FACULDADE NATALENSE PARA O DESENVOLVIMENTO DO RIO GRANDE DO NORTE

Atualização de *Web Services* em tempo real

Vinícius Brandão Mendes

Orientador: Prof. Ricardo Wendel

Relatório técnico apresentado ao Curso de Especialização em Desenvolvimento de Sistemas Corporativos da FARN como trabalho de conclusão de curso.

Natal, RN 2011

Atualização de *Web Services* em tempo real

Vinícius Brandão Mendes

Relatório técnico aprovado em ____ / ____ / _____ pela banca examinadora composta pelos seguintes membros:

Prof. Ricardo Wendel (Orientador) . . . . . FARN

Prof. . . . . FARN

Prof. . . . . FARN

Agradecimentos

Ao meu orientador, professor Ricardo Wendel, sou grato pela orientação.

Aos meus pais, Joalmi Mendes de Oliveira e Maria Gleide Brandão Mendes, pelo apoio total durante a execução do trabalho e durante toda a minha vida, com educação, instrução e companheirismo.

À minha noiva Rosana Curvelo de Souza pelo constante incentivo.

Ao colega Fábio Miranda Costa pelo apoio durante o curso.

Aos meus irmãos, familiares e amigos pela paciência e apoio dispensados durante a execução do trabalho.

Resumo

Neste trabalho será relatado o desenvolvimento de um cliente de *Web Service* baseado em SOAP (*Simple Object Access Protocol*) que deve realizar várias requisições para sincronizar uma grande massa de dados em um tempo relativamente curto. Foram abordadas três técnicas: requisições sob demanda do usuário; atualizador com requisições síncronas e sequenciais em um único processo; e atualizador com requisições síncronas em vários processos.

Palavras-chave: *Web services*, SOAP, computação distribuída, programação concorrente.

Abstract

On the following work, will be reported the development of a Web Service client based on SOAP that should make many requests to synchronize a huge data in a short time. Three techniques were considered: request on user demand; updater with synchronous and sequential requests in a single process; and updater with synchronous requests with multiprocessing.

Keywords: Web Services, SOAP, distributed computing, multiprocessing.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Este trabalho trata de um serviço que deve servir seus usuários os mais rapidamente com cotações geradas por uma fonte de dados externa, que as fornece através de um *Web service* SOAP. Existem inúmeras formas de resolver esse problema. Neste trabalho serão abordadas algumas técnicas possíveis.

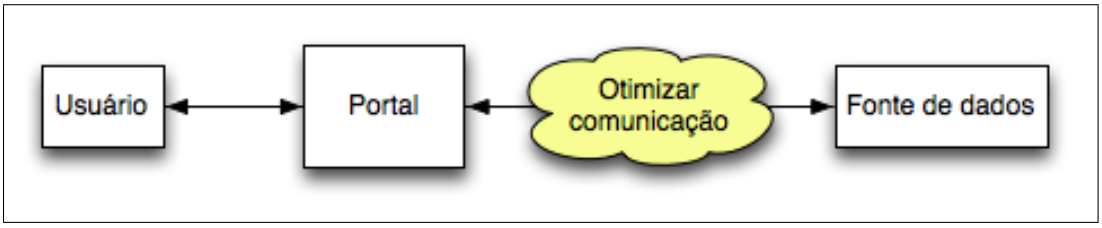

Figura 1.1: Arquitetura básica do problema em questão

1.1 Descrição geral

O trabalho é baseado em um *Web Service* SOAP, que é um protocolo para troca de informações estruturadas em uma plataforma descentralizada e distribuída.

Um portal de internet precisa oferecer a seus usuários em tempo real dados de um fornecedor que os disponibiliza através de um *web service*. Para não ter atraso na informação oferecida, ele precisa requisitar ao fornecedor de dados o mais frequentemente possível ao mesmo tempo que tem que manter um bom tempo de resposta para seus usuários.

1.2 Objetivo

O objetivo do trabalho é resolver os problemas relacionados à otimização do tempo de resposta e à frequência de atualização do dado. Por se tratar de um problema de otimização de tempo de resposta é necessário pensar na latência e no *overhead* gerado por uma conexão HTTP e analisar a melhor arquitetura de sistema possível.

1.3 Motivação

Com o avanço da informática e de conceitos como a computação em nuvem, cada vez mais os servidores estão em constante comunicação ao redor do mundo. Seja para autenticar usuários, seja para persistir dados ou seja para solicitar dados. Um portal de notícias, por exemplo, exibe, além das notícias, dados meteorológicos e econômicos. Esses dados normalmente não são produzidos pelo próprio portal, portanto são necessárias parcerias com fornecedores de conteúdo que disponham de tal informação. Estes fornecedores, em geral, disponibilizam os dados através de *Web Services*.

Dado que no mundo jornalístico a agilidade na entrega de informações é um fator primordial para o sucesso, surge a necessidade de otimizar o tempo de consumo de tais fornecedores a fim de entregar aos usuários a informação mais recente possível.

1.4 Metodologia

O projeto foi dividido em quatro partes:

- estudo sobre *web services* e o protocolo SOAP;
- estudo sobre computação paralela;
- implementação de cliente para *web services* SOAP;
- implementação de estratégias de utilização do cliente.

1.4.1 Estudo sobre *web services* e o protocolo SOAP

Por se tratar de atualização de *web services* foi necessário um estudo sobre os conceitos desta tecnologia.

1.4.2 Estudo sobre computação paralela

O projeto tem como pré-requisito oferecer um baixo tempo de resposta. A computação paralela é um paradigma interessante para otimizar tarefas computacionais e explorar a máquina o máximo possível.

1.4.3 Implementação de cliente para *web service* SOAP

Feito o estudo, foram colocados em prática os conhecimentos a fim de implementar uma solução que possibilitasse o acesso a *web services* que utilizem o protocolo SOAP.

1.4.4 Implementação de estratégias para utilização do cliente

Com uma forma de acessar os dados, foram elaboradas duas estratégias para otimizar o tempo de resposta ao usuário final e garantir a recentidade dos dados apresentados.

Capítulo 2

Web Services

Web service é uma interface acessível através da rede para uma funcionalidade de uma aplicação construída usando tecnologias definidas como padrão na Internet [Doug Tidwell & Kulchenko 2001]. É apenas mais uma camada de troca de mensagens entre uma aplicação e outra. A principal vantagem no seu uso é prover comunicação entre diferentes aplicações independente da plataforma ou linguagem de programação utilizada por elas, o que garante interoperabilidade entre sistemas desde que ambos utilizem o mesmo protocolo de comunicação entre si.

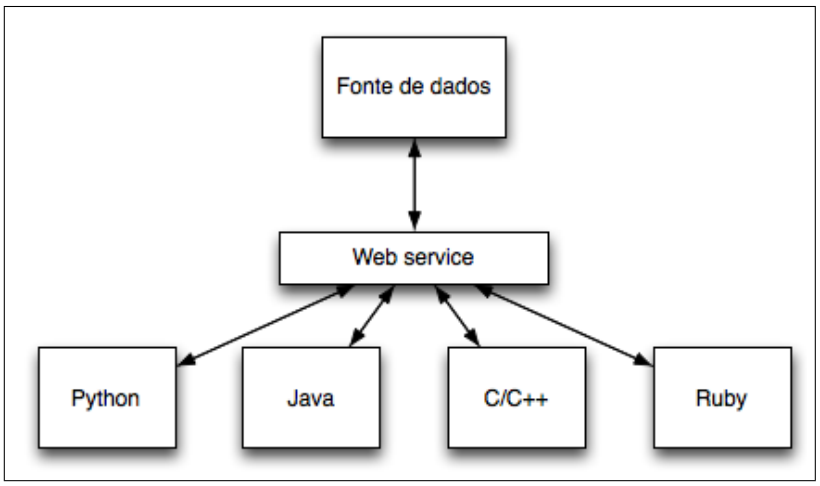

Figura 2.1: *Web services* poder servir a diferentes plataformas

Através do uso de *web services* é possível fazer com que uma aplicação faça chamadas de métodos remotos de outra aplicação. É simples como gerar uma requisição HTTP que encapsule qual método será chamado e quais os parâmetros e esperar uma resposta com o resultado da chamada do método. Para isso se faz necessário quebrar a camada de aplicação da pilha de camadas do modelo de redes TCP/IP em quatro camadas: aplicação, descoberta, descrição e empacotamento.

Figura 2.2: Camadas de um *web service*

2.1 Aplicação

A camada de aplicação é o código que precisará se comunicar com a outra aplicação através do *web service*, que pode ser escrito em qualquer linguagem.

2.2 Descoberta

Esta camada é responsável por disponibilizar metadados sobre *web services* de modo a facilitar a busca pela aplicação necessária para cada projeto.

2.3 Descrição

A descrição define como o *web service* deve ser utilizado. Quais os métodos que ele expõe, quais os parâmetros necessários em cada um deles e quais as possíveis respostas. O *Web Service Description Language* (WSDL) é um padrão muito utilizado para descrever um *web service* através do uso de XML.

2.4 Empacotamento

Trata do empacotamento das mensagens de forma que ambas as partes sejam capazes de entendê-las. Existem vários formatos de empacotamento, em sua maioria baseados em XML (*Extensible Markup Language*), por possibilitar a representação do significado do dado e existirem *parsers* XML para praticamente todas as linguagens e plataformas do mercado. Neste relatório iremos abordar o protocolo de empacotamento SOAP que foi utilizado na implementação do projeto.

2.4.1 SOAP

SOAP (*Simple Object Access Protocol*) é um protocolo cujo objetivo é trocar informações estruturadas em um ambiente distribuído e descentralizado [Gudgin et al. 2007]. É um padrão reconhecido e desenvolvido pelo World Wide Web Consortium (W3C) para desempenhar o papel da camada de empacotamento de um *web service*. Sua especificação define um envelope baseado em XML para conter a informação a ser transmitida e uma série de regras para traduzir dados específicos de uma de plataforma e aplicação para representações XML.

Capítulo 3

Computação Paralela

Computação paralela é uma forma de computação em que vários cálculos são realizados simultaneamente [Lorin 1990]. A maneira tradicional de executar software é a sequencial, o que faz com que um passo de execução só se inicie após a conclusão do passo anterior. Quando se tem apenas uma unidade de processamento esse código é ótimo na maioria dos casos. Já para múltiplas unidades de processamento, o que hoje já é realidade até mesmo em computadores pessoais, o processo sequêncial usaria apenas uma das unidades enquanto as outras ficariam ociosas. A situação se agrava ainda mais quando o processo precisa esperar alguma tarefa como entrada e saída de dados (I/O), pois estas tarefas demandam tempo e atrasariam a execução do processo enquanto a unidade de processamento ficaria ociosa.

A proposta da computação paralela é reduzir o tempo de resposta de um software através da divisão do processamento entre as unidades disponíveis. Em um problema baseado em I/O como a atualização de dados através de um *web service* ficam ainda mais claros os benefícios da computação paralela.

O procedimento de solicitar dados deve ser repetido diversas vezes alterando apenas o parâmetro de entrada. O processo de atualização faz chamadas de I/O ao *web service* e fica ocioso esperando a resposta. Neste momento a unidade de processamento pode processar outra atualização que fará o mesmo procedimento. O que caracteriza um *pool* de processos coordenados por um processo pai que apenas atribui parâmetros de entrada a cada processo.

3.1 *Pool* de processos

Um *pool* de processos é um grupo de processos à disposição de um pai para executar tarefas semelhantes. Geralmente o pai tem um conjunto de parâmetros de entrada e os distribui para cada processo do *pool*, o qual assim que finalizar seu processamento, informa o pai, que se ainda não tiver endereçado todos os parâmetros de entrada, solicita que mais um seja processado.

Capítulo 4

Implementação

A implementação foi feita utilizando a linguagem de programação Python. Foi escolhido um cliente para o protocolo SOAP e encontrada uma API que disponibiliza cotações de moedas através de um *web service* que utiliza este protocolo.

4.1 Por que Python?

Python é uma linguagem interpretada, dinâmica e fortemente tipada. Por ter estas características torna o trabalho mais rápido e diminui o tempo de desenvolvimento do projeto, que no caso é apenas uma prova de conceito, portanto o tempo curto de desenvolvimento é bastante interessante. Isso não significa que Python seja uma linguagem apropriada apenas para provas de conceito, visto que grandes empresas adotam esta tecnologia em seus projetos que estão hoje em produção.

4.2 Cliente SOAP

Para obtenção das cotações do *web service* foi utilizada a biblioteca Suds [Ortel et al. 2009] para fazer a interface entre o código Python e a API externa. A biblioteca reconhece o WSDL e gera classes Python para acesso remoto ao *web service*.

4.2.1 API que disponibiliza cotações

A API disponibiliza um único método para obtenção da cotação de uma moeda em relação a uma outra. Para simplificar a implementação, as cotações foram todas normalizadas para sempre serem em relação ao Dólar Americano.

4.3 Estratégias de utilização do cliente

Foram desenvolvidas duas implementações para o problema com o objetivo de encontrar a mais performática:

- atualização de dados sob demanda;

- atualizador de dados independente de demanda.

4.3.1 Atualização de dados sob demanda

Um das implementações testadas foi conectar à API para obter uma cotação apenas quando o usuário requisitasse. Isso evita a atualização de cotações que não estão sendo requisitadas por usuário nenhum. Em contrapartida, o tempo de resposta ao usuário será alto, visto que será acrescido ao tempo de processamento, o tempo da requisição à fonte de dados, que é através da internet e, portanto, lenta.

Um outro problema neste caso é que a requisição à fonte de dados é bloqueante e será processada pelo mesmo processo utilizado para receber e responder a requisição do usuário, o que faz com que processos que deveriam responder usuários fiquem ocupados com requisições a fontes de dados externas, diminuindo o número de usuários atendidos por segundo.

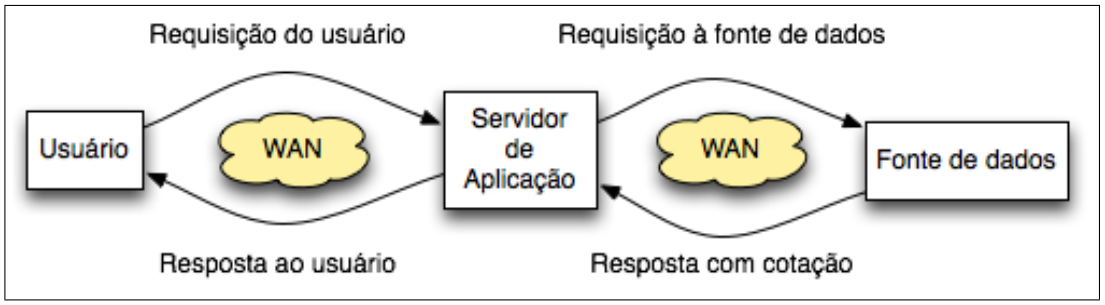

Figura 4.1: Arquitetura da solução com atualização de dados sob demanda.

4.3.2 Atualizador de dados independente de demanda

A segunda implementação testada foi desenvolver um sistema cuja função é requisitar as cotações à fonte de dados em uma frequência determinada e salvá-las em um banco de dados. Desta forma, a cada requisição de usuário, o sistema iria fazer apenas consultas ao banco de dados, que são muito mais rápidas que acessar uma fonte de dados externa.

Outra vantagem é que o banco de dados faz o papel de *cachear* os dados e mesmo que exista falha na comunicação com a fonte de dados, os usuários continuarão recebendo os dados, mesmo que desatualizados.

Em contrapartida, o sistema deve atualizar todas as cotações, mesmo as que não são demandadas tão frequentemente pelos usuários. Isso pode gerar desperdício de poder computacional.

Esse sistema pode ou não utilizar múltiplos processos. Para o caso de um sistema multiprocessado, foi utilizada a biblioteca *multiprocessing*, que é nativa da linguagem Python. No caso, o processo pai obtém a lista de todas as cotações a serem obtidas, e as passa para os processos filhos para que eles façam o trabalho de comunicação com a fonte de dados.

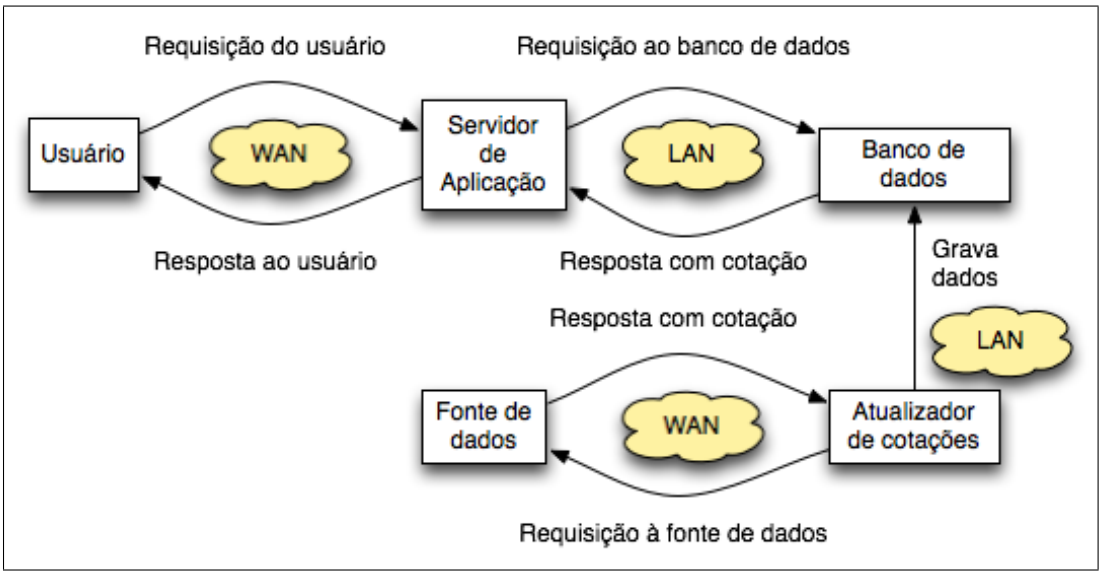

Figura 4.2: Arquitetura da solução com atualizador de dados independente de demanda.

Capítulo 5

Resultados

Para critérios de testes, foi analisada a solução utilizando um atualizador de dados rodando de tempos em tempos. Foram aferidos os tempos necessários para atualizar uma lista com 151 cotações variando o número de processos.

Os testes foram feitos em Macbook Pro com processador Intel Core i5 2.3 GHz de 4 núcleos com 8 GB de memória RAM a 1333 MHz. Como o computador não é dedicado apenas aos testes, existem oscilações referentes a outras tarefas do sistema operacional.

5.1 Requisições paralelas com múltiplos processos

Foram aferidos os tempos de atualização variando o número de processos entre 1 e 16. Para o caso com um processo, a solução se assemelha muito a uma solução sem multiprocessamento. À medida que o número de processos aumenta, o tempo necessário para atualizar todos os dados diminui até o ponto em que o tempo de escalonamento de processos começa a ficar significativo. Um ponto importante a ser observado é que o tempo de processamento efetivo é muito inferior ao tempo total, o que indica que os processadores passam muito tempo ociosos aguardando a resposta da fonte de dados externa.

No. processos	Tempo total (s)	Tempo de usuário (s)	Tempo de sistema (s)
1	138,3	1,01	0,22
2	76,57	1,26	0,28
3	50,61	1,41	0,27
4	40,01	1,57	0,30
5	33,02	1,71	0,33
6	26,48	1,75	0,34
7	24,31	1,92	0,36
8	20,33	1,99	0,37
9	19,87	2,15	0,40
10	16,71	2,25	0,42
11	15,28	2,42	0,44
12	15,60	2,54	0,46
13	11,80	2,63	0,48
14	11,73	2,74	0,51
15	11,19	2,85	0,53
16	11,61	2,96	0,54

Tabela 5.1: Tabela com os tempos aferidos de acordo com o número de processos

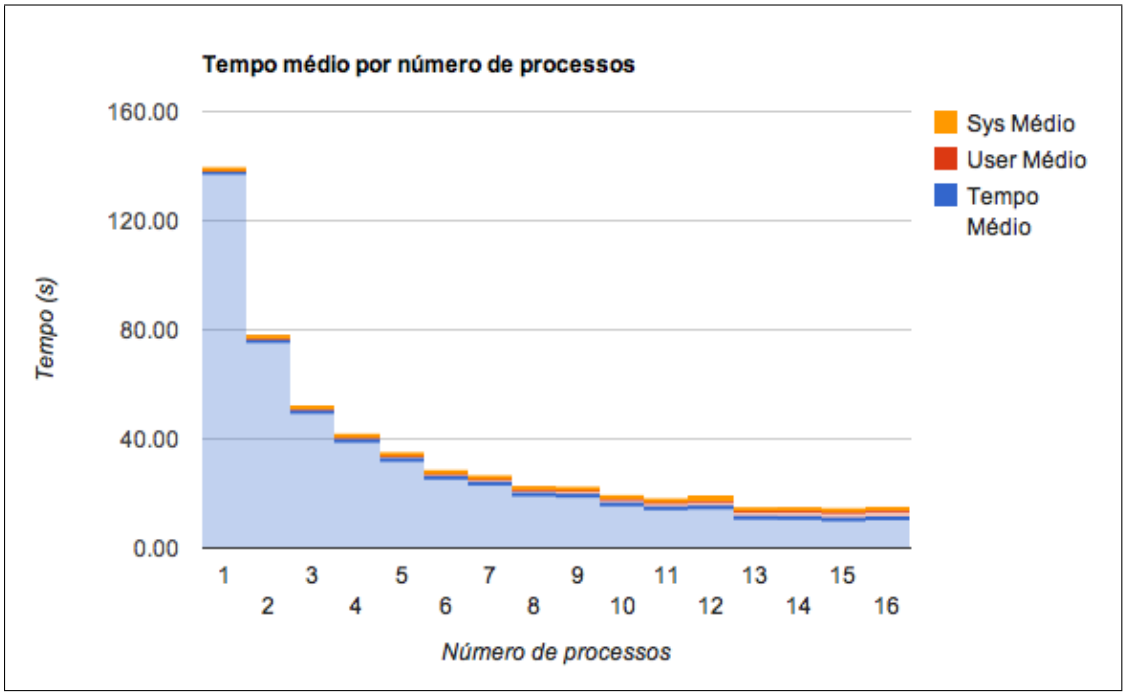

Figura 5.1: Gráfico de tempos aferidos de acordo com o número de processos

Capítulo 6

Conclusão

Com este trabalho é possível concluir que a integração entre sistemas distintos é uma área muito desafiadora, principalmente quando envolve requisitos de atualização em tempo real. Muitas são as possibilidades de arquitetura para otimizar a integração, cabe ao responsável pelo projeto analisar o problema e selecionar a melhor opção.

A escolha desta opção não deve ser apenas teórica. Testes de desempenho são muito importantes e provas de conceito se fazem essenciais.

Referências Bibliográficas

Doug Tidwell, James Snell & Pavel Kulchenko (2001), *Programming Web Services with SOAP*, 1ª edição, O'Reilly.

Gudgin, Martin, Marc Hadley, Noah Mendelsohn, Yves Lafon, Jean-Jacques Moreau, Anish Karmarkar & Henrik Frystyk Nielsen (2007), SOAP version 1.2 part 1: Messaging framework (second edition), W3C recommendation, W3C. http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

Lorin, Harold R. (1990), 'Review of "highly parallel computing"by g. s. almasi and a. gottlieb, benjamin-cummings publishers, redwood city, ca, 1989', *IBM Syst. J.* **29**, 165–166. Reviewer-Lorin, Harold R.
URL: *http://dx.doi.org/10.1147/sj.291.0165*

Ortel, Jeff, Jesper Noehr & Nathan Van Gheem (2009), Suds, Library documentation, W3C. http://fedorahosted.org/suds/.