Pong Distribuído

Tolerância a Faltas Distribuída

Luís Monge nº 28953 i28953@alunos.di.fc.ul.pt
João Alves nº 28014 i28014@alunos.di.fc.ul.pt
João Santos nº 26553 i26553@alunos.di.fc.ul.pt

Grupo: tfd002

20 de Outubro de 2003

Resumo

Este artigo debruça-se sobre a concepção e concretização do Pong (para um máximo de 4 jogadores), adaptado para um Sistema Distribuído com Tolerância a Faltas. O sistema assenta na plataforma de protocolos de comunicação em grupo **Appia**[1].

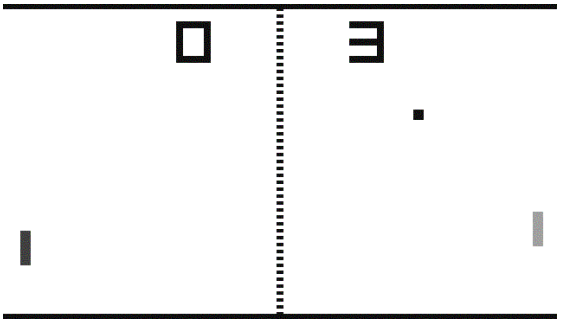

Figura 1: as primeiras versões do Pong.

1 Introdução

Em 1951, *Ralph Baer* criou o Pong, um dos primeiros jogos de computador e que se viria a tornar um clássico (ver Figura 1). O objectivo do jogo é o do jogador defender a sua baliza com uma barra. Este artigo descreve uma possível concretização para o jogo num sistema distribuído com tolerância a faltas. Isto quer dizer que vários jogadores poderão jogar simultaneamente, em locais diferentes do globo, através da internet. Quer dizer também que jogadores poderão entrar, sair ou *crashar* e os restantes continuarem a jogar. Por fim, significa também, que os jogadores poderão jogar de forma fiável e transparente, isto é, os mecanismos para que o jogo funcione de forma correcta serão invisíveis ao utilizador.

[1] `http://appia.di.fc.ul.pt`

2 Vista Geral

Ao conjunto de jogadores que vão estar a jogar ao mesmo tempo chama-se **grupo**. De instantes em instantes, o Appia tira "fotografias"ao estado do grupo, a que deram o nome de *vistas*. À troca de informação entre os vários participantes de um grupo dá-se o nome de **comunicação em grupo**. Para termos comunicação em grupo e termos um Pong realmente distribuído e tolerante a faltas é necessário termos servidores replicados, isto é, cada jogador tem um servidor próprio com o estado do jogo (ver Figura 2). Por isso, desde logo nos surgiu a dúvida de como iríamos sincronizar os vários servidores, tendo em conta que todos têm de ter o mesmo estado ao mesmo tempo, ou seja, a bola terá de estar na mesma posição ao mesmo tempo, em todos os participantes. Ora, este problema não é sim-

ples, pois todos os participantes têm de acordar numa posição comum entre todos. Este é um problema típico dos sistemas distribuídos: problema do consenso[1]. Para resolvermos este problema, e sabendo que o Appia nos oferece um detector de falhas, tomámos a decisão de termos um processo *coordenador* num dos servidores replicados. Por vezes, para resolvermos problemas distribuídos, temos de nos basear no paradigma centralizado. E foi o que fizemos. Este coordenador vai ser responsável, não só por disseminar a posição da bola em cada instante, mas também por receber a posição das barras dos outros e disseminá-las pelos restantes. Posto isto, o problema que falta resolver é o do coordenador sair do jogo ou *crashar*. Caso saía de forma graciosa, o coordenador avisa os restantes que vai sair e que é necessário que outro participante tome o lugar dele. Se não houverem mais participantes, o jogo termina. Se houver só um participante, este toma o lugar do coordenador. Se houverem mais que um participantes, tem de se proceder a uma votação e a um consenso para eleger um novo coordenador. Caso o coordenador tenha um *crash*, os restantes irão detectar a ausência do coordenador (por exemplo, deixaram de receber a posição da bola durante um certo tempo) e, novamente, será lançada um votação para eleger um novo coordenador.

Como é sabido, o Appia permite utilizar pilhas de protocolos[2] que se podem ir adicionando um a um ao topo da pilha. A escolha acertada e justificada dos protocolos que se vão utilizar é talvez das partes mais importantes deste projecto.

Os protocolos que nós considerámos estritamente necessários para resolver este problema são: *Group Communication support infrastructure*, *TCP Simple* e *TotalAbcast*. Dentro do *Group Communication support infrastructure* vamos utilizar a totalidade dos protocolos oferecidos: *bottom*, *suspect*, *sync*, *stable*, *intra*, *inter*, *heal*, *leave*.

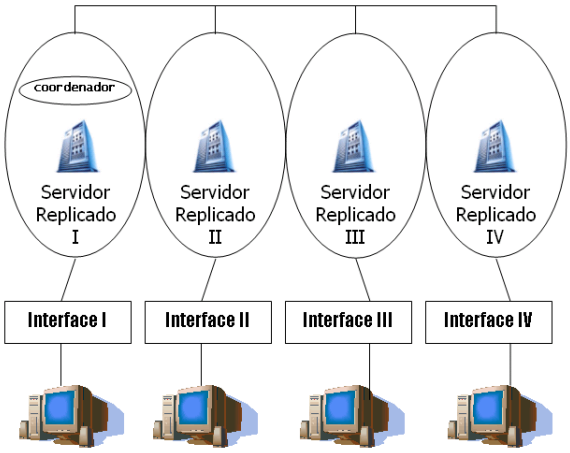

Figura 2: esquema do Pong Distribuído.

Na secção seguinte (secção Componentes) iremos justificar a opção pelos protocolos que escolhemos.

3 Componentes

3.1 Comunicação em grupo

Como já foi salientado anteriormente, iremos suportar a comunicação em grupo através do protocolo *Group Communication support infrastructure* do Appia. Este protocolo reúne uma série de 8 protocolos que consideramos, na sua totalidade, indispensáveis para uma boa realização do trabalho:

- **bottom** - este é protocolo é sempre necessário, pois serve para comunicar entre camadas.
- **suspect** - este protocolo implementa um detector de falhas, que iremos necessitar para detectar a falha de jogadores.
- **sync** - este protocolo assegura que todos os participantes do grupo receberam todas as mensagens antes de a vista[3] mudar.

[2] `http://appia.di.fc.ul.pt/dist/index.html`

[3] vista é o estado actual do grupo

- **stable** - este protocolo é bastante importante, pois garante a distribuição de todas as mensagens por todos os participantes vivos do grupo.
- **intra** - é o protocolo responsável pela correcta manutenção das vistas.
- **inter** - este protocolo é necessário aquando da partição do grupo, pois permite fazer a fusão de duas vistas sobre o mesmo grupo.
- **heal** - este protocolo permite detectar e resolver a partição de um grupo.
- **leave** - este protocolo é necessário quando pretendemos retirar um jogador do jogo.

3.2 Entrega das Mensagens

Para um correcto funcionamento do jogo, a entrega das mensagens terá de respeitar duas propriedades que, de seguida, enunciamos, apresentando as suas respectivas justificações:

Ordem FIFO [1] - Esta propriedade garante que todas as mensagens enviadas pelo coordenador ao grupo são entregues pela mesma ordem. No caso do Pong distribuído, o coordenador envia periodicamente a posição da bola, sendo necessário garantir a ordem FIFO, de modo a evitar situações irregulares como esta: a bola desloca-se numa certa direcção e sentido; se a propriedade em causa não for garantida, pode acontecer que, estando a bola numa posição *x*, recebamos a posição *x + 2* antes da posição *x + 1*, o que visualmente dá a ideia de que a bola muda sentido, sem ter batido em nenhuma superfície.

Ordem Total [1] - Esta propriedade garante que todas as mensagens são entregues ao grupo pela mesma ordem. É necessário garantir esta propriedade de modo a assegurar a justeza entre os jogadores. Pretende-se salvaguardar que, se um dos jogadores tiver um atraso na recepção/processamento das mensagens (em relação aos outros), os restantes não prossigam o jogo com um estado incoerente.

Como vamos utilizar o protocolo *TCP Simple* do Appia (ver protocolo TCP[4]), não necessitamos de usar o protocolo FIFO, pois o TCP já garante ordem FIFO[5]. Achamos que esta opção de utilizar *TCP Simple* é mais vantajosa do que utilizar *UDP Simple* (ver protocolo UDP[6]) em conjunto com o protocolo de ordenação FIFO pois, para além da vantagem de usarmos menos um protocolo do Appia, podemos desfrutar de todas as vantagens inerentes ao TCP, ou seja, e destacando apenas algumas: fiabilidade, correcção de erros, fragmentação e controlo de fluxo.

3.3 Entrada de um jogador

Em relação à entrada de um novo jogador, resolvemos adicionar uma restrição: um jogador só pode entrar quando fôr marcado um golo, aproveitando assim o facto de o jogo estar parado. Adicionámos esta restrição porque a entrada de um novo jogador é um processo moroso, ou seja, iria atrasar, ou mesmo parar, o jogo dos restantes jogadores. Para além disso, existe também um critério de justeza, que é o do jogador ao entrar, poder ser supreendido com a bola muito próxima da sua área de acção. Na pior situação, o jogador que entra, não tem tempo para avaliar a situação e reagir adequadamente e poderá acabar por sofrer um golo.

[4] RFC793 - `http://www.ietf.org/rfc/rfc0793.txt?number=793`

[5] RFC675 - `http://www.ietf.org/rfc/rfc0675.txt?number=675`

[6] RFC768 - `http://www.ietf.org/rfc/rfc0768.txt?number=768`

3.4 Saída de um jogador

Nesta secção, temos de fazer a distinção entre dois tipos de saída: **saída voluntária** - o jogador pretende sair por vontade própria do jogo; **saída por falha** - o jogador parou de responder (ou por *crash* ou por outra falta qualquer).

Na primeira situação, o jogador sinaliza o coordenador que não pretende continuar no jogo. O coordenador retira o jogador da vista de jogadores e actualiza o estado do jogo. Na segunda situação, o coordenador detecta que o jogador deixou de responder (o jogador deixou de enviar o seu estado) e avisa esse jogador que já não está em jogo. Optámos por esta solução (de enviar uma mensagem ao jogador que deixou de responder) porque o jogador pode não ter *crashado*, mas apenas estar atrasado no envio/recepção das mensagens. O envio desta mensagem avisa o jogador faltoso de que já não faz parte do jogo, logo podemos apresentar uma mensagem a avisar o utilizador de tal situação (em vez do jogador ficar eternamente à espera da actualização do estado do jogo) e, ao mesmo tempo, evitamos que o jogador reenvie mensagens e permitimos que este possa sair do programa de forma graciosa. No caso de ocorrer mesmo *crash* do jogador, não há nada a fazer do seu lado, apenas o coordenador tem de actualizar o novo estado do jogo e a vista do grupo, da mesma forma que o fazia nas situações anteriores.

4 Conclusão

Esperamos que este artigo tenha sido bastante auto-explicativo, porque foi essa a nossa intenção. Tentámos focar todos os pontos essenciais do nosso Pong Distribuído mas, com a construção do primeiro protótipo, vamos poder aprofundar e corrigir alguns temas e vamos concerteza ter oportunidade de avaliar o nosso jogo.

Como foi possível constatar durante a leitura deste artigo, o nosso Pong Distribuído vai estar muito dirigido para a tolerância a faltas distribuída e para a fiabilidade, o que, provavelmente, irá ter consequências no desempenho do jogo. Mas pensamos que, dentro das motivações para este projecto, essa opção tomada foi a mais adequada.

Referências

[1] Veríssimo, P. and Rodrigues, L. (2001). *Distributed Systems for System Architects*. Kluwer Academic Publishers

[2] Postel, J. (1980). *RFC 768 - User Datagram Protocol*. The Internet Engineering Task Force (IETF) - `http://www.ietf.org/`

[3] Information Sciences Institute (1981). *RFC 793 - Transmission Control Protocol*. University of Southern California