Relatório – Grupo 11- SD-STORE-A/SD-ID-B

Diogo Painho Rodrigues

73245

Natalino Cordeiro

74117

André Caldeira

75580

Requisito SD-Store-A (Replicação)

-Lançamento dos Servidores

Os nossos servidores/réplicas são lançados manualmente na consola. Definimos um porto da máquina local onde queremos lançar o serviço e passamos um argumento que irá definir o identificador do serviço (SD-STORE-1, SD-STORE-2, ... ,SD-STORE-N). N acaba por ser também o valor que vamos definir no "Front-End" para posteriormente aplicar as chamadas assíncronas e o *Quórum Consensus*.
Os servidores são assim lançados e registam-se com o respectivo nome de serviço e url no servidor de nomes, que está no url: http://localhost:8081, sendo que este porto fica indisponível para lançar servidores.

-Ligação do Cliente aos Servidores

Com vista à replicação do serviço, criámos uma classe "Front-End" onde os pedidos do cliente são devidamente tratados e processados. Este processo começa por em cada método que o cliente invoca, fazer uma listagem de todos os *endpoints url's* onde o serviço com o padrão SD-STORE-X está registado., guardando-os numa lista. De seguida, criamos um único *stub* para a chamada dos métodos nos respectivos servidores, sendo que fazemos uma mudança da variável de contexto a cada iteração sobre a lista de *url's de endpoints* que temos disponíveis. Usando ciclos "for" conseguimos facilmente fazer chamadas síncronas sobre os servidores.

-Handlers

Para conseguir cumprir com o protocolo *Quorum Consensus*, sentimos a necessidade de associar uma *tag* a cada documento armazenado no sistema. Esta *tag* é composta por um numero sequencial e um *clientId* que é sempre único. Ainda assim, para prosseguir com o protocolo, é necessário informar o "Front-End" das *tags* associadas ao pedido para poder tomar as decisões mais corretas perante os pedidos dos clientes, e, para tal, criámos *handlers* onde são trocadas as respectivas *tags*. Os *handlers* trabalham com base em mensagens SOAP constituídas por *headers* e *body's*, no entanto na nossa implementação só manuseamos a secção dos *headers*.

-Chamadas Assíncronas

Na nossa implementação de chamadas assíncronas, não fazemos nada para além do que consideramos normal, ou seja, adicionamos o async.binding.xml para ativar as chamadas assíncronas e dentro do "Front-End" implementámos todo o código relativamente a este tipo de chamadas. Começamos por calcular o Q com base no N , no WR/RT, fazemos os "request's" a todos os url's disponíveis no momento, e de seguida, esperamos por Q

respostas dentro de um ciclo *while*. Estas respostas vêm em formato de *tags* por *handlers*, sendo que são posteriormente tratadas consoante o caso de *load* ou *store*. É preciso referir que o sistema espera por Q respostas mas está preparado para receber respostas de todos os servidores.

-Replicação

Para garantir a replicação entre as réplicas de servidores, aplicamos todo o *Quorum Consensus*. Não tomámos nenhuma decisão fora do normal, visto que seguimos à risca o conteúdo dado nas aulas teóricas e fornecido nos slides. No entanto toda a replicação é gerida com *tags* que são enviadas e recebidas por meio de *handlers.*

-Situações Insatisfatórias do Requisito SD-STORE-A

No decorrer do projeto, encontrámos várias situações que poderiam ser potenciais formas do sistema falhar ou mostrar ineficiência, sendo que as procurámos resolver. Não conseguimos também fazer os testes, principalmente aos *handlers* porque o tempo foi escasso. Ainda assim, não temos a certeza se a construção dos *handlers* para as *tags* estão feitos de forma correta, uma vez que só manuseamos os *headers* e não construímos um *body*.
Também estamos a promover uma situação no *Quorum Consensus* que torna a execução do sistema pouco eficiente, ou seja, à medida que recebemos as respostas armazenamos o conteúdo do lado do cliente de SD, sendo que estamos a trazer todo o conteúdo de todas as réplicas. Por fim, não tivemos tempo de testar se o SD-Store está preparado para trabalhar com múltiplos clientes do *BubbleDocs*.

Requisito SD-ID-B (Segurança dos Documentos)

-Princípios Base

Para podermos ter segurança nos documentos o cliente do lado de ES cria um tipo de cliente do SDStore seguro, ou seja, as operações de *store* e *load* que são invocadas fazem a cifra ou decifra dos bytes trocados e só depois invoca o método de *store* ou *load* do cliente em si que faz a ligação com as réplicas. É preciso ter em conta que para cifrar e decifrar documentos necessitamos de uma chave encriptada que é dada pelo SD-ID e passada como argumento do construtor do cliente do lado de ES quando o mesmo é criado.

-Cifra e Decifra de Documentos

Os documentos são cifrados com base numa chave simétrica DES e encriptados segundo o protocolo DES/ECB/PKCS5Padding, sabemos não ser a melhor solução, mas foi a mais rápida e menos problemática para a implementação.

-Autenticação dos Documentos

Por forma a garantir a integridade dos documentos implementámos o MAC, onde temos uma classe Singleton que serve de dicionário que faz o *hash* entre o DocUserPair e a Key usada para cifrar e decifrar documentos. O documento só é decifrado se passar na verificação da autenticação. Por fim, com a nossa implementação as chaves MAC ficam todas do lado do cliente do SD-Store-Cli garantido alguma segurança na troca dos bytes dos documentos.

-Situações Insatisfatórias do Requisito SD-ID-B

A situação insatisfatória que encontramos neste requisito foi o facto do cliente BubbleDocs ter acesso ao código do cliente não-seguro, sendo que por ai pode passar a barreira de segurança. Ainda assim o uso de uma chave DES torna-se pouco seguro sendo que é cifrada com poucos bytes. Desta forma seria mais conveniente aplicar uma chave AES.