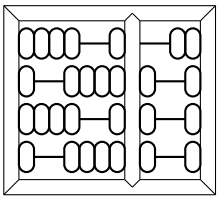
UNIVERSIDADE ESTADUAL DE CAMPINAS

INSTITUTO DE COMPUTAÇÃO



MC833 - Relatório Científico 1

Comunicação Cliente-Servidor Usando Protocolo TCP

135368: Cristina Freitas Bazzano
135749: Flávio Altinier Maximiano da Silva

Campinas - SP
Abril de 2015

Sumário

Resumo

Este trabalho focou-se no estudo temporal de redes cliente-servidor baseadas em comunicação utilizando Java RMI. Foi implementado um serviço de locadora de filmes baseado em *MySQL* no servidor, e ao cliente foram adicionadas diversas operações de acesso ao banco, baseando-se em operações de consultas pequenas, grandes e de escrita. Observou-se que a escrita no banco é o processo mais lento, enquanto operações de consultas extensas são apenas brevemente mais lentas que buscas mais curtas. Comparamos os resultados de tempo desse processo utilizando RMI com os obtidos em um trabalho anterior baseado em TCP diretamente programado na linguagem C. Apesar de ser esperado que as operações em RMI fossem mais lentas, observamos exatamente o contrário; culpamos esse fato à melhor adaptação do MySQL ao Java do que ao C. Além disso, realizamos uma breve análise do tamanho de código e do nível de abstração.

1 Introdução

Na Internet, a grande maioria das montagens de comunicação baseia-se em um sistema cliente-servidor. Nesse contexto, o servidor guarda alguma informação que é relevante ao cliente, que pede essa informação através de uma mensagem na rede; o servidor, por sua vez, responde esse pedido com outra(s) mensagem(ns) contendo a informação requisitada.

O método de comunicação escolhido para este projeto é o *Remote Method Invocation* (RMI) [1, 2]. Esse método, desenvolvido utilizando a linguagem Java, é baseado em comunicação TCP [3] (sem perda de pacotes) e baseia-se em um host (o cliente) ser capaz de invocar métodos do outro (servidor) como se fossem métodos locais. Toda a comunicação TCP é feita "escondida" pelo sistema.

Neste trabalho, num primeiro momento, o foco principal de nossa análise será a medida de tempo total de comunicação entre cliente e servidor, além de uma aproximação do tempo de transmissão das mensagens entre os dois (tempo de as mensagens saírem de um host e chegarem ao outro). Faremos também uma comparação desses tempos com os obtidos em um trabalho anterior, no qual estudamos a implementação direta do protocolo TCP em C. O tamanho de código será também analisado, além de um breve parecer sobre o nível de abstração do sistema desenvolvido.

2 Descrição Geral do Sistema

O sistema construído consiste em uma simples montagem de cliente-servidor, com comunicação utilizando RMI.

O sistema montado é uma emulação de um serviço de locadora online, onde o servidor contém diversas informação sobre os filmes que possui em seu estoque, os quais podem ser acessadas por clientes ou pelo cliente administrador da locadora.

Para sintetizar o problema da melhor forma possível, criamos no nosso banco de dados uma tabela "Locadora" que armazena os seguintes atributos sobre os filmes que possui: título, ano de lançamento, gênero, duração, sinopse, diretor, ator principal, ator coadjuvante principal, exemplares e um id único para cada filme.

Os acessos do cliente ou do administrador da locadora ao servidor são com pedidos de informação ou escrita de dados de cada filme, sendo eles da seguinte forma:

1. Listar todos os títulos dos filmes e o ano de lançamento;
2. Listar todos os títulos dos filmes e o ano de lançamento de um gênero determinado;
3. Dado o identificador de um filme, retornar a sinopse do filme;
4. Dado o identificador de um filme, retornar todas as informações deste filme;
5. Listar todas as informações de todos os filmes;
6. Alterar o número de exemplares em estoque;
7. Dado o identificador de um filme, retornar o número de exemplares em estoque.

Todos esses acessos podem ser feitos em dois modos de consulta: um que realiza a consulta apenas uma vez, para testar sua corretude, e outro que realiza a consulta sequencialmente, cada acesso é repetido 30 vezes e todos os tempos são armazenados em um arquivo *log*, para possibilitar uma futura análise dos mesmos.

Classificamos esses acessos em 3 categorias: as consultas pequenas, consultas grandes e operações de escrita. De acordo com a consulta que cada operação realiza podemos dividi-las da seguinte forma: no primeiro grupo se encontram as operações de 1 a 4 e a operação 7, no segundo grupo temos apenas a operação 5 e no terceiro grupo temos a operação 6.

O primeiro grupo é caracterizado por uma consulta rápida de até 1500 bytes, que é o tamanho máximo MTU (Maximum Transmission Unit) que um pacote pode ter para ser transmitido pela camada Ethernet. O segundo grupo realiza consultas longas, com mais de 1500 bytes, que devem ser quebradas em mais de um pacote para serem enviadas. E o terceiro grupo é caracterizado por uma consulta que carrega um maior atraso do banco de dados, uma vez que no tempo medido está embutido o tempo de escrita no banco.

3 Armazenamento e Estruturas de Dados

Em um sistema real de cliente-servidor os dados são em sua maioria armazenados em Banco de Dados para garantir a persistência e a consistência dos mesmos. Com o objetivo de nos aproximar o máximo possível de um problema real, adotamos o MySQL [4] como o Banco de Dados para armazenar os dados da locadora de filmes no servidor.

Populamos o banco do servidor com dados de 10 filmes retirados do site IMDb [5], os quais estão classificados com as 10 melhores notas no mesmo. Optamos por uma quantia pequena de exemplares para que os tempos medidos nas operações não fossem totalmente influenciados pelo tempo de acesso ao servidor, visto que os tempos de comunicação são o foco da análise do projeto.

No acesso ao banco de dados, os clientes podem apenas acessar a informação, enquanto o cliente administrador da locadora pode, além do acesso, alterar a quantidade de exemplares em estoque. Para garantir esse privilégio de acesso, é necessário uma senha para entrar no modo administrador, a qual é processada pelo programa e só após confirmada o acesso diferenciado ao banco pode ocorrer.

Com o objetivo de fazer um sistema robusto, que garante a confiabilidade do mesmo, optamos por utilizar uma cópia do banco de dados do servidor no lado do Cliente, possibilitando assim uma conferência dos dados retornados pelas consultas. Apesar de ser uma manobra simulada, nos permite confirmar a confiabilidade do protocolo TCP, que para todos os nossos exemplos retornou as consultas em ordem, da forma que foram enviadas pelo servidor. E portanto, essa manobra também nos permitirá dizer se o mesmo ocorre quando utilizamos um protocolo sem confiabilidade, que é o caso do UDP.

4 Detalhes de Implementação

Tanto o cliente quanto a servidor foram implementados baseando-se nas descrições encontradas em [2]. Porém note que métodos do cliente não são necessários estar registrados em RMI, já que ele apenas precisa consultar o servidor (e não o contrário).

Por esse motivo, optamos por manter apenas as operações do servidor em sua pasta pública. Além disso, como no nosso sistema apenas strings são enviadas e recebidas, pudemos remover totalmente a classe *Task.java* e tornar a interface *Compute.java* menos robusta: agora, ele recebe uma string como parâmetro (a query SQL) e retorna outra string (o resultado da operação).

O servidor é iniciado em uma máquina utilizando todas as operações demonstradas no Tutorial [2]. Como especificado, sua interface RMI é apenas a função Compute (que recebe uma query SQL), cuja função é executar a query no banco de dados de filmes e

retornar os resultados.

O cliente, a cada vez que é iniciado, joga na tela um *prompt* com as opções de requisição ao servidor. Em seguida, após escolhida a opção, o cliente dá a opção de fazer o acesso apenas uma vez (para verificar a corretude da aplicação) ou trinta vezes. Se o usuário escolher por fazer o acesso apenas uma vez, o resultado da consulta é exibido na tela do cliente. Se escolher por fazer a consulta trinta vezes, um arquivo de *log* de respostas é criado na mesma pasta do cliente, com o nome *clientTime.txt*.

A consulta é feita ao servidor pelo meio de envio de consultas *MySQL*, que são montadas no próprio programa do cliente. Essas consultas são enviadas como parâmetros da função Compute, presente na interface RMI do servidor, que simplesmente as aplica ao seu banco de dados e retorna o resultado das consultas de leitura ou escrita, como retorno da função Compute.

Esse arquivo de *log* gerado tem o seguinte formato: são trinta linhas e uma coluna, onde cada linha representa um dos testes. Os valores numéricos são os tempos, em nanossegundos, de cada chamada à função Compute pelo cliente.

O servidor também concatena a um arquivo de *log* dados sobre cada conexão que coordena. Como o servidor concatena informações num mesmo arquivo, quando o cliente fizer, por exemplo, trinta consultas de títulos de filmes, as últimas trinta entradas desse *log* do servidor guardarão os dados sobre essas consultas. Cada entrada nesse *log* é um *inteiro* que guarda o tempo de acesso do servidor ao banco de dados; esse tempo é o intervalo desde o início da execução da função Compute até imediatamente antes de seu retorno. Dessa forma, podemos aproximar o tempo de transmissão das mensagens: usando o tempo $tServer$ desse *log* do servidor e a segunda coluna do *log* do cliente ($tClient$), o tempo de transmissão de uma mensagem ao outro será, aproximadamente, dado por:

$$\text{Tempo de Transmissão} = \frac{tClient - tServer}{2} \tag{1}$$

Um outro pequeno detalhe de implementação é que apenas o cliente administrador da locadora deve ter autorização para alterar o número de exemplares em estoque de cada filme. Portanto, quando um cliente decide fazer essa alteração, é perguntado por uma senha. Essa senha é verificada no próprio programa do cliente (é hard-coded como “1234”) e, apenas se estiver correta, o programa continua. Essa escrita no banco de dados também é hard-coded para agilizar os testes (não é necessário perguntar o novo estoque a cada uma das trinta execuções.)

5 Análise dos Resultados

5.1 Tempo total de Comunicação

Em todos os resultados dos tempos médios utilizamos um intervalo de confiança com nível de confiança de 95%, obtido a partir da soma ou subtração de dois desvios padrão.

Primeiramente geramos um gráfico com o tempo total de conexão com o servidor para cada uma das operações. O resultado segue demonstrado na Tabela 1 e na Figura **??**.

Tabela 1: Tabela mostrando o tempo médio e intervalo de confiança para cada uma das sete operações, na ordem em que foram descritas. Tempo calculado no cliente, desde a abertura até o fechamento da conexão.

Operação	Tempo Médio Total (ms)	Intervalo de Confiança (ms)
1	38.42	5.59
2	37.38	9.14
3	37.47	7.59
4	37.10	10.37
5	40.41	9.15
6	114.10	95.27
7	36.49	7.88

Os resultados que obtivemos para o tempo total de conexão dependem bastante das consultas ao banco de dados e do tamanho das mensagens. Esses mensagens nas operações pequenas, como já especificado, podem ser totalmente enviadas em apenas um pacote, enquanto nas operações grandes, vários pacotes são necessários para o envio completo das mensagens. Nas operações de escrita, apesar de só um pacote ser necessário para o envio da mensagem, a consulta ao banco de dados para alteração requer um tempo de conexão bem maior do que as consultas de informação. É interessante notar que o desvio da operação de escrita também é consideravelmente maior do que o das outras operações; acreditamos porém que essa variação é causada pelo atraso do banco de dados, e não apenas da rede.

Com isso em mente, os resultados foram como esperado. As operações pequenas estão todas na mesma faixa e a operação grande está apenas um pouco acima destas, visto que a única diferença entre essas duas categorias é a quantidade de pacotes recebidos, que causou um pequeno aumento no tempo medido. Já a operação de escrita apresenta uma grande diferença em relação as demais; o acesso ao banco para alteração de valores gera um *overhead* muito grande nesse medida, que acaba representando bem mais o tempo de alteração do banco de dados do que o tempo de conexão com o servidor.

5.2 Tempo de Transmissão

Em seguida geramos um gráfico com o tempo total de transmissão das mensagens para cada uma das operações, como explicitado pela equação 1. O resultado segue na Tabela 2 e Figura **??**.

Os tempos de transmissão medem o tempo de comunicação entre o cliente e o servidor. Esse tempo deve depender apenas do tamanho das mensagens, que como já especificado, corresponde apenas a um pacote nas operações pequenas e de escrita e de vários pacotes nas operações grandes.

Os tempo obtidos para as operações pequenas e grandes condizem com essa expectativa; o tempo de transmissão da operação grande é levemente maior que o das pequenas. O tempo de transmissão da operação de escrita, porém, fica muito acima do esperado. Assumimos este ser um erro de medida: dado que isto é uma aproximação, o tempo total pode ter influenciado um atraso maior no tempo de transmissão.

Tabela 2: Mostra o tempo médio e intervalo de confiança de 95% para transmissão de mensagens segundo a equação 1 para cada uma das operações, na ordem em que foram descritas.

Operação	Tempo Médio de Transmissão (ms)	Intervalo de Confiança (ms)
1	11.02	2.86
2	10.35	4.09
3	10.79	3.30
4	10.43	4.61
5	11.99	4.45
6	24.19	48.88
7	10.37	3.66

5.3 Tamanho de Código

As operações de redes são, na verdade, parte pequena do código total da aplicação. No servidor, a parte mais extensa do código são claramente as operações de consulta *MySQL*. No cliente, além das operações de banco de dados, há dezenas de linhas voltadas apenas para a saída na tela dos *prompts* ou a escrita em arquivo.

As operações de redes, mesmo no servidor que faz algumas operações a mais que o cliente, representam muito pouco do código. Considerando-se o servidor, as operações de redes mais os seus testes de erros somam 51 linhas, enquanto o arquivo todo tem 191 (correspondem a cerca de 26%). Para o cliente, as operações de redes representam 31 de um total de 342 linhas (cerca de 9%).

5.4 Confiabilidade

Como o protocolo escolhido foi o TCP, que garante confiabilidade na transmissão de mensagens, a confiabilidade esperada era de 100%. De fato, todas as consultas feitas foram exatamente iguais às geradas localmente para contra-prova, comprovando experimentalmente a confiabilidade do protocolo.

5.5 Nível de Abstração

As operações de redes fornecidas pelo sistema são bastante abstratas, apesar de algumas exigirem o entendimento de ponteiros e espaços de memória, meros detalhes da linguagem C. O "diálogo"entre cliente e servidor é resolvido com mensagens, e as operações são tão abstratas quanto imaginamos: baseiam-se em processos de envio (*send()* e *recv()*). Mesmo operações exclusivas a servidores, como *bind()* e *listen()*, embora não sejam tão instintivas, uma vez entendido o protocolo, fazem sentido e tem boa abstração.

Em suma, é possível perceber que toda a complexidade do protocolo está escondida atrás de operações simples.

6 Conclusões

No sistema cliente-servidor implementado neste projeto a informação que o servidor retém é a do seu banco de dados e todas as mensagens se baseiam em consultas *MySQL*. O tempo total está totalmente influenciado pelo tempo de acesso ao banco, como fica visível na operação de escrita.

Devido a característica desta aplicação apenas uma pequena parte do código do sistema realiza as operações de redes. Apesar disso, essas operações são carregadas pela sua abstração, onde toda a complexidade esta embutida apenas em um comando básico. A abstração condiz com o tempo de execução dessas operações, medidos no tempo de transmissão, que equivale a menos de um quarto do tempo total na maioria das operações.

A confiabilidade garantida pelo protocolo TCP utilizado na comunicação do sistema foi comprovada em todas as consultas. Um futuro trabalho será comparar esses resultados com um sistema que utiliza o protocolo UDP, onde a confiabilidade pode não ser tão perfeita.

Referências

[1] Troy Bryan Downing. *Java RMI: remote method invocation*. IDG Books Worldwide, Inc., 1998.

[2] The Java Tutorials. Trail: Rmi., 2015.

[3] Jon Postel. Transmission control protocol. 1981.

[4] AB MySQL. *MySQL: the world's most popular open source database*. MySQL AB, 1995.

[5] Internet Movie Database. Top 250 on imdb: Highest-rated movies based on votes by imdb users., 2015.