Jogo Pong: Tolerância a faltas com a plataforma *Appia*

Grupo 008

João Antunes 26541 | Ricardo Santos 26405 | Tiago Mendo 28015

Sumário

Cada vez mais a informática é baseada em sistemas distribuídos, dispersos geograficamente e assentes sobre vários níveis de fiabilidade. Com o crescente importância das aplicações distribuídas surgem problemas como garantir que uma falha não prejudique o sistema. Transpondo esse problema para o universo dos jogos, apresentamos uma solução para a criação de um jogo distribuído que seja tolerante a faltas.

A utilização do Appia *possibilita a criação de uma aplicação distribuída (*Pong*) através de pilhas protoculares que fornecerão um detector de falhas e a esperada tolerância a faltas. Pela sua simplicidade, o jogo pode ser implementado em dispositivos móveis, tais como telemóveis ou PDA's, onde a tolerância a faltas é muito importante devido as características do ambiente em que operam.*

1. Introdução

Com o crescente desenvolvimento das novas tecnologias, os sistemas distribuídos tornaram-se importantes e essenciais em sistemas de larga escala. Um dos aspectos fundamentais é a tolerância a faltas que estes sistemas devem fornecer.

No âmbito da cadeira de Tolerância a Faltas Distribuídas [2] teremos de implementar o jogo Pong num sistema distribuído e tolerante a faltas.

A plataforma utilizada (*Appia* [3] suporta a composição de protocolos, ver Secção 4), permitindo através de uma configuração eficiente, a criação de um sistema tolerante a falhas.

Existe também uma emergente corrida a aplicações de entretenimento para dispositivos móveis, nomeadamente *software* lúdico. O jogo Pong é claramente uma aplicação com características que o adequam a este tipo de dispositivos.

2. Motivação

O ambiente da comunicação móvel poderá ser muito hóstil[1] e por isso faz todo o sentido criar jogos multi-jogador distribuídos que sejam tolerantes a faltas. Um problema típico neste contexto é a falha de alguma máquina ou conjunto de máquinas o que poderá levar à incorrecta continuidade do jogo.

No nosso trabalho, o jogo Pong é jogado a 4 jogadores (4 máquinas) que poderão estar dispersos geograficamente. Num típico *jogo em rede* a rapidez do jogo decresce significativamente, podendo haver situações em que uma das máquinas (ou mais) deixe de funcionar correctamente. Numa situação destas é desejável que os restantes jogadores não fiquem prejudicados. Também é necessário que máquinas com maiores atrasos não sejam interpretadas como falhas.

3. Modelo distribuído

Esta versão do Pong vai ser concretizada de modo a suportar vários jogadores em simultâneo geograficamente dispersos. Para isso, foi adoptado um sistema descentralizado para um máximo de quatro jogadores[2], em que um deles será o processo coordenador, designado por servidor mestre, e os outros serão servidores escravos. Assim, haverá *replicação passiva*, na medida em que o servidor mestre envia o estado do jogo aos escravos, em vez de serem os últimos a calculá-lo, como mostra a fig. 1.

Como as máquinas falham arbitrariamente, levanta-se o problema do processo coordenador falhar. Neste caso, haverá rotatividade do processo coordenador. O *Appia* fornece suporte a comunicação em grupo onde permite a existência de uma lista de membros activos, juntamente com o *ran-*

[1] sujeito à rede em que se insere (como a wireless), latência, taxa de erro e outras características

[2] imposição inerente à natureza do jogo

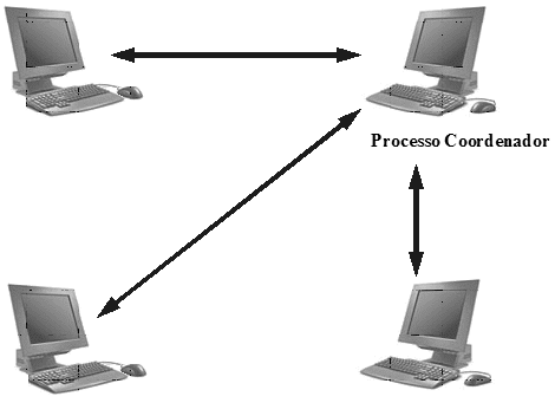

Figura 1. comunicação entre os servidores

king de cada um deles, o que permite que todos os membros identifiquem o próximo processo coordenador sem a necessidade de um consenso.

O tempo de jogo é dividido em intervalos discretos de tempo *t*, e em cada intervalo os servidores escravos enviam ao servidor mestre a posição da sua peça. Após o servidor mestre receber as posições de todos os servidores escravos, calcula a posição da bola, e envia a todos os servidores escravos o estado do jogo actualizado, com a posição de todos os jogadores e da bola.

3.1 Filiação e Particionamento

Num sistema distribuído é por vezes importante existir a noção de grupo e de filiação a um grupo. O jogo Pong passa também por definir um *grupo* de jogadores e consequentemente uma filiação ao grupo. E o que parece ser uma ideia básica traduz-se em algo mais complexo, pois todos os membros de um grupo têm de partilhar a mesma *vista*[3] do grupo. O que implica uma forma de *acordo distribuído* em que todos os membros têm de estar de acordo sobre o que é a vista do grupo.

Uma das causas possíveis para as falhas num sistema distribuído é a *quebra* de rede. Estes cortes em locais arbitrários, pode levar ao *particionamento* da rede. Numa rede passível a particionamento temos, pelo menos três soluções possíveis:

- Bloquear os processos nas partições secundárias;
- *Crashar* os processos nas partições secundárias, para que mais tarde estes sejam reiniciados de novo e o seu estado actualizado;
- Permitir que os processos continuem independentes da partição primária podendo posteriormente fazer a junção às partições.

Tome-se o exemplo de um SGBD de um banco, distribuído pelo país. É suposto este funcionar correctamente mesmo que a ligação física entre Lisboa e o Porto se danifique - originando duas partições disjuntas. Mais tarde, com a situação reposta, as partições serão juntas sem nunca nenhuma das sucursais (em qualquer das partições) ter deixado de fornecer os seus serviços. No caso da aplicação Pong, tal junção de partições não fará sentido, pois tratando-se de um jogo, não será justo haver essa junção posterior de jogos que já se tornaram independentes. Tal situação se tornaria injusta.

Assim, e no actual espírito do grupo, partilha-se a opinião de que a rede está susceptível a particionamento; caso este exista, as partições prosseguirão independentes entre elas, sem nunca se voltarem a juntar no mesmo jogo.

3.2 Detecção de falhas

Na impossibilidade da existência de um detector de falhas perfeito, só se poderá recorrer a uma aproximação do mesmo. Numa rede como aquela em que iremos trabalhar (rede local - Ethernet) tipicamente não haverá grandes atrasos na rede nem perda significativa de pacotes. Mas se a aplicação correr numa rede wireless, o detector de falhas teria de ser *afinado* às características inerentes da rede[4].

Assume-se que poderão falhar no máximo *n-2* máquinas de modo a que o jogo possa ser continuado, i.e. trata-se de um requerimento da aplicação em que existam pelo menos 2 jogadores.

Assume-se também que estamos presente um sistema síncrono, i.e., os atrasos têm limites, o que facilita a detecção de falhas e permite que eventualmente as falhas sejam detectadas (a chamada propriedade de *Strong Completeness* [1])

4. Composição de protocolos

Uma das facilidades do *Appia* é a composição de protocolos. Com isto é possível definir com facilidade uma QoS e organizar de um modo estruturado a composição de protocolos como na fig. 2

Uma das preocupações foi a utilização das camadas que dão suporte aos grupos e filiações como explicado em 3.1.

Como meio de controlar o correcto envio/recepção de mensagens por parte dos processos, foi adicionada a camada FIFO. Como já existe suporte a grupos, e este garante a *sincronização virtual*, só teremos de garantir que as mensagens são enviadas e recebidas pela ordem FIFO. Assim todos os processos *correctos* recebem as mensagens na ordem correcta.

[3] considera-se como vista de um grupo toda a informação relativa ao estado actual do grupo, como por exemplo os membros activos

[4] por exemplo em redes com maior latência, os *timeouts* devem ser maiores

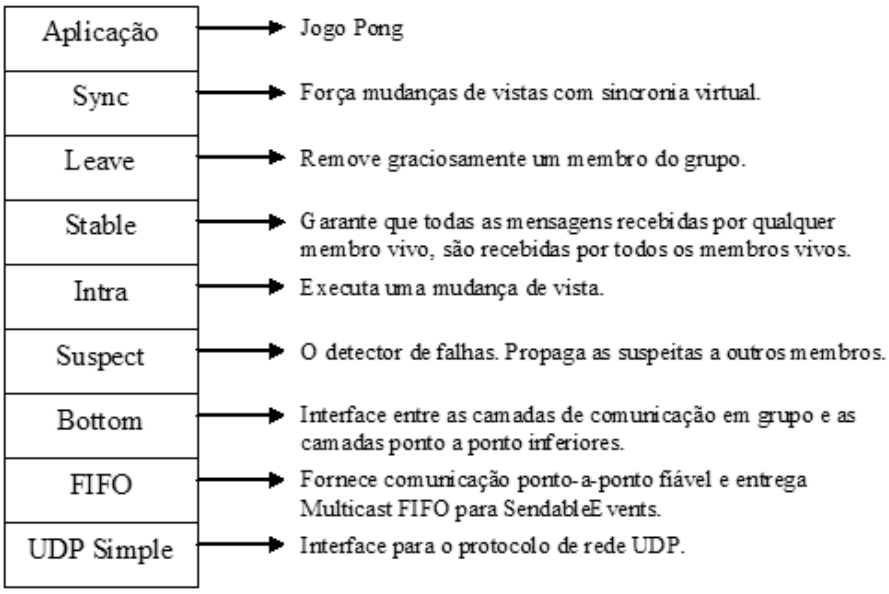

Figura 2. pilha protocular

5. Conclusões

Com base na infraestrutura presente (rede do DI) e da natureza da aplicação várias opções foram tomadas pelo grupo com vista a alcançar o objectivo proposto pela cadeira. Os aspectos mais considerados prendem-se com a plataforma usada na construção deste jogo, nomeadamente o Appia. A escolha de protocolos foi decidida tomando em conta as características próprias do jogo, mais precisamente em relação às condições em que o jogo vai ser construído e utilizado.

Espera-se conseguir bons resultados, tanto no que toca a tolerância a faltas como em ganho de conhecimento pessoal de cada um dos elementos do grupo.

Referências

[1] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 34(1):225–267, 1996.

[2] L. Rodrigues. Tolerância a faltas distribuídas.

[3] L. Rodrigues, H. Miranda, and A. Pinto. Appia. 2000.