Pong a quatro jogadores, distribuido e tolerante a faltas

José Mocito Liliana Rosa Nuno Almeida
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 distruibuído, onde cada jogador mantém uma cópia 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 continuarem 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 resolver 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 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 é aquele que actua sobre o jogo e o processo é aquele que controla as entradas e produz as saídas 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 jogador.

Este artigo está organizado da seguinte forma. A secção 2 expõe algumas razões que nos levaram a realizar este 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 descreve o mecanismo de tolerância a faltas. Finalmente, a secção 6 apresenta a forma como são recuperadas as falhas.

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 pontencialidades, 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].

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 é possível 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.

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.

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.

4.1 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. 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.

4.2 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 [2]. 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 comunciação em grupo [4]. O modelo de *vistas* é adequado ao nosso sistema, facilitanto 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 detecta alterações às *vistas*, introduz-se assim mais um possível ponto de falha.

Optámos então pela utilização dos protocolos de comunicação em grupo já existentes no *Appia* pois tratam-se de implementações mais robustas e testadas, e que se adequam perfeitamente ao nosso sistema.

4.3 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 sicronismo virtual:

- Todos os membros de um grupo vêm o seu grupo sobre a forma de *vistas*.
- Todos os membros de um grupo vêm 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*.

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
VSyncLayer
StableLayer
HealLayer
InterLayer
IntraLayer
SuspectLayer
MergeOutLayer
GossipOutLayer
GroupBottomLayer
TCPCompleteLayer

A descrição do que cada uma das camadas faz pode ser encontrada em [4].

4.4 Inicio do jogo

O jogo inicia-se quando quatro jogadores estiverem disponíveis a jogar. Assim, para que o jogo começe, cada um dos processos envia para o grupo a sua intenção de jogar. Quando todos os processos tiverem recebido as mensagens de todos os outros dá-se inicio ao jogo.

4.5 Detecção de falhas

Como cada processo mantém o seu próprio estado do jogo quando um processo falha os outros podem continuar a jogar mesmo sem terem conhecimento dessa falha. É no entanto útil que os restantes jogadores tomem conhecimento dessa falha para poderem descartar os pontos que seriam atribuidos ao jogador que falhou quando a bola embatesse na sua parede. O protocolo *suspect* implementado na camada *SuspectLayer* resolve este problema simulando um detector de falhas. Quando se dá uma falha de um processo este comunica à camada de suporte à aplicação a ocorrência e esta por sua vez trata dos pormenores da tolerância à falta.

4.6 Camada de suporte à aplicação

Para gerir os eventos criados no decorrer do jogo e as falhas e posterior reintegração de jogadores, uma outra camada é necessária. Este protocolo, por nós concretizado, está assim dividido em três partes. À primeira compete apenas inicializar o canal.

A segunda, relacionada com o decorrer normal do jogo, processa dados relacionados com as acções efectuadas por cada jogador, movimentação da bola, atribuição dos pontos, entre outros, comunicando com a aplicação para informar de alterações ao estado do jogo produzidas pelos adversários e recebendo acções da interface que propaga pelo canal por forma a chegarem aos outros jogadores.

A última parte é responsável por gerir as falhas de jogadores, impedindo a atribuição de pontos a jogadores que falharam, e também pela reintegração desses mesmos jogadores, para que estes possam tomar conhecimento do estado actual do jogo: da sua pontuação e dos seus adversários, e da posição actual da bola e da sua plataforma.

5 Tolerância às faltas

É possível que ocorram falhas nos processos podendo estes ficar impedidos de jogar. Neste caso, como já foi referido, a camada de detecção de falhas vai identificar essa falha e na camada de suporte à aplicação são tomadas providências para que seja tolerada a falta e não sejam contados os pontos que resultariam da bola embater numa parede defendida por um processo que não se encontra correcto.

6 Recuperação de falhas

Quando um processo se encontra em condições de reintegrar o jogo comunica com o servidor de actualização de *vistas* (*GossipServer*). Esta comunicação vai originar um evento no grupo que vai obrigar à junção das duas vistas, a do grupo actual e a do processo que quer reintegrar o grupo.

Quando o processo volta a fazer parte da vista envia um pedido de actualização do estado para o coordenador do grupo, que lhe retornará o estado actual do jogo. A partir deste momento o processo passa a fazer novamente parte do jogo.

No caso de não haver nenhum processo correcto no momento em que um processo prente reintegrar o jogo este aguarda que um outro processo efectue um procedimento de reintegração, pelo que neste caso, como todos os processos têm um estado igual ao inicial, o jogo começa de novo, ou seja, as pontuações são retomadas do zero.

Referências

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

[2] N. Carvalho. *Tutorial for Reliable Broadcast Protocols*, October 2003.

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