UNIVERSIDADE TÉCNICA DE LISBOA
INSTITUTO SUPERIOR TÉCNICO
LEIC-T, LEIC-A, LERC, MEIC-A

2ª parte do Projecto de Sistemas Distribuídos 2011/2012

1 Introdução

A solução desenvolvida no 1º Projecto é muito limitada em termos de tolerância a faltas e segurança.

- Por exemplo, caso um servidor de operador falhe silenciosamente, qualquer operação sobre telemóveis do respectivo operador fica indisponível.
- É também relativamente fácil a atacantes pôr em prática diferentes ataques como, por exemplo, modificar as mensagens trocadas entre servidor de apresentação e servidor de operador por forma a levar um utilizador a pensar que recebeu uma mensagem que nunca lhe foi enviada, ou que tem um saldo inferior ao seu saldo real.
- O sistema é também vulnerável a servidores que incorram em falha bizantina (por exemplo, servidores de operador com uma implementação com erros de programação acidentais ou intencionalmente introduzidos por um atacante).

Na segunda parte do projecto de SD pretende-se introduzir requisitos não funcionais que tornem o projeto mais fiável e seguro. De seguida descrevem-se os requisitos da solução a desenvolver, que focam cada uma das limitações acima apresentadas: tolerância a falhas silenciosas de servidores de operador; segurança da interacção entre servidor de apresentação e servidores de operador; e tolerância a falhas bizantinas de servidores de operador.

2 Requisitos não funcionais

2.1 Tolerância a falhas silenciosas de servidores de operador

Nesta fase, vamos considerar que no nosso modelo de faltas todos os servidores têm um padrão de falha silenciosa. Mais precisamente, deve assumir-se que, entre as N réplicas de cada servidor de operador, uma delas (e não mais que uma) pode falhar silenciosamente. Pretende-se que cada grupo desenhe e implemente uma solução que permita que o sistema continue disponível mesmo quando tal falha ocorre.

Para tal, cada servidor de operador deverá passar a ser replicado em N réplicas. A replicação é activa. Cada pedido que antes (no 1º Projecto) era enviado para um único servidor de operador passará agora a ser enviado para todas as réplicas (desse servidor de operador).

Sugere-se fortemente que a solução seja inspirada no protocolo de quóruns dado nas aulas teóricas.[1] Fica ao critério de cada grupo definir a granularidade do controlo de versões; ou seja, decidir se existe um timestamp por cada operador, ou se por cada telemóvel, etc.

Sempre que o servidor de apresentação pretenda invocar uma dada operação do servidores de operador, o pedido correspondente deve ser enviado para todas as réplicas. O servidor de apresentação fica então à espera da chegada de um número de respostas suficiente (isto é, um quórum). Os quóruns usados devem ser de maioria.

Deve assumir-se que só existe uma instância do servidor de apresentação a correr. Adicional-

[1] Note-se que, por diferentes razões, que o protocolo de quóruns dado nas aulas teóricas não é uma solução correcta para os requisitos deste projecto. Como tal, caberá aos alunos conceber um algoritmo que seja correcto (e optimizado) dados os requisitos descritos neste documento.

mente, deve assumir-se que, apesar do servidor de apresentação ser *multithreaded* e poder receber pedidos concorrentes de invocação de serviço, vindos de múltiplos clientes, o servidor de apresentação serializa esses pedidos. Ou seja, só após uma dada invocação de serviço ser finalizado é que o servidor de apresentação inicia outra invocação de serviço.[2]

O servidor de apresentação conhece, a priori, o número total de réplicas (N). No entanto, para permitir mobilidade das réplicas (por exemplo, quando o administrador migra uma réplica para uma máquina diferente ou para uma rede diferente), os endereços (URL) dos endpoints das réplicas podem variar ao longo do tempo. Para suportar essa mobilidade, o sistema deve recorrer a um servidor privado de UDDI (*UDDI registry*), de endereço bem conhecido.[3] Cada réplica, ao ser lançada (*deployed*), deverá registar o seu serviço no UDDI. Dessa forma, o servidor de apresentação poderá, consultando o servidor UDDI, descobrir dinamicamente os endereços de cada réplica. Compete a cada grupo definir uma política eficiente que minimize as consultas feitas ao servidor UDDI.

Deve assumir-se que as falhas de paragem das réplicas são permanentes e que não existe manutenção para repor em serviço os servidores (novamente uma hipótese que provavelmente seria diferente num caso real). Devido a esta assunção, não é necessário definir e implementar um protocolo de recuperação das faltas de paragem das réplicas para garantir consistência do seu estado quando fossem recolocadas em serviço.

Quanto ao servidor de apresentação, este pode falhar silenciosamente e, de seguida, ser reiniciado (por exemplo, por um administrador da rede). Uma vez que o servidor de apresentação não mantém qualquer estado persistente, reiniciá-lo é uma operação trivial.

Deve assumir-se que não é conhecida uma latência máxima para a transmissão das mensagens através da rede; ou seja, um modelo de comunicação assíncrono. Deve assumir-se que o protocolo de transporte (que, em teoria, pode não ser o HTTP) tolera a perda temporária de mensagens na rede mas não garante o envio ordenado das mensagens.

2.2 Segurança da interacção entre servidor de apresentação e servidores de operador

Uma vez que os componentes do ANA.COM se encontram ligados pela Internet, o sistema está vulnerável a vários ataques. Para este projecto pretende-se criar canais entre servidor de apresentação e servidores de operador que assegurem a autenticidade e integridade das mensagens SOAP trocadas.

A solução deverá ser baseada em assinatura digital de chave pública, usando uma Autoridade de Certificação (CA). A CA deve ser implementada como um Web Service que oferecerá uma interface para gerir e distribuir certificados digitais de chave pública, conforme explicado de seguida.

Quando um servidor é iniciado (deployed), deverá já ter carregados: o seu par de chaves assimétricas (chave privada, chave pública); e a chave pública da CA, que se assume imutável. A CA, por seu lado, conhecerá a chave pública de cada servidor, para além da chaves privada e pública da própria CA. Cada servidor deverá, nessa situação inicial, solicitar à CA que assine o certificado da chave pública desse servidor.[4]

Quando o administrador de um servidor suspeitar que a respectiva chave privada está em risco de ser descoberta, o servidor deve gerar um novo par de (chave privada, chave publica), e solicitar à CA que revogue o certificado anterior e gere um novo com a nova chave pública. [5] Após gerar o certificado, a CA devolve-o como resposta ao servidor que o solicitou.

Junto à assinatura digital enviada em anexo

[2]Esta serialização pode ser facilmente implementada usando um trinco lógico no servidor de apresentação.

[3]Deve ser usado o servidor jUDDI v3.1.3, executado sobre o servidor aplicacional Tomcat. Mais instruções serão publicadas no site dos laboratórios.

[4]Por simplificação do projecto. Na realidade, a entrega da nova chave pública para gerar novo certificado é feita usando um canal seguro (e.g. presencialmente e com prova de identidade).

[5]Mais uma vez, por simplificação, no projecto permite-se que a nova chave pública seja entregue à CA através de um canal inseguro, quando na realidade tal é normalmente feito através de um canal seguro (e.g., presencialmente).

com cada mensagem SOAP devem ir também os certificados necessários para sua validação por parte de quem recebe a mensagem. Cada servidor que receba uma mensagem deverá verificar o seguinte:

1. Validade temporal do certificado
2. Validade da assinatura digital da CA no certificado
3. Verificação que certificado não foi revogado
4. Validade da assinatura digital do cliente

Para o passo 3, cada servidor manterá localmente uma lista negra de certificados revogados. Essa lista é periodicamente solicitada através de uma invocação de um serviço da CA que devolve a lista negra actual. Naturalmente, deverão ser usados os mecanismos necessários para assegurar que a lista negra recebida a partir da CA é autêntica e íntegra.

Os algoritmos de criptografia a utilizar são os implementados na biblioteca JCE do Java.

2.3 Tolerância a falhas bizantinas de servidores de operador

Como último requisito pretende-se tornar mais abrangente o modelo de faltas, considerando que a réplica em falha pode tomar um comportamento bizantino (em vez de falhar silenciosamente).

Cada grupo deverá conceber uma solução que, combinando os mecanismos desenvolvidos para resolver os requisitos das secções anteriores (ou seja, replicação activa por quóruns e assinatura digital de chave pública), consiga também tolerar a falhas bizantina de uma réplica. O algoritmo a conceber não é ensinado nas aulas teóricas, com a intenção de obrigar cada grupo de projecto a construir por si mesmo uma solução criativa partindo dos mecanismos que conhece.

Como exemplos de situações que devem ser toleradas, considerem-se as seguintes:

- Assuma que, entre N réplicas, existe uma que tem a sua base de dados corrompida devido a falha do sistema de ficheiros. Quando chega um pedido de uma operação de leitura (e.g. pedido de consulta de saldo) a todas as réplicas, a réplica bizantina devolve um saldo errado e associa à sua resposta um valor de timestamp artificialmente alto (ambos os valores errados são devidos à base de dados corrompida). Usando o protocolo base (dado nas aulas), o timestamp da resposta da réplica bizantina será provavelmente o maior de todos os timestamps recebidos pelo servidor de apresentação e, consequentemente, o saldo retornado ao utilizador será o saldo errado.
- Um outro exemplo é o de uma réplica cujo programa foi intencionalmente adulterado por um atacante. Sempre que recebe um pedido, esta réplica envia múltiplas respostas, cada uma em nome de uma réplica diferente. Todas as respostas ilegítimas levam um timestamp artificialmente elevado, por forma a tentar levar o cliente a eleger essa resposta como aquela a retornar ao servidor de apresentação.
- Considere agora que o servidor de apresentação envia às réplicas um pedido de uma operação/serviço que modifica o estado das mesmas. Tal como antes, assuma o caso em que uma das réplicas é bizantina. Pode acontecer que a réplica bizantina, ao receber o pedido, o ignore (ou seja, não o execute localmente) e incorrectamente responda *acknowledge* ao servidor de apresentação. Se o servidor de apresentação receber uma maioria de respostas *acknowledge* (incluindo a resposta da réplica bizantina) irá pensar que o seu pedido de escrita foi aplicado sobre uma maioria das réplicas. No entanto, dado que a réplica bizantina não o aplicou, é possível que o pedido tenha apenas chegado e sido aplicado sobre uma *minoria* de réplicas correctas - o que põe em causa a correcção do protocolo de replicação.

Pretende-se que o protocolo de replicação activa seja estendido por forma a que comportamentos bizantinos como os acima mencionados não levem o sistema a devolver respostas incorrectas ao utilizador. Ou seja, o servidor de apresentação só deverá retornar a resposta de maior

timestamp quando tiver a certeza de essa resposta foi dada por, pelo menos, uma réplica correcta. Adicionalmente, o protocolo de replicação desenvolvido nesta fase deverá assegurar-se que qualquer pedido de operação/serviço que envolva alteração de estado só é considerado como concluído depois de uma maioria de réplicas correctas (e não uma maioria de réplicas quaisquer) confirmarem que receberam e aplicaram o mesmo pedido.

3 Metodologia

A metodologia proposta para a 2ª parte do projecto de SD é a seguinte:

- **Etapa 1, em grupo de 6, semana 1:** Em grupo, todos os elementos discutem e concebem, em pseudo-código, uma solução que resolva os 3 conjuntos de requisitos enunciados neste documento. Opcionalmente, sugere-se que o esboço da solução seja entregue ao docente de laboratório para que ele o possa comentar, apontar limitações e sugerir melhorias.
- **Etapa 2, em grupos de 3, semanas 2 e 3:** Um sub-grupo de 3 elementos dedica-se a realizar uma solução autónoma para os requisitos da Secção 2.1 (replicação activa de servidores de operador). O outro sub-grupo de 3 realiza uma solução autónoma que resolva o pedido na Secção 2.2 (segurança). As soluções desenvolvidas por cada grupo deverão ser desenvolvidas autonomamente e deverá ser possível testar cada uma independentemente na entrega final.
- **Etapa 3, em grupo de 6, restantes semanas:** Finalmente, as soluções de cada sub-grupo deverão ser combinadas por forma a obter uma solução que também cumpra os requisitos enunciados na Secção 2.3 (tolerância a falhas bizantinas de servidores de operador).

As implementações dos requisitos não funcionais descritos neste enunciado deverão ser, tanto quanto possível, separadas adequadamente da implementação dos requisitos funcionais. Deverá também consistir em código o mais genérico e reutilizável possível.

3.1 Grupos só-SD

O presente enunciado aplica-se também aos grupos só-SD. No entanto, no caso de grupos só-SD aplica-se a seguinte excepção, por forma a adaptar a carga de trabalho à dimensão reduzida dos grupos:

- Não é necessário usar certificados de chave pública nem desenvolver uma CA. Em alternativa, a chaves pública de cada servidor deve ser permanente e pré-conhecida por todos os restantes servidores.

4 Entrega e Avaliação

O prazo de entrega da primeira parte do projecto de SD é dia 15/Maio às 23:59.

A entrega final deverá incluir um artigo de 3 páginas A4 (letra em tamanho 11, margens de 2cm) a descrever a solução desenhada e implementada para tolerar falhas bizantinas de servidores de operador.

No caso de grupos ES/SD, a entrega do projecto é feita através do repositório svn, criando a *tag* RELEASE_5. Esta *tag* representará a versão do código produzido referente à 2ª parte do projecto de SD e à 4ª parte do projecto de ES que os alunos querem submeter para avaliação.

No caso de grupos só-SD, a entrega do projecto é feita através do Fénix. Instruções mais detalhadas serão publicadas no site da cadeira.

As discussões serão realizadas em grupo, sendo as questões dirigidas a cada aluno individualmente.

A demonstração será planeada pelo grupo, por forma a demonstrar as principais virtudes do projecto dentro do tempo limitado definido para a demonstração (a duração da demonstração será publicada no site). Na demonstração o grupo deverá usar:

- Um operador apenas, replicado em N réplicas.
- O valor N usado deve ser o mínimo possível que permita tolerar uma réplica em falha bizantina; compete ao grupo decidir qual o valor N adequado.