar por um lado as vantagens de uma aproximação distribuída ao problema e por outro a forma simples como se consegue atingir resultados bastante satisfatórios utilizando ferramentas já disponíveis.

3 Pong a quatro jogadores

O Pong tradicional é um jogo em que participam apenas dois jogadores. É jogado num campo quadrado em que dois lados opostos funcionam como balizas (e serão designados como tal daqui em diante), uma por jogador. Existe uma bola que se move no campo, fazendo ricochete quando embate nas paredes que delimitam o campo. Cada jogador

possui uma plataforma que pode deslocar lateralmente de modo a tentar evitar que a bola toque na sua baliza. A cada toque que a bola efectue numa das balizas é adicionado um ponto à pontuação do jogador adversário. No fim ganha o jogador que tiver mais pontos.

Neste artigo é considerada uma variante do Pong jogada com até quatro elementos. O ambiente do jogo é precisamente o mesmo, isto é, um campo quadrado onde cada jogador defende uma parede, que será a sua baliza. A mecânica do jogo mantém-se podendo cada jogador deslocar lateralmente a sua plataforma.

O sistema de pontuações sofre uma pequena alteração. Como se trata de um jogo com quatro jogadores não seria justo utilizar o sistema tradicional porque para cada jogador existem três adversários. Assim, sempre que a bola embate na baliza de um jogador a sua pontuação é incrementada de um ponto. No fim do jogo ganha quem tiver a menor pontuação.

Outra particularidade está relacionada com o número de jogadores que são necessários para jogar. Como já foi referido podemos ter no máximo quatro jogadores. Desta forma, basta que existam dois jogadores prontos a jogar para se iniciar o jogo.

4 Arquitectura

O facto de estarmos perante um sistema multi-utilizador e distribuído implica assumirmos que é possível ocorrer faltas, por exemplo a falha de um jogador. Para que os jogadores não afectados por falhas possam prosseguir o jogo é necessário tolerar essas faltas. Para isso utilizámos mecanismos de comunicação em grupo e de replicação de informação para garantir a independência dos processos que controlam os jogadores.

Para facilitar a tarefa de estabelecer a comunicação entre processos e assim concentrar todos os esforços nos mecanismos de tolerância a faltas distribuída utilizámos o Appia como plataforma de suporte à comunicação. Como veremos mais adiante esta escolha revelou-se muito mais útil que o que seria de esperar neste momento.

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. É da responsabilidade do canal oferecer as garantias (*QoS*) necessárias ao bom funcionamento do jogo.

5 Interface gráfica

A interface gráfica apresenta numa janela o campo de jogo com os jogadores e a bola. Cada jogador tem a possibilidade de actuar sobre o jogo movendo a respectiva plataforma com o rato ou cursor do teclado. Essas movimentações são comunicadas aos restantes jogadores através do canal Appia constituído para o efeito. Da mesma forma a interface gráfica recebe do canal as movimentações efectuadas pelos adversários possibilitando assim a actualização do estado do jogo em cada jogador.

Como vamos ver mais à frente, a comunicação fiável em grupo vai permitir à aplicação garantir que quando envia uma mensagem a informar que pretende alterar o estado todos os processos correctos vão recebê-la e pode por isso alterar também o estado na interface.

6 Comunicação fiável em grupo

Para a comunicação em grupo surgiram-nos duas hipóteses possíveis de concretização. A primeira, baseada em protocolos de difusão fiável, nomeadamente no *Protocolo de Difusão Fiável Uniforme* descrito e implementado em [4]. Este mecanismo evita a utilização de um servidor para auxiliar a integração ou exclusão de processos no grupo, no entanto exige que cada processo saiba à partida em que endereços os outros jogadores vão estar.

A segunda hipótese baseia-se no conceito de sincronismo virtual [1], e utiliza os protocolos já existentes no Appia para suporte à comunicação em grupo [5]. O modelo de *vistas* é adequado ao nosso sistema, facilitando em larga medida a gestão da sincronia dos estados entre os vários jogadores. No entanto, o facto de no Appia a implementação destes protocolos depender de um servidor que dá a conhecer os processos uns ao outros, introduz-se assim mais um possível ponto de falha.

Optámos pela 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.

7 Coerência do estado

É importante neste momento apresentarmos a definição correcta de **estado do jogo**. Assim, a cada instante o estado do jogo é constituído pela posição da bola, a posição do *paddle* de cada jogador e as respectivas pontuações.

Dado que estamos a falar de um sistema que vai funcionar sobre uma rede de computadores onde as características das linhas de comunicação são potencialmente diferentes é normal as mensagens não chegarem a todos os destinatários ao mesmo tempo. Temos assim um problema de coerência dado que é possível duas mensagens chegarem a dois destinatários por ordens diferentes.

A solução encontrada passa pela utilização de um protocolo de ordenação total. Assim, todas as mensagens enviadas no grupo são entregues a todos os processos pela

mesma ordem. Esta característica aliada ao facto de estarmos a trabalhar sobre sincronia virtual oferece a garantia de coerência do estado relativamente às mensagens trocadas entre os processos e que correspondem essencialmente a mensagens que sinalizam movimentos de *paddles*.

Mesmo com esta garantia existe ainda o problema da movimentação da bola. Se a ordem de movimentação for local, isto é, se cada processo for responsável por movimentar a bola, é normal existirem incoerências entre os processos resultantes da heterogeneidade das máquinas utilizadas, das diferenças de relógios entre os processos e das diferenças de momentos em que se dá início a movimentação da bola.

O protocolo de ordenação total das mensagens surge novamente como parte da solução deste problema. Necessitamos também de introduzir um novo conceito, o de **coordenador do jogo**. Este papel é desempenhado por um qualquer jogador eleito de forma determinística por todos os jogadores com base na *vista* actual. O coordenador desempenha várias tarefas que serão apresentadas mais tarde. No contexto actual interessa apenas saber que é ele quem determina quando a bola deve ser movimentada e comunica essa decisão a todo o grupo. É nesta fase que a ordem total de mensagens tem extrema importância. Como o movimento da bola é determinado por uma mensagem, esta acção é ordenada de igual forma em relação a todas as outras possíveis acções (movimentos dos *paddles*) em todos os processos. Com este mecanismo conseguimos que todos os processos estejam coerentes relativamente ao estado actual do jogo.

Falta apenas referir que não houve necessidade de concretizar o protocolo de ordenação total de mensagens visto que o Appia já fornece três protocolos que utilizam diferentes algoritmos de ordenação e que servem na perfeição ao nosso sistema.

8 Tolerância a faltas

É possível que ocorram falhas nos processos podendo estes ficar impedidos de jogar. Como cada processo detém o estado actual do jogo não existe uma relação de dependência entre os intervenientes, por isso caso algum processo falhe, os outros continuaram o seu normal funcionamento. O mecanismo de detecção de falhas descrito mais à frente permite que os processos tenham conhecimento das falhas de outros processos e possam agir por forma a permitir o desenrolar normal do jogo, nomeadamente na não contagem de pontos relativos aos processos que falharam.

8.1 Detecção de falhas

Como já foi referido, após a falha de um jogador é útil que os restantes jogadores tomem conhecimento dessa falha para poderem descartar os pontos que seriam atribuídos ao jogador que falhou quando a bola embatesse na sua parede. Para isso utilizamos o conceito de *vista* para detectar jogadores que tenham abandonado o jogo. Assim, sempre que é detectada uma nova *vista*, cada processo compara os membros actuais com os da *vista* anterior e determina os jogadores que *morreram* agindo posteriormente de forma a actualizar o seu estado e deixando de contabilizar pontuações para esse jogador.

8.2 Recuperação de falhas

Quando um processo se encontra em condições de reintegrar o jogo tenta juntar-se novamente ao grupo. As camadas de comunicação em grupo encarregam-se de proceder à actualização da *vista* e assim todos os processos que pertencem ao grupo passam a ter conhecimento do novo processo. Neste momento uma de duas situações pode ocorrer: ou o grupo tem menos de quatro elementos e por isso o novo processo passa a fazer parte do jogo, ou já existem quatro jogadores e então o processo fica numa fila aguardando uma possibilidade de reintegrar o jogo.

No contexto deste artigo assumimos que quando um processo falha isso implica a sua extinção, isto é, considera-se um processo *morto*. Desta forma, quando se dá a reintegração desse processo ele não contém qualquer informação do estado e comporta-se por isso como um jogador novo.

8.2.1 Integração de um processo num jogo a decorrer

Se existirem menos de quatro jogadores em jogo, um processo pode integrar de imediato o jogo. Assim, quando o coordenador recebe uma nova *vista* e detecta o aparecimento de um novo jogador envia uma mensagem para o grupo contendo o estado actual do jogo. Só quando todos os elementos do grupo receberem esta mensagem é que é possível retomar o jogo, isto é, trocar mensagens entre eles. Desta forma garantimos que o estado recebido pelo novo processo é actual e fica coerente relativamente aos outros processos e portanto pode começar a participar no jogo.

De notar que não foram feitas considerações especiais relativamente ao facto de ser ou não o jogador falhado a retomar o lugar vago, isto é, um processo novo integra o jogo assumindo a pontuação de um qualquer jogador que tenha falhado anteriormente, ficando assim com a sua pontuação. Este comportamento pretende apenas simplificar o processo de integração. Seria certamente possível uma outra solução, mais complexa mas também mais justa.

8.2.2 Fila de espera

Quando um jogador tentar integrar o jogo e já existem pelo menos quatro jogadores este vai para uma fila de espera. Sempre que um processo que está integrado no jogo falha, o

primeiro processo na fila de espera toma o seu lugar, usando o algoritmo de integração já descrito.

9 Alguns detalhes de implementação

As características com maior relevância na nossa arquitectura foram já descritas atrás de forma genérica, sem que existisse na maioria das vezes a preocupação de explicitar a forma como foram implementadas em *software*.

Apresentamos nas próximas subsecções alguns detalhes importantes na implementação do nosso sistema que ainda não foram referidos e que deverão ser encarados como possíveis soluções dos subproblemas que pretendem resolver.

9.1 Eleição do coordenador

Para determinar o coordenador auxilia-mo-nos na ordenação dos elementos do grupo na *vista* actual, dado que esta ordem é igual para todos os processos integrados no grupo.

O algoritmo de eleição do coordenador é extremamente simples e consiste básicamente em três princípios:

- Se um processo for um novo jogador e for o primeiro elemento na *vista* assume-se como coordenador. Se receber uma mensagem a sinalizar o estado actual do jogo, então actualiza o seu estado e descarta os estatutos de coordenador e de novo jogador.
- Um processo que seja coordenador numa *vista*, se continuar correcto na *vista* seguinte, mantém o estatuto de coordenador.
- Se o processo coordenador de uma determinada *vista* morrer, na *vista* seguinte o primeiro processo na tabela de jogadores activos que contenha estado assume a posição de coordenador.

9.2 Atribuição automática dos *paddles*

Mais uma vez, utilizamos a ordenação dos elementos do grupo já fornecida pelo mecanismo de sincronia virtual do Appia para auxiliar a atribuição dos *paddles*.

Cada processo mantém uma tabela com os jogadores activos no jogo. Inicialmente, a ordenação desta tabela corresponde à ordem pela qual os elementos aparecem na *vista* e o *paddle* é atribuído com base nessa mesma ordem.

Quando se dá início ao jogo a atribuição dos *paddles* é fixada e mantém-se invariável até que um processo falhe. Quando isso acontece o lugar ocupado por esse processo fica à disposição. Quando um processo pretende integrar o jogo, os processos que já se encontravam a jogar determinam o primeiro espaço livre na tabela de jogadores activos e atribuem esse espaço ao novo processo. O processo coordenador envia então para o grupo o estado já actualizado relativamente ao novo jogador e este quando recebe essa mensagem actualiza o seu estado local e determina assim o seu *paddle* com base na posição que ocupa na tabela.

9.3 Movimentação da bola

Inicialmente tinhamos previsto utilizar uma funcionalidade do Appia que permitia criar um temporizador que é lançado no canal de forma periódica e que pode ser recebido por qualquer camada.

Numa primeira implementação todos os processos criavam um temporizador mas apenas o coordenador lhe atribuia relevância e o utilizava para determinar o momento em que disseminava a ordem de movimentação da bola.

Posteriormente introduzimos uma optimização onde apenas o coordenador inicializava o temporizador e sempre que havia uma mudança de vista esse temporizador era desligado e reinicializado novamente pelo novo coordenador.

Em teoria esta solução parece adequada, no entanto na prática existem alguns problemas. Em primeiro lugar, devido à forma como os temporizadores são ligados e desligados, quando o coordenador desligava o temporizador ainda recebia posteriormente alguns eventos originados por este, ou seja, o efeito de desligar não era imediato. Por outro lado, quando existiam falhas de processos, a frequência com que surgiam os eventos gerados pelo temporizador e o facto de este evento não passar pela camada de ordenação total de mensagens, levava a que por vezes os processos não chegassem a receber o evento que sinaliza a mudança de *vista* e por isso ficávamos com uma situação de interbloqueio.

Por estas duas razões optámos por não utilizar este tipo de temporizador e utilizar um mais comum, fornecido pela classe *javax.swing.Timer*. O funcionamento é em tudo idêntico, mas como este evento não passa pelo canal do Appia não cria a situação de interbloqueio descrita anteriormente.

10 Pilha protocolar

No Appia as propriedades suportadas pelo canal de comunicação são definidas com base nos protocolos utilizados e denomina-se **Qualidade de Serviço** (*QoS*).

No nosso sistema definimos uma pilha protocolar que tem as propriedades habituais de um sistema com sincronismo virtual:

- Todos os membros de um grupo observam o seu grupo sobre a forma de *vistas*.
- Todos os membros de um grupo observam a mudança de *vista* pela mesma ordem. Existe uma ordem total das *vistas*.

- Todas as mensagens enviadas durante uma determinada *vista* são entregues nessa mesma *vista*.
- Se uma mudança de *vista* é necessária, todas as mensagens são entregues antes da mudança da *vista*.

Outra propriedade importante é a oferecida pela ordenação total de mensagens que, como já foi referido, garante que as mensagens são entregues pela mesma ordem em todos os processos intervenientes.

Com estas propriedades garantimos que os estados dos vários processos são sempre actuais e sincronizados, e garantimos também a tolerância a faltas provocadas por falhas de processos assim como também a sua reintegração no jogo.

A pilha protocolar tem portante o seguinte aspecto:

ApplSupportLayer
TotalAbcastLayer
VSyncLayer
StableLayer
HealLayer
InterLayer
IntraLayer
SuspectLayer
MergeOutLayer
GossipOutLayer
GroupBottomLayer
TCPCompleteLayer

A camada primeira camada da pilha, **TCPCompleteLayer**, é responsável pela comunicação real entre os processos usando o protocolo TCP. As nove camadas acima desta executam diversas tarefas relacionadas com a comunicação em grupo e são elas que actuando em conjunto oferecem as garantias habituais de um sistema de comunicação com sincronismo virtual. A descrição detalhada do que cada uma destas camadas faz pode ser encontrada em [5]. A camada **TotalAbcastLayer** oferece a garantia de ordem total na entrega de mensagens à camada imediatamente acima, denominada **ApplSupportLayer** que corresponde à camada de suporte à interface do jogo e consiste na implementação da grande maioria das funcionalidades descritas neste artigo.

11 Resultados

A infra-estrutura utilizada para realizar os testes à aplicação consistiu num rede de diversos computadores, potencialmente diferentes no que se refere a capacidade de processamento e equipamento de comunicação, ligados por uma rede local (*LAN*) *ethernet*.

Foram concebidos diversos cenários com as mais variadas combinações de falhas de processos que nos foi possível realizar. Apresentamos apenas alguns cenários a título de exemplo:

- Falha de apenas um processo
- Falha de múltiplos processos
- Falha de múltiplos processos com consequente integração de elementos em fila de espera
- Falha de todos os processos em actividade no jogo e consequente integração de elementos em fila de espera

Qualquer um destes cenários foi bem sucedido no que se refere ao comportamento desejado.

Outros resultados interessantes prendem-se com o desempenho do sistema relativamente ao normal desenrolar do jogo. À medida que adicionamos jogadores ao jogo nota-se uma degradação do desempenho resultante do aumento do número de mensagens trocadas entre todos os elementos. A utilização do cursor em detrimento do rato para controlar os *paddles* ajuda a atenuar os efeitos referidos pois são geradas bastante menos mensagens.

12 Conclusões

A versão do Pong descrita neste documento consegue de forma simples enfrentar problemas inerentes à distribuição como a falha de jogadores e os efeitos das trocas de mensagens numa rede na coerência do estado jogo entre os vários jogadores.

Para a simplicidade da solução muito contribuiu a utilização da plataforma de composição de protocolos Appia que facilitou bastante a tarefa visto que oferece grande parte das abstracções necessárias à concretização da aplicação.

Parece-nos no entanto que os aspectos menos positivos relativos ao desempenho podiam possivelmente ter sido debelados caso tivessemos optado por desenvolver mecanismos mais específicos ao nosso problema em detrimento da utilização de uma plataforma mais genérica como é o Appia.

13 Trabalho Futuro

O facto de existir um coordenador que se responsabiliza por comunicar o movimento da bola a todos os elementos do grupo de comunicação leva-nos a pensar que em ambientes onde a comunicação seja bastante dispendiosa em termos de desempenho esta poderá não ser a melhor solução. Seria por isso interessante de futuro conceber uma forma de tornar local a cada processo a decisão de movimentação da bola, sincronizando de forma periódica o estado, na

esperança de reduzir considerávelmente o número de mensagens a serem trocadas.

Não foram feitas considerações essenciais no que se refere ao tratamento de falhas que não levassem ao cancelamento do processo falhado, como é o caso de atrasos exagerados nas comunicações e partições na rede. Estes casos seriam fácilmente resolvidos com pequenas alterações à abordagem escolhida neste artigo.

Existe também espaço para progredir noutras direcções que não as directamente relacionadas com a tolerância a faltas, como é o facto de os jogadores integrarem o jogo sem que existe uma forma de autenticação, permitindo outros jogadores ocuparem o lugar de jogadores que falharam, e consequentemente estes terem de ficar de fora quando recuperarem e pretenderem reintegrar o jogo.

Referências

[1] K. Birman. Virtual synchrony model. Technical report, Cornell University, July 1993.

[2] L. R. Hugo Miranda, Alexandre Pingo. Application program interface specification of *Appia* version 1.2. Technical report, Universidade de Lisboa, July 2001.

[3] L. R. Hugo Miranda, Alexandre Pinto. *Appia*, a flexible protocol kernel supporting multiple coordinated channels. In *Proceedings of The 21st International Conference on Distributed Computing Systems (IDCS-21)*, page page to appear, Phoenix, Arizona, USA, April16-19 2001. IEEE Computer Society.

[4] L. R. Nuno Carvalho. *Implementing Reliable Broadcast Protocols in* Appia, November 2003.

[5] A. Pinto. *Appia Group Communication Manual*, February 2001.