Instituto Superior Técnico,
Universidade Técnica de Lisboa, Portugal

ANACOM

Tolerância a falhas e Segurança

GrupoA-0211

64714, Alexandre Quitério, alexandre.quiterio@ist.utl.pt
66978, Emídio Silva, rafaelsilva@ist.utl.pt
67013, João Oliveira, joao.delgado.oliveira@ist.utl.pt
67020, Joaquim Guerra, joaquimguerra@ist.utl.pt
67036, Miguel Neves, miguel.neves@ist.utl.pt
67074, Rodrigo Bruno, rodrigo.bruno@ist.utl.pt

25 de Maio de 2012

Introdução

No âmbito das cadeiras de Engenharia de Software e Sistemas Distribuídos foi-nos dado um projeto para grupos de seis elementos que consistia na implementação de uma aplicação distribuída constituída por uma rede de operadoras de telemóveis online. No contexto de Sistemas Distribuídos, foi-nos pedido para concretizar requisitos não funcionais de disponibilidade, integridade e autenticidade da informação armazenada nas bases de dados das operadoras. Para concretizar estes requisitos tivemos de aplicar conhecimentos adquiridos na cadeira nas áreas de replicação e de segurança, juntando ainda os conhecimentos adquiridos em Engenharia de Software relacionados com gestão de projeto.

Neste relatório iremos descrever o nosso algoritmo de replicação com tolerância a falhas bizantinas e silenciosas e como este foi implementado num ambiente assíncrono como o do projeto. Iremos ainda indicar como dividimos o nosso grupo em dois subgrupos para Quóruns e Segurança. O modelo de faltas considerado admite não mais do que uma falha silenciosa e uma bizantina (podem ser em simultâneo).

Protocolo de Replicação

Para garantir tolerância a uma falta bizantina no projeto tivemos de implementar uma variação do protocolo de Quóruns estudado nas aulas. Note-se que, no contexto do projeto, para além da rede ser assíncrona, a mesma não garante que as mensagens são entregues por ordem, no entanto garante a entrega das mesmas ao destinatário, a menos que o mesmo esteja em baixo. Nesta secção iremos descrever o protocolo, sendo a implementação do mesmo detalhada na secção seguinte.

Para o nosso protocolo funcionar corretamente precisa, no mínimo, de 5 réplicas, em que apenas uma delas pode estar em falha bizantina e outra em falha silenciosa. Para que as diferentes réplicas não tenham informação inconsistente, a nossa implementação ordena as mensagens que possam alterar o estado das bases de dados (mensagens de escrita), no entanto, as mensagens que não têm este efeito (mensagens de leitura) não são ordenadas com o objectivo de tornar o protocolo mais eficiente, visto que estas não têm qualquer impacto na consistência das bases de dados.

Cada réplica tem um timestamp na base de dados, semelhante ao protocolo de Quóruns. Este timestamp é alterado sempre que a réplica recebe uma mensagem de escrita que altere o estado da sua base de dados (pode não alterar em caso de exceção). O novo timestamp é atribuído com base no relógio do servidor de apresentação, o qual coloca o tempo atual do seu relógio em todas as mensagens que envia. Optámos por ser o servidor de apresentação a determinar os timestamps para garantir que os mesmos estão sincronizados entre as diferentes réplicas. Optámos também por manter apenas um timestamp para toda a base de dados porque, visto que o servidor de apresentação serializa os pedidos, esta opção tem granularidade suficiente para garantir a execução correta do protocolo e simplifica bastante a implementação do mesmo.

Passemos agora à descrição do protocolo em si, começando pelas mensagens de leitura. O servidor de apresentação envia a mensagem para as 5 réplicas em paralelo, ficando de seguida à espera de 4 respostas. Note-se que, no contexto deste relatório, admitimos que o termo resposta inclui as exceções que os servidores de aplicação podem lançar. Após receber as 4 respostas, o servidor começa por escolher uma resposta A de entre as que têm o timestamp mais elevado e procura uma outra resposta B igual entre as que têm o mesmo timestamp que A. Se encontrar alguma resposta B igual a A, então retorna A, caso contrário descarta A, escolhe a próxima resposta com o timestamp

mais elevado e volta a executar o mesmo procedimento para esta resposta. Este procedimento permite tolerar uma falha silenciosa ou atrasos na rede a comunicar com uma das réplicas. Para além disso, ainda permite tolerar o caso em que a base de dados de uma outra réplica está corrompida mesmo que o timestamp da mesma seja muito elevado, visto que o protocolo retornaria uma resposta dessa réplica apenas se recebesse outra igual, de uma réplica diferente, com o mesmo timestamp, o que não acontece visto que admitimos apenas uma falta bizantina no modelo de faltas considerado. E se o servidor de apresentação não receber duas respostas iguais? Após detalhar as escritas será explicado porque é que isto nunca acontece no modelo de faltas considerado.

As respostas às mensagens de escrita são processadas de forma muito semelhante às mensagens de leitura, a única diferença está na ordenação das mensagens. No servidor de apresentação é mantido um número de ordem por operadora, que é incrementado sempre que é enviada uma mensagem para as réplicas da mesma. Nos servidores de apresentação, as mensagens de escrita que não sejam entregues dentro de ordem são bloqueadas até chegarem as mensagens anteriores. Caso o número de ordem seja 0, os servidores de apresentação reiniciam os seus contadores. Optámos por implementar a ordenação desta forma porque o servidor de apresentação pode ser reinicializado, caso em que os seus números de ordem são repostos 0 visto que o mesmo não guarda estes números de forma persistente, e, apesar desta não ser a solução mais segura, é bastante simples e não requer que os servidores de aplicação estejam sincronizados com o servidor de apresentação inicialmente.

Como as mensagens de escrita são ordenadas e o servidor de apresentação espera por 4 respostas entre 5, apenas poderá haver uma réplica com a base de dados desatualizada num pedido de leitura. Sendo assim, o pior caso ocorre quando o servidor de apresentação recebe duas respostas corretas, uma resposta bizantina e uma resposta com um timestamp atrasado, sendo sempre possível ao servidor de apresentação determinar uma resposta correta sem reenviar o pedido no modelo de faltas considerado.

O protocolo está ainda preparado para o caso em que um servidor em falha bizantina responde várias vezes ao mesmo pedido na tentativa de levar o cliente a escolher uma resposta falsa. O protocolo usa os certificados implementados na segurança para verificar se já foi recebida uma resposta desse servidor.

Implementação

Nesta secção iremos apresentar alguns detalhes de implementação relevantes do protocolo detalhado na secção anterior, começando pelo comportamento do servidor de apresentação em caso de catástrofe.

O nosso protocolo está preparado para tolerar, no máximo, uma falha bizantina, uma falha silenciosa e um servidor com informação desatualizada. Em caso de catástrofe tínhamos duas opções, ou o protocolo devolve uma notificação de erro ou faz o seu melhor para conseguir devolver uma resposta correta. Achámos que era preferível tentar devolver sempre uma resposta de forma a tornar a nossa aplicação o mais adaptável possível caso os requisitos de disponibilidade sejam alterados e haja adição ou remoção de réplicas. Caso o número de réplicas seja reduzido para três, a nossa aplicação implementa um protocolo de votação síncrono como o ensinado nas teóricas.

Quanto ao UDDI, o servidor de apresentação mantém uma cache de endereços dos servidores das operadoras com o objectivo de aumentar a eficiência do servidor de apresentação. As pesquisas ao

servidor UDDI são feitas antes de enviar uma mensagem de escrita ou quando o protocolo não recebe respostas suficientes a uma mensagem de leitura.

De forma a garantir autenticidade e integridade das mensagens trocadas utilizamos o RSA e o MD5 para assegurar este requisitos não funcionais. Para garantir autenticidade, basta encriptar o digest resultante com a chave privada da entidade que pretende enviar a mensagem, desta forma o destinatário pode garantir que ao desencriptar a mensagem com a chave pública dessa operadora que a mesma mensagem não foi adulterada. Para garantir que as mensagens não são forjadas o emissor da mensagem tem de utilizar o MD5 e efetuar um digest do body. O destinatário ao desincriptar a signature e efetuar paralelamente um digest ao body da mensagem, se a signature desencriptada for igual ao digest calculado no momento garante-se que a mensagem não sofreu alterações. Com estes mecanismos garantimos que se alguma das mensagens não cumprir com estes prossupostos é descartada pelos handlers.

Divisão em subgrupos

Após fazermos uma análise detalhada do enunciado e de termos levantado uma lista de requisitos de ambas as partes principais do projeto (tolerância a faltas e segurança), decidimos dividir a equipa em dois subgrupos.

Para esta divisão reunimo-nos e discutimos sobre qual a área em que cada elemento se sentia mais à vontade e possuía mais conhecimento de forma a aumentar a motivação, produtividade e eficiência de cada elemento. Chegámos a um consenso rapidamente visto que parte do grupo chegou rápido a uma solução para o protocolo de tolerância a faltas e os interesses de cada elemento permitiram facilmente a criação de dois subgrupos com um número de elementos igual. De seguida apresentamos a subdivisão final do grupo:

- Segurança – Alexandre Quitério, João Oliveira e Joaquim Guerra;
- Tolerância a Faltas – Emídio Silva, Miguel Neves e Rodrigo Bruno.

Antes de iniciar a concretização dos requesitos não funcionais do enunciado de Sistemas Distribuídos, no momento da divisão de equipas foi decidido que cada uma delas deveria desenvolver a sua parte o mais individual possível, minimizando as dependências entre ambas, de forma a facilitar a demonstração final do projecto. Este critério foi cumprido por ambas as equipas e desta forma, no momento da junção, as alterações a fazer, para conseguir implementar de forma completa o nosso protocolo de tolerância a faltas, foram mínimas.