# Web Proxy Cache Server

Daniel Carvalho e Ricardo Branco

University of Minho, Department of Informatics, 4710-057 Braga, Portugal
e-mail: {a61008,a61075}@alunos.uminho.pt

**Abstract.** Este artigo reflecte a implementação e a descrição do funcionamento de um servidor de *"proxy"* com cache na linguagem de programação Java. Nesta implementação recorre-se ao uso de "threads" de forma a ser possível tratar cada pedido independentemente mantendo controlo sobre a concorrência entre pedidos ao servidor e sobre o armazenamento na cache. Para além disso o servidor suporta também outro tipo de objectos que não somente páginas HTML.

## 1 Introdução

No âmbito da unidade curricular de Redes de Computadores foi proposta a implementação de um servidor de "proxy" com cache, com suporte a vários utilizadores em Java que suportasse vários objectos que não só HTML. Neste nível tornam-se importantes os conhecimentos obtidos tanto na própria unidade curricular como na de Sistemas distribuídos onde a aprendizagem passa também por manipulação de software cliente-servidor na sua vertente de computação distribuída. Após a análise do código disponibilizado, este foi completo e melhorado de forma a proporcionar o funcionamento desejado.

## 2 Completamento de código ("Fill-in")

Na sua maioria a necessidade de completar o código passava pela análise de variáveis de manipulação de dados já definidas no código fornecido e pela interpretação e criação de variáveis Socket e SocketServer para tornar possível a criação de ligações entre o cliente e o servidor de "proxy" e entre o servidor de "proxy" e o servidor que consta no pedido do cliente. Tornou-se também necessária a análise e interpretação dos pedidos GET feitos pelo cliente e das respostas dos servidores. Por fim e num nível mais avançado foram implementadas as funcionalidades de cache e threads que se falarão nas secções seguintes. Exemplos de completamento:

```
socket = /* Fill in */;
// After
socket = new ServerSocket(port);

request = /* Fill in */
// After
request = new HttpRequest(fromClient);

DataInputStream fromServer = /* Fill in */;
response = /* Fill in */;
DataOutputStream toClient = /* Fill in */;
/* Fill in */
// After
DataInputStream fromServer =
    new DataInputStream(server.getInputStream());
response = new HttpResponse(fromServer);
```

```
      DataOutputStream toClient =
          new DataOutputStream(client.getOutputStream());
      ProxyCache.caching(request, response);
      toClient.writeBytes(response.toString());
      toClient.write(response.body);
```

### 2.1 Cache

No que diz respeito à implementação de um sistema de cache no servidor, recorreu-se ao uso de estruturas de dados do Java e a ficheiros de forma a fazer corresponder cada URI específico ao seu ficheiro de cache criado após ser estabelecida uma ligação entre o servidor designado pelo dito URI e o servidor de "proxy". Assim sendo, é criada uma directoria designada por Cache na directoria de execução do software servidor onde são armazenados os ficheiros obtidos ao fazer "caching" dos servidores constantes nos pedidos dos clientes. A estrutura de dados que permite a cada URI fazer corresponder o seu ficheiro em cache é a HashTable, os motivos do seu uso serão explicados na secção seguinte. Código demonstrativo:

```
private static Map<String, String> cache = new Hashtable<String, String>();
public synchronized static void caching
            (HttpRequest pedido, HttpResponse resposta) throws IOException{
     File ficheiro;
     DataOutputStream paraficheiro;

     ficheiro = new File("cache/","cached_"+System.currentTimeMillis());
     paraficheiro = new DataOutputStream( new FileOutputStream(ficheiro));
     paraficheiro.writeBytes(resposta.toString()); /* Escreve headers */
     paraficheiro.write(resposta.body); /* Escreve body */
     paraficheiro.close();
     cache.put(pedido.URI, ficheiro.getAbsolutePath());
     System.out.println("Caching from: "+pedido.URI+
             " para "+ficheiro.getAbsolutePath());
    }
public synchronized static byte[] uncaching
        (String uripedido)   throws IOException{
   File ficheirocached;
   FileInputStream deficheiro;
   String hashfile;
   byte[] bytescached;

   if((hashfile = cache.get(uripedido))!=null){
   ficheirocached = new File(hashfile);
   deficheiro = new FileInputStream(ficheirocached);
   bytescached = new byte[(int)ficheirocached.length()];
   deficheiro.read(bytescached);
   System.out.println("Caching: Hit on "+uripedido
         +" returning cache to user");
   return bytescached;
   }
   else {
   System.out.println("Caching: No hit on "+uripedido);
   return bytescached = new byte[0];
   }
}
```

### 2.2 Threads

De forma a suportar os vários pedidos de cliente e conseguir controlar a concorrência entre eles recorreu-se ao uso de Threads. Assim sendo cada thread trata de processar o pedido de cada cliente associado bem como a resposta a esse cliente. Assim sendo também o "caching" de páginas é feito por cada thread assim como a verificação de "cache hits" são feitos por cada thread individualmente. Tendo em conta o uso de threads optou-se por usar a estrutra de dados do Java, Hashtable que já é por definição "synchronized" ou seja, preparada para assegurar o controlo de concorrência na sua própria implementação. Não obstante também os métodos de "caching" e "uncaching" são "synchronized".

```
// Classe ProxyCache
  client = socket.accept();
  (new Thread(new Threads(client))).start();

// Classe Threads
public class Threads implements Runnable{
private final Socket client;

    public Threads(Socket client) {
        this.client = client;
    }

    public void run() {
Socket server = null;
HttpRequest request = null;
HttpResponse response = null;

    // Código de tratamento de client requests e server responses
    }
}
```

## 3 Conclusões

Neste projecto foi necessário tomar algumas decisões já descritas e explicadas durante o artigo, sendo as mais importantes relacionadas com o funcionamento e armazenamento da cache assim como o uso de threads no controlo das ligações dos clientes ao servidor para tratamento dos HTTP requests e responses. Foi possível ver o funcionamento ao nível protocolar do HHTP, como a forma como os pedidos são feitos assim como as respostas, bem como a interacção entre várias máquinas mediadas por esse protocolo.