ULHT – Universidade Lusófona de Humanidades e Tecnologias
Licenciatura em Engenharia Informática / Computação Distribuida

Projecto: Armazenamento Associativo

1. Introdução

Este trabalho tem como objetivo o desenvolvimento de um sistema de Armazenamento Associativo baseado numa arquitectura híbrida em ambiente distribuído. Armazenamento Associativo também conhecido como "Content Addressable Storage", ou abreviado como CAS, é um mecanismo para armazenar informações com base não no endereço de localização, mas no endereço do conteúdo. Quando se solicita ao CAS o armazenamento de um ficheiro, ele calcula um hash com base no seu conteúdo, retorna o endereço do conteúdo armazenado e prossegue para o armazenamento físico do conteúdo. Este endereço é então usado como chave para recuperar o conteúdo, de modo que quando solicitamos ao CAS este endereço ele retorna o conteúdo associado.

Os algoritmos de hash mapeiam dados de comprimento variável para dados de comprimento fixo e permitem uma associação única entre o conteúdo e o seu hash. O processo é unidirecional e impossibilita descobrir o conteúdo original a partir do hash. O "checksum" (soma de verificação) muda se um único bit for alterado, acrescentado ou retirado ao conteúdo. Este sistema garante:

- Integridade dos conteúdos: Se o conteúdo armazenado no CAS for alterado haverá uma incongruência entre o hash utilizado como endereço e o hash gerado dinamicamente.
- Evitar redundância: Se solicitarmos ao CAS o armazenamento de um conteúdo já armazenado este reconhece que o hash já está presente e, portanto, não o armazena novamente.

O trabalho deverá ser realizado na linguagem Java e produzido individualmente ou em grupos de dois alunos.

2. Arquitectura

A solução que se pretende desenvolver deverá implementar um sistema CAS distribuído. Este sistema possui múltiplos nós que comunicam entre si e clientes que interagem com um nó central, como representado na figura 1:

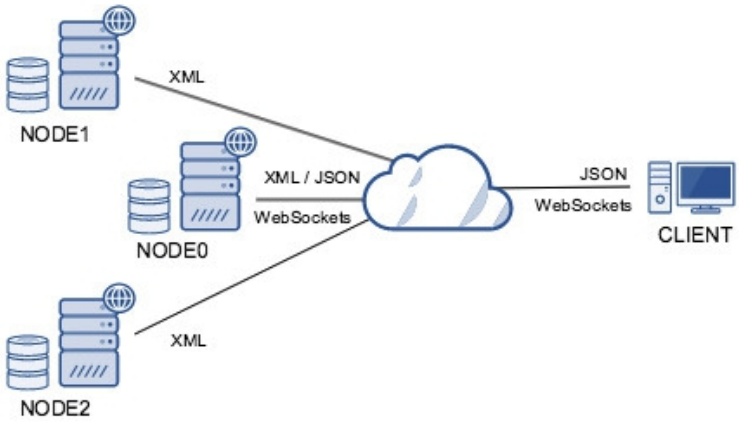

Figura 1: Arquitectura

Os nós executam o sistema CAS em modo distribuído e comunicam entre si através de mensagens XML. O NODE0 (nó central) disponibiliza aos clientes a capacidade de interagir com os serviço do sistema CAS através de mensagens JSON. Os clientes informam-se sobre o estado dos nós através de WebSockets.

3. Implementação

Os nós deverão implementar a capacidade de executar operações CRUD (Create, Read, Update, Delete) sobre o sistema CAS por parte de qualquer nó através de mensagens XML suportadas em WebServices REST ou XML-RPC. O NODE0 deverá implementar particularmente a capacidade de executar operações CRUD sobre o sistema CAS por parte de qualquer cliente através de mensagens JSON suportadas em WebServices REST. O NODE0 e clientes implementam WebSockets de modo a que o cliente consiga obter informação relativa ao estado dos nós.

A solução deve oferecer as seguintes operações entre Cliente-Nó Central:

- inserir recurso / upload (Create)
- obter lista de recursos (Read);
- obter informação sobre recurso (Read);
- obter recurso / download (Read);
- apagar recurso (Delete);
- obter lista de nós ON/OFF

A solução deve oferecer pelo menos as seguintes operações entre os nós:

- inserir recurso / upload (Create)
- obter recurso / download (Read);
- obter informação sobre recurso (Read);
- actualizar recurso (Update);
- apagar recurso (Delete);
- actualizar estado ON/OFF (Update)

4. Persistência dos dados

A solução deverá recorrer á serialização de objectos para ficheiros de tal forma que seja possível a recuperação de toda a informação necessária para manter o serviço CAS em caso de falhas transitórias.

5. Sistema CAS

A solução deverá suportar o sistema CAS, permitindo múltiplos clientes e múltiplos nós.

5.1 Nós

Os nós implementam a capacidade de executar operações CRUD (Create, Read, Update, Delete) necessárias para o funcionamento do sistema CAS. O formato de mensagens / serviços é definido pelo(s) aluno(s) de acordo com o ponto 3. Utilize por defeito o porto **8080** para comunicações HTTP. Os Nós implementam um ambiente multi-threading, servindo ao mesmo tempo mais de um nó. Para registo de todas actividades de comunicação deve haver como opção guardar num ficheiro de log, *debug.log*.

Os nós devem manter tabelas de pares (chave, valor) enviando mensagens entre si. Qualquer nó participante pode eficientemente recuperar o valor associado a uma dada chave. A responsabilidade de manter o mapeamento de chaves para valores é distribuída entre os nós tal que mudanças no conjunto de participantes causem o mínimo de desordem. Os nós devem manter as seguinte tabelas armazenadas:

(1) Uma tabela de nós (Tabela 1) contendo o endereço IP dos nós no formato IPV4 bem como o respectivo estado ON/OF. Estado "ON" no caso de estar conectado á rede e "OFF" no caso de estar desconectado.

CHAVE	**VALOR**
<IPV4>	<ON/OFF>
Tabela1: Nós	

Deve existir um ficheiro de configuração *nodes.xml*, contendo uma lista com os endereços IPV4 dos diferentes nós. Este ficheiro XML é utilizado para carregar o valor da tabela quando o nó inicia. Qualquer dos nós verifica a cada 10 segundos se houve mensagens "ON" de actualização da tabela por parte dos outros nós. No caso de um dos nós não ter enviado mensagens durante esse período de tempo, é considerado "OFF". Se um nó estiver em modo "OFF" por mais de 60 segundos, é removido da tabela.

(2) Uma tabela de recursos (Tabela 2) contendo o hash do recurso bem como o respectivo URL (Uniform Resource Locator).

CHAVE	**VALOR**
<HASH>	<URL>
Tabela2: Recursos	

Deverá ser utilizada a função hash SHA-1 para gerar o hash correspondente ao recurso. Exemplo de

um hash correspondente a um recurso:

```
415ab40ae9b7cc4e66d6769cb2c08106e8293b48
```

O nó realiza o "checksum" para cada processo de inserção (Create) ou alteração (Update) da tabela e em caso de falha rejeita a operação.

Um URL completo deve seguir a seguinte estrutura:

```
esquema://dominio:porto/caminho/recurso
```

- O esquema é o protocolo. Deverá ser HTTP.
- O domínio é o endereço da máquina: designa o nó que disponibiliza o recurso solicitado. Apenas serão válidos os endereços que constam da tabela de nós.
- O porto é o ponto lógico no qual se pode executar a conexão com o Nó. Deverá ser 8080
- O caminho especifica o local onde se encontra o recurso, dentro do Nó.

Exemplo:

```
http://192.168.10.250:8080/recursos/exemplo.txt
```

5.2 Nó Central

O nó central (NODE0) implementa todos os serviços suportados pelos restantes nós e adicionalmente implementa os serviços necessários para lidar com os clientes. O formato de mensagens / serviços é definido pelo(s) aluno(s) de acordo com o ponto 3. Adicionalmente deve implementar um servidor WebSockets de modo a que os clientes obtenham informação relativa ao estado dos nós. Utilize por defeito o porto **8080** para comunicações HTTP e porto **6060** para comunicações WebSockets. O Nó central implementa um ambiente multi-threading, servindo ao mesmo tempo mais de um nó e mais de um cliente. Este nó funciona como um proxy/cache do sistema CAS. Os clientes não deverão aceder aos recursos por outra via. Ao receber um pedido para inserir um novo recurso (Create) este nó escolhe de modo aleatório o nó que vai aceitar a operação (incluindo ele próprio). Ao receber um pedido para obtenção de um recurso / download (Read) ele trata de obter o recurso (caso não o tenha) e serve o pedido a partir do cache em disco.

Para registo de todas actividades de comunicação deve haver como opção guardar num ficheiro de log, debug.log.

5.3 Cliente

Deve criar uma aplicação "stand-alone" que suporte o acesso a todos os serviços disponibilizados especialmente para o cliente no nó NODE0 de modo a interagir com o sistema CAS. O cliente deve realizar o "checksum" para cada processo de obtenção do recurso e em caso de falha rejeita a operação.

Adicionalmente deve implementar um cliente WebSockets para obter informação relativa ao estado dos nós. Utilize por defeito o porto **8080** para comunicações HTTP e o porto **6060** para comunicações WebSockets. No desenvolvimento da aplicação cliente pode ser utilizada qualquer linguagem de programação. Veja-se na figura 2 um exemplo da execução do cliente.

```
$ echo "OLA CAS" > exemplo.txt

$ java client upload exemplo.txt
79d78012ef221104629f60ef2c114108df6dd1a8

$ rm exemplo.txt
```

```
$ java client list
79d78012ef221104629f60ef2c114108df6dd1a8
da4b9237bacccdf19c0760cab7aec4a8359010b0
77de68daecd823babbb58edb1c8e14d7106e83bb
1b6453892473a467d07372d45eb05abc2031647a

$ java client download 79d78012ef221104629f60ef2c114108df6dd1a8
exemplo.txt
Checksum OK

$ cat exemplo.txt
OLA CAS
```

Figura 2: Exemplo de execução de um cliente

6. Interface gráfica

Exceto na utilização de um cliente browser para acesso aos serviços CAS ou WebSocket, a solução deve utilizar simplesmente linha de comando. O desenvolvimento de uma interface gráfica não será bonificado.

7. Tratamento de excepções

A solução deverá ser tolerante a falhas temporárias de curta duração na ligação do cliente. Ou seja, se a rede ou o NODE0 ficarem indisponíveis por alguns instantes, a aplicação cliente irá receber uma excepção. Esta excepção deverá ser tratada do lado do cliente de forma a tentar de novo a ligação. O utilizador não pode visualizar a ocorrência de excepções na aplicação. Quando não existe conectividade com o NODE0, o cliente deverá simplesmente indicar que aguarda ligação ao NODE0. Se o cliente recebe uma excepção na comunicação com o NODE0, deve haver novas tentativas (**5** tentativas em **5** segundos) para conectar novamente. Se depois das tentativas o NODE0 não estiver disponível, o cliente termina.

8. Relatório

Para além do software desenvolvido deverá ainda escrever um relatório do trabalho prático. O relatório deverá ter os seguintes tópicos:

- Introdução
- Arquitectura da solução
- Tratamento de falhas transitórias
- Manual de utilização
- Manual de instalação e configuração
- Descrição dos testes efectuados à aplicação

9. Entrega

O prazo para a entrega do trabalhos é de acordo com as datas publicadas no Moodle. Não serão

aceites trabalhos fora do prazo. A entrega deverá ser feita via Modle num ficheiro zip contendo o relatório (obrigatório) em PDF e as listagens dos programas realizados devidamente comentado, obedecendo OBRIGATORIAMENTE ao seguinte formato: <nr_aluno>-projecto-CD.zip

10. Referências

Apache - Tomcat
http://tomcat.apache.org/

Apache - Maven
http://maven.apache.org/

Apache - Ant
http://ant.apache.org/

Apache – XML-RPC
http://ws.apache.org/xmlrpc/

Eclipse - Jetty
http://eclipse.org/jetty

W3C - Web Services Architecture
http://www.w3.org/TR/ws-arch/

RESTful Web Services in Java
https://jersey.java.net/

Build RESTful web services with Java technology
http://www.ibm.com/developerworks/training/kp/j-kp-rest/index.html

Java Server Pages Technology
http://www.oracle.com/technetwork/java/javaee/jsp/index.html