TFD 2005/2006
Fault Tolerance: Multiplayer Pong

Filipe Cristino – TFD 13
facristino@gmail.com

Abstract: Neste relatório iremos apresentar solução para o conhecido jogo de Pong com algumas modificações. Nesta versão trata-se de um Pong para quatro jogadores seguindo um modelo cliente-servidor com replicação de servidor por modo a permitir tolerância a faltas; para tal usaremos o sistema Appia para gerir a comunicação em grupo dos servidores, permitindo assim a sua replicação. Iremos tentar reduzir o número de mensagem intra-servidores mantendo ao mesmo tempo coerência de estados entre as réplicas. Neste relatório será discutido quais os mecanismos presentes no Appia de modo a permitir tolerância a faltas. É nosso objectivo obter uma transparência cliente-servidor de modo a que o cliente desconheça a existência das réplicas.

1. Introdução

O Pong, jogo já conhecido por muitas gerações e sendo aclamado como o primeiro jogo de computador, é usado neste trabalho como plataforma para uma demonstração de tolerância a faltas em ambienteis distribuídos. Certamente um problema interessante e para nos ajudar na sua resolução usaremos o Appia, uma ferramenta de aplicações distribuídas e tolerância a faltas desenvolvido e fornecido pelo docente da cadeira.

Como fonte de ideias e exemplos, usaremos modelos presentes em aplicações actualmente existentes e disponíveis a publico, nomeadamente outros jogos multiplayer de modelo cliente-servidor, actualmente espalhados pelo mundo e com grandes números de utilizadores. Esses modelos de cliente-servidor distribuídos e com grandes capacidades de jogares, forneceram boas bases de conhecimento e exemplos de implementações, alguns dos quais poderemos usar neste projecto por forma a solucionar o nosso problema.

Nos capítulos seguintes iremos tentar apresentar abordagens que possam resolver o problema da transparência, conectividade, replicação, tolerância a faltas e redução de mensagens entre réplicas. Algumas destas solução já se encontram presentes no Appia, mas devido a natureza modular e dinâmica do Appia [2], cabe a nós, como grupo, descobrir aquilo que achamos ser a melhor maneira de organizar e implementar dessas soluções.

Algumas questões sobre este problema surgem com facilidade, mas as soluções nem sempre podem ser simples. Iremos rever e tentar solucionar algumas dessas questões neste relatório, nomeadamente, como manter a posição da bola igual em todas as réplicas; se o servidor a que um cliente se encontra ligado falha, como manter esse jogador no jogo de um modo transparente e sem falha de conectividade; como reduzir o número de mensagens na rede de modo a que os servidores não sofram de congestão de tráfego; quem tomas as decisões entre todas as replicas do servidor; e mais questões que possam surgir durante o desenvolvimento do projecto.

O artigo está estruturado de forma a primeiro tentar perceber a fundo os problemas que se nos deparam, em seguida uma breve relação desses problemas com aplicações já existentes no mercado, sejam elas outros jogos ou puramente comerciais. Seguidamente tentaremos apresentar uma ou mais solução para os problemas postos no capítulo anterior seguido de uma avaliação da sua exequibilidade e finalmente uma conclusão do progresso feito desde o surgimento dos problemas e das questões até à sua resolução.

O trabalho terá como ponto de partida um jogo de Pong de quatro jogadores em arquitectura cliente-servidor simples, sem replicação e sem tolerância a faltas.

2. Os problemas da replicação

Como se sabe, a replicação de aplicações com vista a tolerar faltas tem os seus problemas. A coerência entre estados das réplicas é um deles. Como garantir que todas as replicas tem o seu estado interno do jogo ou do programa igual entre elas? O problema vai mais além do que simplesmente copiar o estado entre máquinas, temos que garantir que a falha de uma máquina não afecta nem a computação das outras maquinas nem o seu estado ou maneira como vêem o mundo no qual está inseridas. Como já foi dito anteriormente, usaremos o Appia para resolver esse problema de uma forma simples, elegante e transparente, mas temos que perceber o problema e a solução apresentada pelo Appia para o podermos utilizar. Entre os sub problemas da coerência de estados temos o problema de como garantir que uma mensagem do servidor chega as suas réplicas, que as mensagens sejam todas entregues ao mesmo tempo e pela mesmo ordem que foram enviadas. Para garantir que uma mensagem em difusão do servidor seja entregue a todas a sua réplicas temos que usar protocolos de difusão fiável, protocolos esses que nos dão garantias que se uma mensagem é enviada então essa mensagem chega a todas a replicas. Para isso usaremos uma camada do Appia de "reliable broadcast" contida na camada de comunicação em grupo, é também necessário usar ordem causal e ordem total para garantir que as mensagem de estado sejam todas entregues ao mesmo tempo e pela mesma ordem.

Para ajudar a reduzir o tráfego na rede usaremos outra camada do Appia, uma camada de nakFIFO, que ao invés do FIFO normal que devolve confirmações nas mensagens recebidas, o nakFIFO devolve warning de mensagens não entregues. Esta opção deverá reduzir consideravelmente o número de mensagens na rede especialmente entre cliente e servidor.

Para dar alguma transparência ao cliente, de modo a que este não saiba quantas replicas existem e se a replica a que o cliente esta ligado falhar, este não perder a conectividade com o jogo. Através de uma camada de comunicação Point to Point para Group será conseguida a transparência e tolerância a faltas pretendida.

Na nossa solução vamos adoptar por uma aproximação na qual as decisão importantes, como por exemplo se é ponto ou não, serão tomadas por um único servidor por forma a evitar estados e decisões divergentes, usaremos concenso regular por hierarquias [1] entre os servidores para atingir esse efeito. Como cada servidor em um ID interno ao Appia e os IDs são conhecidos em todos os servidor, o servidor com ID mais baixo fica automaticamente eleito o servidor que passaremos a chamar de Servidor de Bola. Este servidor tomará com autoridade absoluta o estado e quaisquer outras decisões relevantes à bola de jogo. Na eventualidade de esse servidor falhar o servidor com o segundo ID mais baixo passa a ser automaticamente o coordenador de bola, e como todas as réplicas sabem os IDs umas das outras todas eles sabem sempre e automaticamente quem é o Servidor de Bola. Para detectar essas falhas, será usada a camada de detecção de falhas do Appia, o suspect.

A comunicação cliente-servidor será feita usando TCP fiável e a comunicação de estados entre servidor com UDP fiável, para que mudanças de estados e o estabelecimento de novos membros do grupo seja feita de uma maneira rápida mas fiável. Estimasse que as mensagem que estejam na rede a qualquer dado momento sejam as mensagens de troca de estados entre servidores, troca de estados com o cliente e movimentos do paddle entre o cliente e o servidor.

3. Implementação

Todos os servidores estarão incluídos em um grupo de comunicação fornecido pelo Appia utilizando uma camada especial para permitir a comunicação com os clientes.

Para os servidores temos duas aproximações possíveis que possivelmente serão utilizadas ao mesmo tempo, relativa à actualização de estado em réplicas. Na primeira actualização, cada vez

um que um jogador altera o seu estado (move o seu paddle) esse estado é agregado com o estado que os servidores tem de momento e difundido pelas restantes replicas; em segundo se nenhum jogador alterar o seu estado antes que um timeout pré determinado acabe, o estado actual dos servidores é actualizado em todas as réplicas, em cada um dos eventos o timeout é reposto a zero.

O servidor de bola irá coordenar a posição da bola por todas a replicas e clientes e como tal terá a sempre a ultima palavra relativamente ao estado da bola. Esta aproximação vai utilizar vectores de direcção: o Servidor de Bola apenas vai emitir updates sobre a bola quando esta muda de vector de direcção ou quando se der um ponto, mas como para um cliente pontuar a bola tem que mudar de direcção logo ele só vai emitir updates em uma situação, quando o vector de direcção da bola muda. Enquanto os updates não são emitidos o processamento da posição da bola é feito localmente por cada servidor e no caso de a bola mudar de vector de direcção em um servidor que não seja o Servidor de Bola, a mudança de vector é feita localmente mas se não for recebido nenhum "ball update" por parte do Servidor de Bola, pouco tempo depois do vector de bola ter mudado localmente, é então emitido um query ao Servidor de Bola a confirmar a posição da mesma. Pretende-se assim evitar a sobrecarga no Servidor de Bola ao enviar menos updates por unidade de tempo.

O servidor de gossip não será implementado fisicamente mas ao invés o endereço do gossip será definido como o endereço de multicast da rede para simplificar a implementação.

Idealmente seria bom evitar clientes ligados ao Servidor de Bola e ligados a servidores diferentes por motivos de carga de servidores, mas podemos ter várias consequências dessa decisão. No primeiro cenário no qual os quatro ou menos clientes se encontram todos ligados ao mesmo servidor, temos a vantagem de termos a certeza que o estado dos quatro clientes é sempre igual entre eles, devido ao facto de todos eles receberem updates ao estado do jogo do mesmo servidor, logo mesmo que essa replica esteja dessincronizada com as outras replicas, ao menos mantêm consistência com todos os clientes. Como ponto negativo deste primeiro cenário é o facto de que se essa replica falha é necessário realocar as quatro clientes para outra replica gastando recurso para tal e possivelmente causando a dessincronização dessa replica. No segundo cenário, cada cliente tenta-se ligar a uma réplica diferente; como ponto positivo temos o factor de reduzir o factor de carga em cada réplica, ajudando-as a manter a sincronização com as outras. Como ponto negativo, acontece que se torna mais difícil manter os quatro clientes com estados consistentes entre eles, visto cada um receber updates de estado de réplicas diferentes que podem, ou não podem, estar sincronizadas entre elas. No terceiro cenário, os cliente comunicam directamente com o grupo de replicas como se a ligação fosse simplesmente peer to peer. Oferece a vantagem de as únicas actualização necessárias trocas entre replicas seriam os updates de bola, utilizando o modelo de Servidor de Bola descrito acima, os updates de estado, nomeadamente do paddle seriam propagados pelos próprios cliente (sem o conhecimento destes) no grupo de replicas a ser mantido. A desvantagem é que se torna difícil decidir qual a replica que responde com updades de estado a qual cliente, e no caso de decidir que só uma replica emite esses updates, torna-se difícil decidir qual a replica que o ira fazer (Com excepção do Servidor de Bola).

Camadas do Appia a serem utilizadas nesta proposta de replicação, para além das camadas base (sem nenhuma ordem especifica): [3]

```
appia.protocols.nakfifo.NakFifoLayers
appia.protocols.totalFastAb
appia.protocols.causal
appia.protocols.group.bottom
appia.protocols.group.suspect
appia.protocols.group.vsync
appia.protocols.group.stable
```

```
appia.protocols.group.leave
appia.protocols.group.intra
appia.protocols.group.inter
appia.protocols.group.heal
appia.protocols.updsimple
appia.protocols.tcpcomplete
```

4.Conclusão

Neste relatório procuramos apresentar a solução que achamos melhor para estes problemas para estes problemas específicos da replicação em sistemas distribuídos, recorrendo à framework Appia. Os serviços implementados por esta plataforma simplificam e tornam transparentes, do ponto de vista de quem procura desenvolver soluções para este tipo de problemas, toda a "mecânica" responsável pelo os vários tipos de comunicação, pelo que se trata de uma valiosa ferramenta para o fim desejado – uma solução, do nosso ponto de vista, simples e suficiente para tolerância de possíveis faltas na replicação de servidores do Pong. Resta, a partir de aqui, desenvolver o trabalho proposto, acrescentando, se possível, melhorias ou correcções aos diversos pormenores de implementação e registar as alterações às potênciais limitações encontradas nesta proposta preliminar.

5. Bibliografia

[1] Rachid Guerraoui and Luís Rodrigues, "*Introduction to Reliable Distributed Programming (Preliminary Draft)"*, Capitulo 5.1.3, pagina 195, *"Fail-Stop Algorithm: Hierarchical Consensus"*

[2] Fernando Felício, Susana Guedes, Valter Conceição, *"Jogo Multi-Utilizador em Tempo-Real: 4Pong"*, 9 de Dezembro de 2003, Capitulo 1, pagina 1 *"Introdução"*, TFD06

[3] http://appia.di.fc.ul.pt/files/appia-2.0-1.jar , Appia software libraries, Hugo Miranda, Alexandre Pinto e Luis Rodrigues