Jogo Multi-Utilizador Em Tempo-Real: 4Pong

Tfd06: Fernando Felício 27908 Susana Guedes 27860 Valter Conceição 28011

October 20, 2003

*Abstract: : Existe uma procura crescente de sistemas distribuídos de tempo-real fiáveis por toda a comunidade informática. São necessárias novas técnicas para estas aplicações perpetuarem face a alterações nas condições da rede. Este artigo apresenta uma implementação do clássico jogo Pong da Atari à qual chamámos **4Pong**. Face às suas básicas necessidades de implementação, são apresentados aspectos fundamentais para implementar qualquer software síncrono multi-utilizador em tempo-real. A funcionalidade chave baseia-se na existência de, não dois, mas quatro jogadores em simultâneo.*

1 Introdução

Ao longo dos tempos tem-se verificado um aumento da necessidade de especificar sistemas em termos de requisitos temporais ditados pelo ambiente. Estes são conhecidos como sistemas de tempo-real e a sua computação definida tanto em termos dos seus resultados lógicos como do tempo no qual eles são fornecidos. Há muitos exemplos de sistemas de tempo-real: sistemas de controlo do tráfego aéreo, sistemas de reservas de bilhetes on-line ou até mesmo jogos de computador em tempo-real. [1]

Com o passar do tempo o campo computacional tem mutado a sua antiga faceta da computação pessoal para uma computação multi-pessoal extremamente prometedora. A distribuição tem penetrado na área de tempo-real não só por esse novo desafio como também pela necessidade de descentralizar, equilibrar a carga computacional, replicar e paralelizar a computação. Estas novas necessidades levaram ao aparecimento dos sistemas distribuídos de tempo-real como são hoje conhecidos. Nestes, as garantias temporais têm de ser fornecidas sobre um sistema de máquinas ligadas pela rede. Uma vez conseguido isso, as garantias devem ser associadas a outros atributos, tais como a modularidade, a separação geográfica, a independência das faltas e o balanço da carga, entre outros. [1]

Embora os sistemas distribuídos de tempo-real pareçam uma óptima solução para estes novos requisitos, seria inútil considerar a sua existência apenas em ambientes perfeitos[1] ou em ambientes onde as faltas pudessem ser ignoradas. Como solução a este problema surgiram os sistemas distribuídos de tempo-real fiáveis nos quais se tenta garantir o funcionamento dos sistemas distribuídos de tempo-real mesmo sobre situações faltosas. Um exemplo de um sistema com estas características será um jogo de computador para multi-jogadores em tempo-real. [1]

Todas estas evoluções ao nível das aplicações resultaram num aumento da sua complexidade. De forma a fornecer um desempenho satisfatório, estas aplicações têm-se tornado muito exigentes em termos do suporte à comunicação. Uma aproximação promissora para aliviar a complexidade destes sistemas é confiar em arquitecturas de comunicação configurável capazes de suportar reutilização e composição de componentes. Núcleos de protocolos[2] recentes, como o Ensemble e o Coyote, oferecem um ambiente onde diferentes micro-protocolos podem ser combinados. No entanto, poucos sistemas suportam a coordenação de múltiplos canais. O Appia é um núcleo de protocolo que oferece uma forma limpa e elegante de expressar restrições entre canais. Esta funcionalidade é obtida como uma extensão das funcionalidades dos sistemas correntes. O Appia tem uma boa flexibilidade e um design modular que permite que as pilhas da comunicação sejam compostas e reconfiguradas em tempo de execução. [2]

Com este artigo pretendemos acompanhar a evolução natural dos sistemas considerando uma aplicação antiga, o jogo Pong da Atari[3], e recriando-a numa versão multi-utilizador, distribuída e fiável. O jogo, ao qual chamámos **4Pong** , será desenvolvido utilizando a tecnologia Appia. Mas porquê perder tempo com este jogo? Bom, este jogo pode ser considerado como a aplicação mínima que uma arquitectura para um software síncrono multi-utilizador em tempo real deve ser capaz de implementar. Mas a grande diferença para

[1] Ambientes isentos de faltas, ou ambientes onde estas são ignoradas.

[2] Protocol kernels

[3] O Pong da Atari foi o primeiro jogo de computador a ser desenvolvido e suportava apenas um utilizador.

as implementações já existentes baseia-se na existência de, não dois, mas quatro jogadores em simultâneo, o que se traduz num aumento da complexidade de implementação e no consequente refinamento das técnicas de implementação utilizadas.

O resto do artigo encontra-se organizado da seguinte forma. A Secção II descreve os detalhes do nosso jogo, inclui a descrição do jogo, a arquitectura, a gestão do grupo de participantes, o protocolo de mensagens, o modelo de faltas e a divergência de estados. A Secção III aborda o trabalho relacionado e a Secção IV apresenta as conclusões.

2 Jogo: 4Pong

2.1 Modelo do Sistema e Suposições

4Pong é jogado num campo quadrado com uma baliza por jogador. Cada jogador possui um cursor que pode deslocar lateralmente, de modo a tentar evitar que a bola entre na sua baliza. Em todos os lados onde não existam jogadores a baliza é apenas uma parede onde a bola faz ricochete.

Todos os jogadores começam com zero pontos. Se a bola passa o paddle de um jogador batendo na sua parede, o último jogador a tocar na bola marca um golo. Após um golo, a bola é reposta em jogo pelo jogador que o sofreu, isto é, parte do seu paddle. Cada golo vale um ponto e, no final do jogo, o jogador com mais golos é declarado vencedor.

O jogo começa assim que existam pelo menos dois jogadores. A bola é lançada do centro do campo com um sentido e uma velocidade bem conhecidos.

Durante o jogo é possível que entrem ou saiam jogadores. Quando um novo jogador entra em jogo é-lhe atribuída a pontuação inicial de zero pontos e pode de imediato começar a jogar. Quando um jogador sai de jogo mas ainda ficam suficientes jogadores em campo para prosseguir o jogo, o seu paddle é removido, a sua baliza passa a ser uma simples parede e a sua pontuação é apagada.

O jogo pode terminar em duas situações. A primeira situação é quando a pontuação de um jogador atinge os 20 golos, sendo declarado o vencedor. A segunda situação é quando já não há suficientes jogadores em campo e, nesse caso, o único jogador em campo é declarado o vencedor.

2.2 Arquitectura

Desde que as arquitecturas centralizadas começaram a evoluir para arquitecturas descentralizadas que se têm debatido as vantagens e desvantagens dessa evolução. Numa arquitectura centralizada existe uma autoridade central coordenadora que organiza e condiciona todos os outros participantes. Como consequência, esta solução pode originar uma performance inaceitável já que toda a carga de computação e difusão de informação se encontra num só ponto do sistema, o coordenador. Numa arquitectura distribuída, cada participante mantém uma cópia dos dados processando tudo o que puder de forma local antes de se actualizar com os outros participantes. Embora a consistência se torne mais difícil de garantir, a performance fica significativamente melhorada. [1]

Neste jogo a arquitectura utilizada será distribuída do tipo cliente-servidor. Nesta arquitectura, um jogador será considerado o servidor e o(s) restante(s) cliente(s). Um jogador é servidor quando a bola se encontra no seu domínio[4] e os papéis vão sendo trocados mediante a posição da bola. O(s) cliente(s) vão executando localmente os cálculos possíveis esperando que o servidor os informe das alterações ao jogo.

2.3 Noção de grupo

Visto que este é um jogo multi-utilizador será utilizado um sistema de gestão dos vários participantes. O sistema Appia oferece a funcionalidade de gestão de grupos com recurso a técnicas de sincronismo virtual [3][4]. Esta técnica é vantajosa já que permite a cada utilizador conhecer os outros participantes e, por outro lado, facilita a gestão de entradas e saídas do grupo de jogo. Esta tecnologia permite ainda detectar falhas nos diversos elementos do grupo (ver secção D).

2.4 Protocolo de mensagens

Nesta aplicação é necessário garantir que todas réplicas apresentem o mesmo estado de jogo. Para isso é necessário não só minimizar o tráfego na rede como mascarar o tempo de latência. Por outro lado é necessário que, mesmo tolerando os atrasos, a aplicação seja rápida o suficiente de forma a fornecer uma boa jogabilidade. é necessário, portanto, fornecer uma interface coerente, que se mantenha rápida mesmo com os diversos atrasos na rede.

Como descrito na secção B, será utilizado o método de replicação das actividades distribuidas, não só porque minimiza o total de informação trocada, mas também porque a carga de computação está distribuída pelos vários pontos do sistema, o que aumenta a sua tolerância a faltas[5].

[4] Supondo que o campo quadrangular se encontra dividido em quatro zonas triangulares, uma para cada jogador.

[5] Na solução coordenada o que acontece no caso de uma falha no coordenador?

Para usar o modelo replicado é necessário definir quais as mensagens que são trocadas pelas diversas réplicas. Vamos dividir as alterações ao estado da aplicação em dois conjuntos, as determináveis e as indetermináveis. Por alterações determináveis referimos aquelas que, para um dado evento, todas réplicas obtêm o mesmo resultado sem que haja troca de mensagens[6]. As alterações indetermináveis são aquelas que a não se conseguem prever, nomeadamente interacção de um utilizador numa das replicas. No nosso caso apenas as alterações indetermináveis devem ser propagadas pela rede.

Existem três tipos de alterações que têm de ser propagadas:

1. Quando o jogador move o paddle.
2. Quando um jogador intercepta a bola. Neste caso deve indicar a nova posiçâo, a velocidade e a direcçâo da bola.
3. Quando um jogador sofre golo. Neste caso a bola vai ser colocada no paddle do jogador em questâo e este recomeça o jogo.

2.5 Modelo de faltas

Sendo esta uma aplicação distribuída é necessário ter em conta as faltas gerais que ocorrem em todos os outros sistemas deste tipo. Existem dois grandes grupos de faltas a analisar nos sistemas distribuídos: as omissivas e as assertivas.

No primeiro grande grupo encontra-se as faltas assertivas. Estas existem no domínio do valor e resultam de uma interacção inesperada por parte de um componente. As faltas assertivas podem ser classificadas como semânticas ou sintácticas.

Nas faltas semânticas os valores dos dados enviados numa mensagem ente dois componentes não fazem sentido, por exemplo, a velocidade da bola ser negativa. Estas faltas podem colocar problemas ao nível da consistência dos dados na máquina receptora e, no pior caso, podem ser propagados desta para as outras máquinas. Para tratarmos deste problema, descartamos todas as mensagens em que sejam detectadas alterações. A detecção será efectuada por um algoritmo que tem como base a gama de valores lógicos para cada campo de dados das mensagens. Sempre que o valor não pertença à gama indicada a mensagem é descartada.

As faltas sintácticas não são mais que um subconjunto das faltas semânticas onde os dados são transmitidos nas mensagens são diferentes daqueles esperados pelo receptor, por exemplo, esperamos a velocidade da bola e recebemos "ab". Como solução para este problema o receptor descarta a mensagem sempre não a consiga ler da forma esperada.

No primeiro grupo encontram-se as faltas omissivas. Estas, ao contrário das assertivas, existem no domínio do tempo e surgem sempre que um componente não efectua uma determinada interacção. As faltas omissivas podem ser de paragem, de omissão ou temporais.

As faltas temporais surgem quando um componente se atrasa numa acção. Estas faltas introduzem a noção de grau de latência que se traduz no valor do atraso. A solução a este problema está descrita no capítulo seguinte sobre divergências.

Nas faltas de omissão um componente omite uma acção. Estas são um subconjunto das faltas temporais onde o grau de latência tende para infinito. O factor grau de omissão exprime o total de faltas omissivas sucessivas. Estas faltas estão mascaradas já que cada participante estima o estado seguinte enquanto espera a actualização. Neste caso a actualização não chega e as estimações prosseguem até à actualização seguinte.

As faltas de paragem ocorrem, tal como o nome indica, quando um componente pára. Estas faltas são um subconjunto das faltas de omissão onde o grau de omissão tende para infinito. Como solução a este problema, sempre que seja detectada a paragem de um componente este é retirado de jogo. Esta detecção é possível graças ao mecanismo de gestão de grupos mencionado na secção B.

2.6 Solução para a divergência de estados

Tal como em qualquer sistema distribuído em tempo-real, o jogo **4Pong** pode originar divergências entre as simulações das diferentes réplicas. Neste capítulo vamos apresentar em pormenor um dos problemas que podem ocorrer, bem como a solução adoptada para diminuir os seus efeitos.

Imaginemos um cenário em que a bola encaminha-se para a zona do jogador B, vinda de A. As outras replicas, vão tentar adivinhar se o jogador B falhou a bola ou se a interceptou, caso em que ficam sem saber qual a nova direcção e velocidade. Mesmo que B tenha enviado a mensagem com a respectiva informação basta que exista alguma latência na rede para que a mensagem se atrase, caso em que cada réplica mostra um estado aproximado que pode ser inconsistente com o estado real. Quando recebem a mensagem de B com o novo estado têm de corrigir a posição e trajectória da bola, o que, aos olhos do jogador, dá a noção de um salto no jogo e quebra toda a sensação de se estar a jogar em tempo real.

Como solução para o problema da sincronização dos

[6]Por exemplo, dada a posição inicial da bola e o vector velocidade, todas as réplicas deverão ser capazes de calcular e representar a trajectória da bola.

estados das réplicas iremos utilizar um sistema de retardamento da representação gráfica do jogo. Segundo este método, quando uma bola se afasta de um jogador é utilizada uma heurística para representar o seu movimento de forma retardada. Assim, o jogador receptor tem tempo de acertar, ou não, na bola e propagar o estado resultante da sua interacção para os outros participantes. Se o retardamento for o adequado, cada réplica recebe o novo estado da bola mesmo na altura em que ele deve acontecer, o que faz com que a representação gráfica seja a correcta.

Idealmente, a retardação faz com que a simulação chegue ao ponto da alteração indeterminável mesmo quando é recebida a actualização. Dado que este valor é uma aproximação, a função heurística baseia-se numa tabela com os tempos de latência da rede para cada réplica existente. Mas esta solução levanta o problema de quais os tempos de latência aceitáveis. Basta que a latência seja muito elevada para que os atrasos da bola se tornem mais distractivos do que úteis. Isto pode ocorrer sempre que há sobrecarga na rede, sobrecarga do processador, ou ambos. Neste caso, a divergência de estados tem tendência a aumentar incontrolavelmente. Como solução, qualquer máquina que entre num estado de sobrecarga é retirada do jogo podendo, no entanto, voltar a entrar mais tarde.

3 Trabalho relacionado

A comunicação em grupo através de sincronismo de vistas foi anteriormente estudada por José Pereira, Luís Rodrigues e Rui Oliveira [4]. A estratégia de sincronização por retardamento foi anteriormente utilizada no jogo Ppong! desenvolvido por James Begole e Clifford Shafer em 1999 [5].

4 Conclusões

O jogo textbf4Pong recria o jogo Pong da Atari numa versão multi-utilizador, distribuída e fiável, sendo considerado como a aplicação mínima que uma arquitectura para um software síncrono multi-utilizador em tempo real deve ser capaz de implementar. Comparado com implementações já existentes, esta permite até quatro jogadores em simultâneo. A implementação segue uma arquitectura cliente-servidor onde cada processo é uma réplica autónoma. Para minimizar o tráfego na rede, apenas os estados indeterminísticos são comunicados. De forma a evitar alterações bruscas de estado foi desenvolvida uma estratégia de retardamento através da qual se tenta fazer com que os updates sejam visualizados na altura em que deveriam ocorrer. Através das técnicas utilizadas, a aplicação garante que na maioria das situações, todas as réplicas apresentam o mesmo estado de jogo. O tráfego na rede foi minimizado e os seus atrasos mascarados o que permite que a aplicação seja rápida o suficiente para fornecer uma boa jogabilidade.

References

[1] *P*aulo Veríssimo e Luís Rodrigues, "Distributed Systems for System Architects", Kluwer Academic Publishers ISBN 0-7923-7266-2

[2] *U*RL: http://appia.di.fc.ul.pt

[3] *P*into, "Appia Group Communication Manual", 2001.

[4] *J*osé Pereira, Luís Rodrigues e Rui Oliveira, "Reducing the Cost of Group Comunication with Semantic View Synchrony*"

[5] *J*ames Begole e Clifford Shaffer, "Internet Based Real-Time Multiuser Simulation: Ppong!", TR-97-02, Virginia Polytecnic Inst. e State University, Department os Computer Science.