FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA – UNIFOR

José Airton Fernandes da Silva

IMPLEMENTAÇÃO E AVALIAÇÃO EMPÍRICA DE POLÍTICAS DE INVOCAÇÃO PARA SERVIÇOS WEB REPLICADOS

Fortaleza

2004

FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA – UNIFOR

José Airton Fernandes da Silva

IMPLEMENTAÇÃO E AVALIAÇÃO EMPÍRICA DE POLÍTICAS DE INVOCAÇÃO PARA SERVIÇOS WEB REPLICADOS

DISSERTAÇÃO APRESENTADA AO CURSO DE MESTRADO EM INFORMÁTICA APLICADA DA UNIVERSIDADE DE FORTALEZA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM INFORMÁTICA APLICADA.

Orientador: Prof. Dr. Nabor das Chagas Mendonça

Fortaleza

2004

(folha aprovação)

Silva, José Airton Fernandes da

Implementação e Avaliação Empírica de Políticas de Invocação para Serviços Web Replicados. Fortaleza. Universidade de Fortaleza (UNIFOR). Dissertação de Mestrado. 2004.

90 p.: il. 210 x 297 mm (MIA/UNIFOR, M.Sc. Ciência da Computação)

1. Engenharia de Software
2. Sistemas Distribuídos
3. Serviços Web

AGRADECIMENTOS

A Deus, pela força que sempre esteve comigo neste desafio.

Meus sinceros agradecimentos ao Prof. Nabor das Chagas Mendonça, pela orientação e inestimável ajuda, fundamentais para a conclusão deste trabalho.

Aos professores Pedro Porfírio Muniz Farias e Ricardo de Oliveira Anido, pela presença na banca examinadora.

Agradeço, em especial, à minha esposa Auxiliadora e aos meus filhos Miguel e Amanda, pela paciência e apoio.

Ao professor e colega Arnaldo Dias Belchior, pela amizade e incentivo na realização deste trabalho.

Ao colega Paulo Pereira Jucá, pelo apoio e ajuda na viabilização das condições de infra-estrutura para a condução de parte dos experimentos deste trabalho nas instalações do Banco do Nordeste.

Aos demais professores do mestrado, pelas contribuições indiretas.

À secretaria do MIA, pela atenção e presteza sempre imediatas.

Aos demais colegas aqui não citados que de alguma forma me ajudaram e incentivaram.

Ao Banco do Nordeste do Brasil S.A., por ter patrocinado este projeto.

Resumo da Dissertação apresentada ao MIA/UNIFOR como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática Aplicada.

IMPLEMENTAÇÃO E AVALIAÇÃO EMPÍRICA DE POLÍTICAS DE INVOCAÇÃO PARA SERVIÇOS WEB REPLICADOS

José Airton Fernandes da Silva

Abril / 2004

Orientador: Prof. Dr. Nabor das Chagas Mendonça
Programa: Informática Aplicada

A tecnologia de serviços Web oferece mecanismos que permitem a comunicação interoperável entre aplicações e serviços no ambiente Web. Os serviços Web têm ensejado a busca por mecanismos que assegurem qualidade de serviço (QoS), especialmente quanto à disponibilidade e ao desempenho no acesso aos serviços. Esses indicadores de qualidade têm se tornado uma exigência em função da quantidade de informações e serviços atualmente disponíveis. A replicação de recursos, de forma geograficamente distribuída na Web, tem sido a forma empregada para aumentar a capacidade de atendimento às requisições, tanto com relação à disponibilidade quanto ao desempenho. Quando há um recurso replicado, neste ambiente, um cliente necessita selecionar qual o melhor servidor para atender à sua requisição. Entretanto, a escolha do melhor servidor pelo cliente não é uma tarefa trivial, uma vez que essa decisão pode ser afetada por vários fatores, tais como a carga de trabalho dos servidores, a latência da rede, e capacidade de conexão do lado do cliente. Nesta dissertação, implementamos e avaliamos empiricamente várias políticas para seleção de servidores provedores de serviços Web replicados no ambiente real da Internet. Os objetivos do trabalho são possibilitar a realização de chamadas às réplicas de um serviço Web que apresentem o melhor desempenho, e dar transparência às aplicações consumidoras do serviço no envio de suas requisições.

Abstract of the Dissertation presented to MIA/UNIFOR as a partial fulfillment of the requirements for the degree of Master of Applied Informatics (M.Sc.).

IMPLEMENTATION AND EMPIRICAL EVALUATION OF INVOCATION POLICIES FOR REPLICATED WEB SERVICES

José Airton Fernandes da Silva

April / 2004

Adviser: Prof. Dr. Nabor das Chagas Mendonça
Program: Applied Informatics

Web Services technologies provide mechanisms that enable interoperability between services and applications in the Web environment. Quality of Service (QoS) requirements, such as high availability and performance, are now a fundamental aspect of Web services provision. These requirements are even more important considering the vast amount of resources and services currently available on the Internet. Resource replication, especially over geographically distributed locations, is a traditional way of improving resource availability and access time. On the client side, resource replication means that the client application may need to select the server with the best performance among those that host a copy of the resource. However, choosing the best server is not a trivial task, as this decision may be affected by a number of factors, such as server workload, network latency and connection bandwidth at the client side. In this dissertation, we implement and evaluate empirically several server selection policies for accessing Web services that are geographically replicated over the Internet. Our main goals are to develop mechanisms that make it possible for the client application to dynamically select those replicas offering the best performance, and to give transparency to the application programmer when invoking a replicated Web service.

ÍNDICE

ÍNDICE

LISTA DE FIGURAS

LISTA DE FIGURAS

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1

Introdução

Este capítulo apresenta as questões que motivaram a realização deste trabalho, como também seus objetivos e sua organização.

1.1 Motivação

Os serviços Web são uma padronização do emergente paradigma de *Computação Orientada a Serviço* (Papazoglou e Georgakopoulos, 2003) para o contexto da Web, constituindo-se num poderoso mecanismo de integração entre aplicações distribuídas, independentemente de linguagem de programação, plataforma de execução e protocolo de comunicação (Cauldwell *et al*, 2001). Mais que simples recursos da Web (documentos HTML, imagens, *scripts*, etc), os serviços Web são aplicações fracamente acopladas que podem ser localizadas e acessadas utilizando tecnologias padronizadas da Internet, tanto entre si quanto a partir de aplicações clientes. Os principais padrões de tecnologias utilizados pelos serviços Web são: SOAP (*Simple Object Access Protocol*) (Box *et al*, 2000), que é um protocolo baseado em XML (Bray *et al*, 2000) para a troca de mensagens entre aplicações; WSDL (*Web Services Description Language*) (Christensen *et al*, 2001), que é uma linguagem para a descrição da interface de serviços Web, também baseada em XML; e UDDI (*Universal Discovery, Description and Integration*) (Bryan *et al*, 2002), que define um padrão para o registro e a localização dinâmica de serviços Web.

Os serviços Web surgem num cenário onde oferecer qualidade de serviço é um fator preponderante para o sucesso das organizações, especialmente com relação à disponibilidade e ao tempo de resposta no acesso aos recursos, dada a concorrência cada vez maior por recursos na Internet (Conti *et al*, 2002a). Nesse contexto, técnicas de replicação podem oferecer uma alternativa atraente, tanto para aumentar a disponibilidade dos recursos, através da redundância de servidores, quanto para reduzir o seu tempo de acesso, através da distribuição da carga de trabalho entre os servidores no atendimento às requisições dos clientes (Berners-Lee *et* al, 1996).

As abordagens para acesso a recursos replicados na Internet estão divididas, de um lado, em mecanismos para a distribuição automática das requisições dos clientes, no lado

dos servidores, e de outro lado, em mecanismos incorporados nos próprios clientes, ou em servidores *proxy*, para explorar formas mais efetivas de acesso a recursos replicados em servidores geograficamente distribuídos. Neste trabalho, focamos nas abordagens que tratam o acesso a recursos replicados do lado do cliente, partindo do pressuposto que as réplicas dos serviços são conhecidas e mantidas em um estado consistente entre si pelos seus respectivos provedores. Portanto, abordagens referentes à alocação e ao gerenciamento das réplicas, tipicamente implementadas do lado do servidor, estão fora do escopo desta dissertação.

Pelo lado do cliente, várias pesquisas (por exemplo, Amini *et al*, 2003; Hanna *et al*, 2001; Dykes *et al*, 2000; Sayal *et al* 1998) já foram conduzidas com foco na avaliação de políticas de seleção de réplicas, no sentido de identificar aquela que possa oferecer o melhor desempenho. Essas pesquisas, porém, se limitaram a avaliar políticas de acesso a recursos na forma de documentos HTML e arquivos de imagens. Embora conceitualmente aplicáveis ao domínio dos serviços Web, na prática essas políticas, da maneira como foram propostas, não permitem uma aplicação direta nesse novo contexto devido às características dos serviços Web, que estão mais próximos ao modelo de execução de objetos distribuídos na Internet do que da simples recuperação de documentos e imagens a partir do sistema de arquivos de um servidor Web.

Mais recentemente, alguns trabalhos (Azevedo *et al* 2003; Keidl *et al*, 2002; Padovitz *et al*, 2003) abordaram a seleção dinâmica de serviços Web, também pelo lado do cliente. No entanto, nenhum desses trabalhos avaliou empiricamente o desempenho oferecido pelas suas políticas de seleção de servidores, nem o ganho que elas propiciariam em relação às alternativas propostas por outros trabalhos similares existentes na literatura.

1.2 Objetivos

Esse trabalho tem como principal objetivo contribuir para melhorar o desempenho das aplicações no acesso a serviços Web geograficamente replicados. Mais especificamente, o trabalho visa explorar o modelo de descrição de serviços oferecido pela linguagem WSDL para tratar a localização das réplicas dos serviços, e propor e avaliar empiricamente alternativas que possam tornar mais eficiente o processo de invocação das réplicas no lado do cliente. Para atender a esses objetivos, apresentamos o *framework* RWS (*Replicated Web Services*) (Silva e Mendonça, 2004), que é uma extensão do *framework* AXIS (Apache, 2002) para a implementação de serviços Web na linguagem Java. O AXIS, assim

como outros *frameworks* para serviços Web existentes, por exemplo, .NET (Microsoft, 2002a) e JWSDP (Sun, 2003), não explora de forma adequada alternativas de invocação para melhorar e tornar transparente o acesso a serviços Web replicados, deixando a cargo da aplicação a decisão sobre qual réplica do serviço invocar.

Em contraste aos trabalhos acima mencionados, o *framework* RWS implementa um processo transparente de invocação de serviços Web replicados, baseado em diferentes políticas de seleção de servidores. Além disso, as políticas implementadas foram avaliadas empiricamente, através de experimentos envolvendo um serviço replicado em quatro servidores distribuídos em três continentes. Este serviço foi invocado utilizando as diferentes políticas de seleção de réplicas incorporadas ao *framework*, a partir de dois clientes com diferentes características de conexão. De maneira geral, os resultados obtidos mostram que, além da capacidade individual de cada servidor, a escolha da melhor réplica do serviço é afetada principalmente pelas diferenças de configuração de rede entre os clientes, bem como pela distribuição da carga de trabalho de cada cliente ao longo do dia.

Espera-se que o resultado desse trabalho possa auxiliar na evolução dos atuais *frameworks* que dão suporte aos serviços Web, e a novos que venham a ser implementados, especialmente no que se refere à busca de soluções mais eficientes para o acesso a serviços Web com réplicas geograficamente distribuídas.

1.3 Organização do trabalho

Este trabalho está organizado em seis capítulos, incluindo esta introdução, descritos a seguir:

No capítulo 2, **Tecnologia de Serviços Web**, discorremos sobre a tecnologia dos serviços Web e as demais tecnologias relacionadas, dando um maior detalhamento da linguagem de descrição de serviço WSDL.

No capítulo 3, **Estratégias de Replicação na Web**, revisamos as principais técnicas de replicação utilizadas na Web, com ênfase nos trabalhos que abordam estratégias de acesso a recursos replicados, pelo lado do cliente, fazendo uso de políticas de seleção de servidores. Apresentamos ainda uma análise do suporte oferecido pelos principais *frameworks* de suporte a serviços Web existentes, no contexto específico do acesso a serviços replicados.

No capítulo 4, **RWS – Um *Framework* para Invocação de Serviços Web Replicados**, apresentamos o *framework* RWS, destacando a implementação do processo de tratamento das réplicas e das diferentes políticas de seleção de servidores que o *framework* incorpora.

No capítulo 5, **Avaliação Experimental**, descrevemos a metodologia empregada para condução dos experimentos de avaliação, e analisamos os resultados obtidos do ponto de vista quantitativo e qualitativo.

Por fim, no capítulo 6, **Conclusão**, apresentamos as principais contribuições e resultados deste trabalho, e indicamos possíveis linhas para trabalhos futuros.

Capítulo 2

Tecnologia de Serviços Web

Este capítulo apresenta a tecnologia de Serviços Web, descrevendo suas principais características e tecnologias associadas.

2.1 Histórico

Nos últimos anos, várias tecnologias foram propostas para dar suporte à construção de aplicações baseadas no paradigma de componentes. Entre elas, as duas mais conhecidas são COM (*Component Object Model*) (COM, 1992), introduzida pela Microsoft, e CORBA (*Commom Object Request Broker Architecture*) (CORBA, 1991), introduzida por um consórcio de empresas denominado OMG (*Object Management Group*) (OMG, 1989).

COM e CORBA são tecnologias que encapsulam código binário para suportar chamadas entre componentes independentemente do modo como eles foram implementados. As duas tecnologias têm em comum o fato de não serem facilmente interoperáveis. De acordo com Tabor (2001), o maior problema das duas arquiteturas é que elas são específicas de plataforma. A alternativa para a convivência entre as duas, através da utilização de componentes "pontes", acaba não sendo eficiente na prática pela dificuldade de mapeamento do conjunto de funcionalidades de uma tecnologia para a outra.

Com a larga utilização das redes de computadores, algumas dessas tecnologias foram estendidas para permitir a comunicação entre componentes fisicamente distribuídos. Por exemplo, a OMG padronizou o IIOP (*Internet Inter-ORB Protocol*) (IIOP, 1996) para a tecnologia CORBA, a Microsoft criou o DCOM (*Distributed COM*) (DCOM, 1996), e a SUN criou o RMI (*Remote Method Invocation*) (RMI, 1997), específico para a linguagem Java.

Essas novas tecnologias foram propostas visando, principalmente, aplicações que residem numa mesma rede local. Porém, com o crescimento da Internet, as aplicações passaram a ser geograficamente distribuídas, executadas em diferentes sistemas operacionais, e escritas usando diferentes linguagens de programação. Nesse novo

contexto, vários desafios para o desenvolvimento de aplicações distribuídas utilizando essas tecnologias tiveram que ser considerados:

- a implementação dos componentes, utilizando estes modelos, tornou-se complexa;
- seus protocolos de comunicação são simétricos, ou seja, requerem um mesmo modelo de componentes em ambos os lados da comunicação, o que pode representar uma forte restrição para a comunicação entre aplicações de diferentes organizações;
- alguns desses protocolos são específicos de plataforma;
- nenhuma dessas tecnologias ganhou uma ampla aceitação da indústria.

A tendência cada vez maior para realização de negócios pela Internet tornou evidente a necessidade da integração de aplicações distribuídas através de tecnologias diferentes das opções até então existentes. A solução para isso seria uma nova tecnologia que permitisse integrar diferentes aplicações de forma fácil e simples. Essa tecnologia alternativa deveria:

- ser fácil de implementar e manter;
- estar baseada em padrões abertos amplamente conhecidos;
- garantir a entrega/troca de mensagens sem soluções complexas de software;
- ser independente de sistema operacional ou da linguagem de programação utilizada.

Em 1998, com as discussões em torno da tecnologia XML, um grande passo foi dado na direção do atendimento dessas necessidades de integração. Com o surgimento de padrões baseados em XML para a troca de dados e comunicação entre aplicações, tornou-se possível o desenvolvimento de serviços facilmente integráveis e independentes de plataforma, dada o alto grau de interoperabilidade obtido através do uso desses padrões. Isso serviu de base para a adoção dos chamados *XML Web Services* ou simplesmente Serviços Web.

2.2 Conceito

Encontramos algumas definições dadas aos serviços Web que os caracterizam ora como uma interface, e ora como parte da lógica de negócios das aplicações. Em Snell *et al.* (2001), encontramos a seguinte definição:

> *"Um serviço Web é uma interface posicionada entre o código da aplicação e o usuário desse código. Atua como uma camada de abstração, separando os detalhes específicos de plataforma e linguagem de programação de como a aplicação é invocada. Os serviços Web não são novos, mas representam a evolução de princípios que guiaram a Internet por anos*".

Em Kreger (2001), temos a seguinte definição:

> *"Um serviço Web é uma interface que descreve uma coleção de operações que são acessíveis pela rede, utilizando mensagens padronizadas. É descrita utilizando uma notação formal em XML, chamada de descrição de serviço"*.

Em Chappel e Jewell (2002), a seguinte definição dada é:

> "*Um serviço Web é uma parte da lógica de negócios que está localizada em algum lugar da Internet e que é acessível através de protocolos padronizados da Internet, tais como HTTP ou SMTP*".

Um conceito mais abrangente é dado por Austin *et al.* (2002):

> "*Um Serviço Web é uma aplicação identificada por um URI (Universal Resource Identifier), cujas interfaces e implementações são capazes de serem definidas, descritas e localizadas, utilizando-se a linguagem XML e suas ferramentas. Um Serviço Web deve ser capaz de interagir com outras aplicações, através da troca de mensagens baseadas em XML, utilizando os protocolos de comunicação existentes na Internet*".

De maneira geral, os serviços Web representam uma camada de abstração que separa plataforma e linguagem de programação dos detalhes da aplicação (Snell *et al.*, 2001). O nível de abstração obtido faz com que a plataforma em que se executa um serviço Web seja irrelevante.

Por serem baseados em padrões da Internet, a interoperabilidade é um dos principais benefícios obtidos com a tecnologia de serviços Web, muito embora, para obtê-la, tenha-se o custo da latência de comunicação entre as aplicações, conforme mostram os estudos realizados por Litoui (2002) e Elfwing *et al.* (2002). Por outro lado, as soluções

baseadas nas tecnologias tradicionais de componentes (por exemplo, CORBA, DCOM e RMI) apresentam grandes dificuldades de integração. Numa relação de custo/benefício, a latência imposta pelas camadas adicionais para utilização de um serviço Web deve ser considerada à luz do fato de que ela torna a comunicação entre plataformas distintas e heterogêneas uma realidade.

2.3 Caracterização dos serviços Web

Trabalhos como Conti *et al.* (2002b) e Vingralek *et al.* (1999) têm utilizado a terminologia "Serviços Web" para caracterizar recursos da Web em geral (páginas HTML, imagens, *scripts*, etc). A diferença entre esses recursos e o novo conceito é a padronização que foi dada aos serviços Web. Entre os padrões utilizados, temos o protocolo SOAP (Box *et al.*, 2000), que, em contraste aos padrões binários até então existentes, oferece neutralidade de linguagem de programação e de formato de representação dos dados.

As definições de serviço Web apresentadas no item anterior estão associadas, de certa forma, às características dessa tecnologia descrita em Snell *et al.* (2001):

- Baseados em XML – XML, usado como uma camada de representação dos dados, facilita a interoperabilidade, eliminando a heterogeneidade de plataforma, sistema operacional e rede;
- Fracamente acoplados – O processo de ligação entre os serviços Web, incluindo a sua localização e utilização, é mais simples, sem as complexidades impostas pelas tecnologias anteriores, que exigem um forte acoplamento entre os componentes;
- Capazes de interagir de forma síncrona ou assíncrona;
- Suportam Chamadas de Procedimento Remoto (RPC) – serviços Web permitem que as aplicações clientes realizem chamadas a métodos remotos utilizando protocolos baseados em XML. A linguagem de descrição de serviço WSDL (Christensen *et al.*, 2001) expõe os métodos e os parâmetros que um serviço suporta.
- Suportam troca de documentos – a comunicação através da troca de documentos, ao invés do envio e recebimento e parâmetros formais, facilitam a integração entre negócios.

2.4 Estrutura da tecnologia de serviços Web

Numa visão dada por Snell *et al.* (2001), os serviços Web estão estruturados numa pilha, compondo uma arquitetura implementada em cinco camadas, cada uma baseada em um diferente tipo de tecnologia, conforme mostra a Figura 2.1. Essas cinco camadas, juntamente com suas respectivas tecnologias, são descritas a seguir:

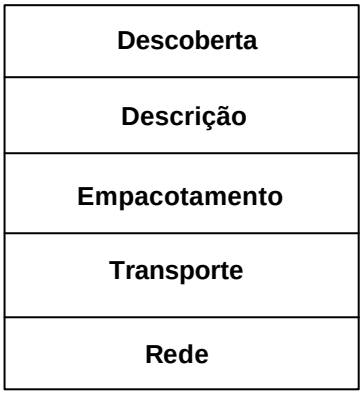

Figura 2.1: Pilha da tecnologia dos serviços Web (Snell *et al.*, 2001).

- **Descoberta -** Esta camada provê o mecanismo para a descoberta dos serviços. É o meio pelo qual os consumidores dos serviços podem obter dos provedores a descrição dos serviços que eles disponibilizam. A tecnologia UDDI (Bryan *et al.*, 2002) é utilizada nessa camada.
- **Descrição -** A camada de descrição especifica o serviço em termos do que ele pode suportar, tais como os protocolos de transporte e de empacotamento. Representa o meio pelo qual o consumidor de um serviço obtém informações sobre como contatá-lo e utilizá-lo. A tecnologia WSDL é o padrão para a descrição de serviço mais utilizado. Outras abordagens menos populares incluem o RDF (*Resource Description Framework*) (RDF, 2000) e a DAML (*DARPA Agent Markup Language*) (DAML, 2000). Ambos fornecem uma descrição semanticamente mais rica para os serviços, porém com muito mais complexidade que a WSDL.
- **Empacotamento -** Para permitir a interação entre as aplicações, as mensagens devem ser empacotadas em um formato que ambos os lados possam interpretar. O processo de empacotamento é conhecido também pelo termo serialização. O empacotamento das mensagens trocadas pelos serviços Web é baseado na tecnologia SOAP, que utiliza XML como a linguagem para codificar a estrutura das mensagens, incluindo os tipos e os valores dos dados utilizados como parâmetros.

- **Transporte** - A camada de transporte inclui várias tecnologias que permitem a comunicação entre aplicações clientes através da Internet. As mais conhecidas são os protocolos TCP, HTTP e SMTP. Na prática o protocolo HTTP é de longe o mais utilizado, muito embora não forneça suporte a comunicação assíncrona.
- **Rede** - A camada de rede fornece os serviços básicos de comunicação para a camada de transporte, tais como endereçamento e capacidade de roteamento.

2.5 Arquitetura Orientada a Serviço

Uma arquitetura orientada a serviço (*Service Oriented Architecture - SOA*) (Papazoglou e Georgakopoulos, 2003; Kreger 2001; Snell *et al*., 2001) representa a visão de um modelo de computação integrada, baseado nas principais tecnologias da pilha dos serviços Web. Conceitualmente, o modelo SOA compreende a participação de três entidades, desempenhando papéis de provedor, registro e consumidor, e executando interações que envolvem a publicação, descoberta e invocação dos serviços.

Nesse modelo, os serviços estão estruturados sob dois aspectos:

- Serviço – A implementação propriamente dita do serviço Web. Um serviço pode ser uma simples classe Java, uma aplicação COBOL executando num *mainframe*, etc. Os requisitos são que ele seja acessível através de um servidor Web e utilize as tecnologias padronizadas;
- Descrição do serviço – A interface do serviço Web. A descrição é feita pela linguagem WSDL, estruturada em XML, e inclui: tipos de dados, operações, protocolos e informações de localização.

Os papéis das entidades participantes são:

- Provedor – Do ponto de vista de negócios, é considerado o proprietário do serviço. De uma perspectiva de arquitetura, é a plataforma que hospeda o serviço.
- Registro - Gerencia um repositório com as informações necessárias para se fazer uso do serviço Web, oferecendo uma classificação dos serviços por categorias e um serviço de busca inteligente.
- Consumidor – Do ponto de vista de negócios, é a parte que busca certas operações a serem executadas. Do ponto de vista arquitetural, é aquele que

descobre, invoca ou inicia a interação com serviços disponibilizados por um ou mais provedores.

A Figura 2.2 representa a arquitetura de interação entre o provedor, o registro e o consumidor (Kreger, 2001).

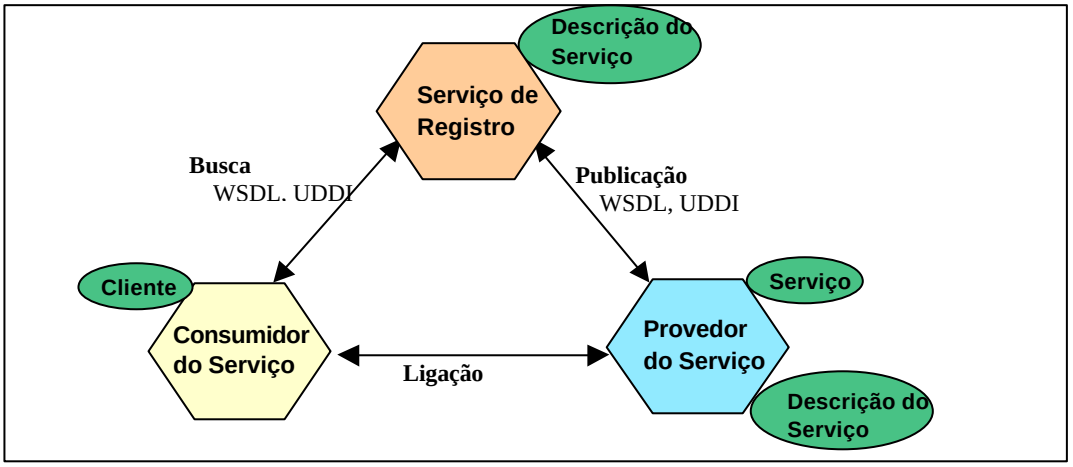

Figura 2.2: Papéis das três entidades envolvidas numa arquitetura orientada a serviço (Kreger, 2001).

As interações fundamentais entre os três participantes são:

- Publicação – O provedor publica metadados sobre um serviço. Os provedores são normalmente órgãos de padronização, fornecedores de software e desenvolvedores.
- Localização do Serviço – O consumidor é uma aplicação que faz uma consulta a um registro para obter a descrição dos serviços. O critério utilizado para localizar um serviço inclui a sua classificação e o protocolo suportado.
- Ligação – Esta interação envolve o consumidor e o provedor. A ligação é implementada a partir da descrição do serviço com a formatação de uma chamada de operação a ser enviada ao provedor. A descrição do serviço (WSDL) informa quais protocolos são suportados (HTTP, MIME, SMTP etc).

2.6 Fatores para adoção

Problemas clássicos, envolvendo a integração de aplicações comerciais, requerem soluções de fácil implementação. Dentro de uma mesma empresa, por exemplo, existem pequenas ilhas de tecnologias. Como exemplo, uma aplicação de gerenciamento de relacionamento com clientes (CRM – *Customer Relationship Management*) pode não ter a

capacidade de interagir com nenhuma outra fora do contexto da mesma. Os problemas de integração desse tipo de aplicação existem e aumentam cada vez mais. Os serviços Web oferecem uma opção atrativa para resolução desses problemas de integração de sistemas.

Outros modelos de computação distribuída tiveram a oportunidade de obter uma aceitação ampla pela indústria, mas não a conseguiram. DCOM e CORBA, em particular, oferecem grandes vantagens sob o ponto de vista técnico para resolver questões reais, porém não obtiveram a adoção que os seus proponentes esperavam. Embora DCOM tenha obtido uma maior aceitação, está praticamente limitada à plataforma Microsoft. CORBA, que é controlado pela OMG, sofre do problema da falta de uma maior disponibilidade de software. Até hoje existem poucos vendedores de implementações CORBA no mercado.

A adoção de CORBA ou DCOM requer que as partes envolvidas instalem *middleware* que são específicos dessas tecnologias. Muitas vezes, isso não é possível por questões de plataformas ou pelo simples motivo que empresa já usuária de uma tecnologia não queira fazer uso de outra.

Os serviços Web também exigem um *middleware*, porém as principais arquiteturas de desenvolvimento e ambientes de execução já incorporam essa camada adicional, tornando-os interoperáveis mesmo considerando que o consumidor utilize uma arquitetura e o provedor utilize uma outra diferente. Um exemplo é a possibilidade de comunicação bidirecional que pode ser obtida entre uma aplicação gerenciada pelo servidor de aplicações BizTalk (BIZTALK, 2002), da Microsoft, e uma aplicação gerenciada pelo servidor de aplicação J2EE (J2EE, 1999), da Sun Microsystems. O requisito necessário é que as aplicações utilizem mensagens SOAP e transmissões em HTTP, por exemplo. O envolvimento das empresas Sun Microsystems, com a arquitetura de aplicações J2EE, e da Microsoft, com a arquitetura.NET, em torno dos serviços Web, é importante porque ambas podem ter suas especificidades dentro de seus campos de atuação, todavia oferecendo interoperabilidade entre seus módulos, por meio de uma tecnologia única.

2.7 Tecnologias associadas

Nas seções seguintes, descrevemos as principais tecnologias de suporte aos serviços Web, com destaque para a linguagem WSDL, a qual utilizamos no contexto deste trabalho.

2.7.1 SOAP *(Simple Object Access Protocol)*

SOAP é um protocolo de comunicação para troca de informações entre aplicações, não definindo um modelo de programação, nem uma interface para programadores de aplicações (API). SOAP define um simples mecanismo de troca de mensagens, por meio de um modelo de empacotamento e codificação de dados baseado em esquemas XML (*XML Schemas*) (Fallside, 2001).

SOAP também não define o protocolo de transporte a ser utilizado. Limita-se a definir o formato das mensagens, podendo usar as tecnologias de transporte já em uso na Internet, como os protocolos HTTP, SMTP, entre outros.

SOAP é um padrão do W3C (*World Wide Web Consortium*) (W3C, 1994). Isto significa que sua especificação não é de controle de nenhuma empresa específica, mas de uso comum, como os demais padrões adotados na Internet (HTML, XML, HTTP e SMTP). SOAP também não substitui inteiramente protocolos similares existentes, tais como DCOM, CORBA/IIOP ou RMI, nem tem um modelo de objeto próprio, mas permite a ligação entre aplicações que utilizem diferentes modelos de objetos e linguagens de programação.

Chappel e Jewell (2002) caracterizam o protocolo SOAP explicando a composição da sigla que o identifica:

- Simples (S*imple)* – A abordagem de expressar dados e transportá-los pela Internet utilizando HTTP é uma implementação simples. Em SOAP, tudo é expresso em termos de HTTP, SMTP, MIME etc. Até mesmo a idéia do envio de anexos numa mensagem SOAP é simples, pois se baseia em MIME, protocolo existente e bastante utilizado na Internet. Tudo isto permite que plataformas e linguagens conversem entre si de forma independente.
- Objeto (*Object*) – Por meio de SOAP, é perfeitamente possível descrever uma chamada remota de procedimento (RPC) ou invocar um método de um objeto remoto da forma que é feita utilizando outras tecnologias que implementam RPC.
- Acesso (*Access*) – Um fator importante do protocolo é a acessibilidade. O acesso é feito utilizando-se principalmente HTTP e, em menor escala, SMTP, uma vez que esses são os protocolos em nível de rede universalmente utilizados. Muitos *firewalls* estão configurados para permitir o tráfego desses protocolos, possibilitando às mensagens irem além das barreiras de segurança.

- Protocolo (*Protocol*) – O conjunto dessas características fazem do SOAP um protocolo ideal para a troca de informações num ambiente distribuído.

2.7.2 WSDL (*Web Services Description Language)*

Um documento WSDL se assemelha, em termos de conteúdo, a um documento IDL CORBA ou IDL Microsoft. Ele é utilizado para definir as operações, interfaces, tipos de dados, localização e protocolo de ligação de um serviço Web. Por outro lado, por ser baseada em XML, a WSDL oferece graus de extensibilidade não encontrado nas especificações IDL tradicionais. Esta extensibilidade permite:

- descrever os serviços e suas operações sem levar em conta o formato da mensagem ou protocolo de rede utilizado;
- considerar as mensagens como descrições abstratas dos dados a serem trocados;
- tratar "*port types*" como coleções abstratas das operações de um serviço. Cada "*port type*" pode ser mapeado para diferentes protocolos de transporte e formatos de representação de dados.

Um documento WSDL não contém código, mas metadados sobre as operações implementadas no serviço Web. A estrutura básica de um documento WSDL é a apresentada na Figura 2.3. Os itens assinalados com (*) podem aparecer uma ou mais vezes no documento.

A geração de um arquivo WSDL pode ser obtida a partir do código de implementação do serviço. De forma similar, o código de implementação para consumir o serviço pode ser gerado da interpretação do conteúdo de um documento WSDL. Existem ferramentas que realizam esse trabalho de forma automática, seja a partir do código fonte, seja de um serviço publicado. Por exemplo, no *framework* AXIS (Apache, 2002) esse trabalho é desempenhado pela classe *WSDL2Java.* Uma vez obtido o arquivo WSDL contendo a descrição de um serviço, a classe mencionada gera toda as interfaces e classes necessárias para que uma aplicação Java possa consumir o Serviço Web.

A seguir, apresentamos os principais elementos que constituem um documento WSDL:

```
<definitions>
  <types>
    <schema></schema>(*)
  </types>
  <message>(*)
     <part></part>(*)
  </message>
  <PortType>(*)
     <operation>(*)
        <input></input>
        <output></output>
        <fault></fault>(*)
     </operation>
  </PortType>
  <binding>(*)
     <operation>(*)
        <input></input>
        <output></output>
     </operation>
  <binding>
  <service>(*)
     <port><port>(*)
  </service>
</definitions>
```

Figura 2.3: Estrutura básica de um documento WSDL (Christensen *et al.*, 2002).

a) O elemento <definitions>

Atua como um container para a descrição do serviço. Nesse elemento, encontram-se definidos todos os espaços de nomes (*namespaces*) que irão identificar os demais elementos contidos no corpo do documento WSDL.

b) O elemento <types>

Descreve os tipos de dados utilizados nas mensagens. O padrão de definição de tipos é o XSD (XML *Schema Definitions*) (Biron e Malhotra, 2001). Tipos de dados especiais na forma de classes são definidos como tipos complexos (*complex types*) como mostra a Figura 2.4.

c) O elemento <message>

Este elemento define os dados utilizados nas mensagens trocadas com um serviço Web. Além da identificação, a mensagem é composta de elementos <part> que identificam os parâmetros das operações com seus tipos de dados, tanto aqueles de entrada como os de retorno, se houver. A Figura 2.5 ilustra a composição de um elemento <message>.

O atributo *name* do elemento <message> identifica a mensagem, e o mesmo atributo, no elemento <part>, dentifica o parâmetro. O atributo *type* identifica o tipo de dado. O tipo pode também ser um tipo complexo definido anteriormente, no elemento <types>, a exemplo do tipo "Book" na Figura 2.4. Uma mensagem pode ser tanto uma chamada a uma operação, uma resposta ou uma mensagem de falha.

```
<wsdl:types>
  <schema targetNamespace="http://bookstore"
          xmlns="http://www.w3.org/2001/XMLSchema">
    <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
    <complexType name="Book">
      <sequence>
        <element name="isbn" nillable="true" type="xsd:string" />
        <element name="year" type="xsd:int" />
        <element name="title" nillable="true" type="xsd:string" />
      </sequence>
    </complexType>
    <element name="Book" nillable="true" type="tns1:Book" />
  </schema>
</wsdl:types>
```

Figura 2.4: Exemplo da estrutura de um elemento <types> num documento WSDL.

```
      <wsdl:message name="FindBookRequest">
        <wsdl:part name="in0" type="xsd:string" />
      </wsdl:message>

      <wsdl:message name="FindBookResponse">
        <wsdl:part name="FindBookReturn" type="xsd:string" />
      </wsdl:message>
```

Figura 2.5: Exemplo da estrutura do elemento <message> num documento WSDL.

d) O elemento <portType>

Este elemento declara o conjunto de operações suportado por um serviço Web. Cada operação é definida dentro de um subelemento <operation>, que fornece os identificadores dos métodos suportados pelo serviço, e as ligações que são definidas no elemento <message>. Cada operação é mapeada de acordo com os seguintes padrões de comportamento:

- Requisição-Resposta (*request-response*) – Esta operação segue um modelo no qual um consumidor envia uma requisição ao serviço e este retorna uma resposta em modo síncrono. Este modelo é similar a uma chamada de procedimento remoto (RPC). As mensagens são identificadas pelos subelementos <input> e <output>. Pode também incluir um subelemento

<fault>, que indicará algum erro de processamento, caso ocorra. A Figura 2.6 ilustra uma operação com este comportamento;

- Solicitação-Resposta (*solicit-response*) - Neste modelo, o serviço Web solicita uma resposta de um consumidor. O elemento <output> deve aparecer antes do elemento <input>, indicando que na operação o serviço primeiro envia uma mensagem e depois recebe uma resposta. Inclui também, um subelemento <fault>.
- Invocação de via única (*one-way)* – Neste modelo, o consumidor envia apenas uma mensagem para o serviço, sem o recebimento de resposta associada. Define apenas um subelemento <input> sem mensagem de <output>.
- Notificação (*notification*) – A operação se resume, apenas, a uma notificação do serviço enviada ao consumidor. O serviço envia mensagens assíncronas ao consumidor, ou seja, as mensagens não são respostas a nenhuma requisição.

```
<wsdl:portType name="BookStore">
  <wsdl:operation name="FindBook" parameterOrder="in0">
   <wsdl:input message="impl:FindBookRequest"
               name="FindBookRequest" />
   <wsdl:output message="impl:FindBookResponse"
               name="FindBookResponse" />
  </wsdl:operation>
</wsdl: portType>
```

Figura 2.6: Exemplo da estrutura do elemento <PortType> num documento WSDL.

e) O elemento <binding>

Este elemento declara o protocolo e o formato de dados para um elemento <portType> que pode ser HTTP, SOAP ou MIME. O elemento <binding> descreve a definição abstrata das operações, mapeando-as para o protocolo que o serviço utiliza. Por exemplo, o SOAP tem seu próprio cabeçalho e corpo, podendo ambos estar inseridos dentro de pacote http, como mostra a Figura 2.7:

f) O elemento <service>

Este elemento é utilizado para descrever a localização do serviço. O elemento <service> se constitui de um ou mais elementos <port>, os quais podem representar um protocolo ou URL de acesso ao serviço (*endpoint*). O elemento <port> tem um atributo *name* que identifica, unicamente, um dado protocolo de ligação ou endereço de localização do serviço, e um atributo *binding* que faz referência ao elemento <binding> contido no

WSDL. O elemento <soap:address> contém a URL completa para a chamada de uma das operações do serviço.

O presente trabalho explora a possibilidade de podermos definir múltiplas URLs para um mesmo serviço, nesse elemento, como uma das bases para o tratamento da localização das réplicas de um serviço Web replicado. Esse tratamento é descrito em mais detalhes no capítulo 4.

```
<wsdl:binding  name="BookStoreServiceSoapBinding"
               type="impl:BookStore">
  <wsdlsoap:binding style="rpc"
                   transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="FindBook">
      <wsdlsoap:operation soapAction="" />
        <wsdl:input name="FindBookRequest">
          <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://servidor/axis/services/BookStoreService"
            use="encoded" />
        </wsdl:input>
        <wsdl:output name="FindBookResponse">
          <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://servidor/axis/services/BookStoreService"
            use="encoded" />
      </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

Figura 2.7: Exemplo da estrutura do elemento <binding> num documento WSDL.

2.7.3 UDDI (*Universal Description, Discovery and Integration)*

O UDDI fornece um mecanismo padrão para publicação e descoberta de informações sobre os serviços Web. O padrão UDDI é uma iniciativa da indústria, e tem como objetivo criar uma plataforma independente e aberta para descrever, descobrir e integrar serviços Web.

A descoberta dos serviços, de forma manual e em nível global, poderá não ser adequada, considerando que os serviços Web estão se tornando a base do comércio eletrônico em várias formas. A proposta do UDDI tem como base facilitar a descoberta de potenciais parceiros que disponibilizem serviços na Internet.

Um serviço UDDI pode registrar informações sobre serviços dentro da seguinte estrutura:

- Páginas Brancas (*white pages*) – inclui informações básicas de contato e identificação sobre uma empresa. Esta informação permite descobrir um serviço Web pela identificação da empresa;
- Páginas Amarelas (*yellow pages*) – Descreve o serviço utilizando diferentes classificações. Permite descobrir um serviço baseado na sua categorização.
- Páginas Verdes (*green pages*) – Descreve informações técnicas sobre o serviço e onde o mesmo pode ser localizado.

O UDDI *Business Registry* (OASIS, 2002) é um serviço único distribuído entre alguns pontos da Internet que replicam entre si os conteúdos dos serviços registrados. O serviço é oferecido com uma interface para acesso a partir de navegadores Web, uma API para acesso a partir de aplicações clientes ou na forma de um serviço Web.

Atualmente, o UDDI *Business Registry* é disponibilizado pelas empresas IBM (IBM, 2002), Microsoft (Microsoft, 2002b), SAP (SAP, 2002) e NTT *Communications Corporation* (NTT, 2002). As quatro empresas disponibilizam o serviço na forma de um serviço Web replicado em quatro servidores distribuídos em três continentes: América (IBM e Microsoft), Europa (SAP) e Ásia (NTT).

2.8 Outras tecnologias

Além das tecnologias mencionadas nas seções anteriores, outras especificações relacionadas aos serviços Web têm sido propostas na direção da composição, coordenação, transação e segurança dos serviços (Curbera *et al.*, 2003).

A *Business Process Execution Language for Web Services* (BPEL) (Andrews *et al.*, 2003) define uma linguagem para criação de composição de serviços na forma de processos de negócios. A BPEL está sendo padronizada pela *Organization for the Advancement of Structured Information Standard* (OASIS).

Para coordenação e transação, surgem a *WS-Coordination* (Cabrera *et al.*, 2003) e a *WS-Transaction* (Cabrera *et al.*, 2002), que objetivam o controle da execução de serviços num contexto de transações distribuídas dentro e entre organizações.

No que se refere à segurança, temos a *WS-Security* (Atkinson *et al.*, 2002), que propõem extensões à estrutura de mensagens SOAP para prover as características de integridade, confidencialidade e autenticação no acesso aos serviços.

2.9 Sumário

Neste capítulo, discorremos sobre algumas tecnologias de componentes que inicialmente deram suporte à construção de aplicações distribuídas, para então situar a nova tecnologia dos serviços Web nesse contexto. Apresentamos a tecnologia de serviços Web, caracterizando-a na forma padronizada pelo W3C, e descrevemos sua arquitetura e tecnologias associadas.

No próximo capítulo, revisamos as principais técnicas de replicação utilizadas na Web destacando as abordagens que tratam o acesso a recursos replicados, incluindo os serviços Web, pelo lado do cliente.

Capítulo 3

Estratégias de Replicação na Web

Este capítulo dá uma visão geral da replicação de recursos na Web, com ênfase nas estratégias de acesso a recursos replicados pelo lado do cliente.

A Internet tem crescido vertiginosamente ao longo dos anos. O aumento acelerado do número de usuários que utilizam a rede mostra a dimensão desse crescimento. De acordo com pesquisa realizada pelo *Computer Industry Almanac Inc*. (Almanac, 2002), estima-se que existiam 533 milhões de usuários fazendo uso da rede no ano de 2001. Em 2002, foram 665 milhões. A previsão é que esse número deva evoluir para 943 milhões no final de 2004 e 1,4 bilhões em 2007.

Esse crescimento deve-se, principalmente, à popularidade da Web, que cada vez mais atrai novos usuários. Uma conseqüência direta desse crescimento é o tráfego acentuado na rede, e o aumento na carga de trabalho dos servidores. Esses dois problemas são decorrentes, principalmente, da crescente demanda e também da transferência repetitiva de recursos de um ponto para outro na rede. A mesma informação trafega na Internet diversas vezes, acessada por múltiplos clientes. Para documentos e serviços solicitados com muita freqüência, é considerável o aumento do tempo de resposta percebido no lado cliente.

Como forma de tratar os problemas acima mencionados, vários trabalhos têm sido realizados no sentido de propor mecanismos que tornem o acesso aos recursos disponibilizados na Web mais eficiente. Boa parte dos mecanismos propostos são baseados em técnicas de replicação.

O restante deste capítulo trata mais detalhadamente a questão da replicação de recursos na Web, descrevendo e comparando algumas das principais técnicas de replicação utilizadas.

3.1 Técnicas de replicação

Em meados da década de 90, alguns anos após o surgimento da Web, Berners-Lee *et al.* (1996) já destacavam o crescimento exponencial da Web. Naquela época, eles também já ressaltavam a questão da replicação como forma de obter maior escalabilidade e melhoria no tempo de acesso aos recursos para os usuários finais.

A replicação deve ocorrer de forma que seja transparente aos usuários. Isto requer que mecanismos sejam construídos para redirecionar requisições de clientes para os servidores com réplicas dos recursos. Um dos objetivos é encontrar o servidor que ofereça melhor tempo de resposta para cada cliente. Manter a consistência dos conteúdos entre os servidores é outro fator importante.

Ao longo dos anos, várias técnicas envolvendo replicação foram desenvolvidas, com enfoque na infra-estrutura do lado do servidor ou na forma de um serviço *proxy* localizado próximo ao cliente ou mesmo incorporado a ele.

Os serviços *proxy* desempenham o papel de manter cópias dos recursos mais solicitados, que são armazenados em locais próximo ao cliente. O armazenamento, ou *caching,* pode ser feito em disco e/ou na memória principal do servidor (Markatos e Crete, 1996). A utilização desta técnica também pode ser aplicada na aplicação cliente ou nos próprios servidores Web que hospedam os recursos (Dingle e Parl, 1996).

Pelo lado da infra-estrutura do servidor, temos as técnicas que estão baseadas no serviço de resolução de nomes DNS *(Domain Name Service)* (DNS, 1987). A técnica DNS *round-robin*[1] (Schemers, 1995) teve sua primeira implementação em servidores Web homogêneos conduzida pelo *National Center for Supercomputing Applications* (NCSA) (Kwan *et al.,* 1995). Esse esquema é empregado, por exemplo, pela Netscape. O nome do servidor *home.netscape.com* na verdade é uma referência (*alias)* para outros nomes de servidores. A Netscape incorpora no seu navegador um mecanismo baseado neste esquema de resolução de nomes. Quando um cliente tenta acessar a página *home.netscape.com,* é feito um mapeamento para um nome de servidor correspondendo ao nome real na forma *homeN.netscape.com,* onde N é um número entre 1 e 32. A desvantagem desse mecanismo

[1] *round-robin* - a cada solicitação de resolução de nome de um servidor, um dos endereços IP, de um conjunto de servidores espelhados, é retornado num algoritmo semelhante ao de processamento de uma fila circular.

é que ele só funciona no navegador da Netscape. Além disso, por estar baseada apenas em DNS, esta técnica não leva em conta a capacidade e a disponibilidade de cada servidor.

Uma outra alternativa para as técnicas baseadas em DNS é a utilização dos produtos *CISCO Distributed Director e CISCO Local Director* (CISCO, 2003). No primeiro, a seleção do servidor leva em conta a localização geográfica do cliente. Esta solução implementa algumas métricas para a resolução do nome, tais como: a distância de roteamento baseada em número de *hops*, seleção randômica, etc. No segundo, a primeira abordagem foi ampliada, de forma que a seleção do servidor também considerasse a proximidade do cliente e a carga nos servidores.

Outra abordagem de replicação é o agrupamento dos servidores hospedeiros dos recursos replicados numa infra-estrutura única, que agrupa os servidores num conjunto denominado *cluster*. Várias pesquisas (Dias *et al.*, 1996; Damani *et al*., 1997; Hunt *et al.*, 1998) propõem o mapeamento dos endereços dos servidores para um único endereço IP virtual. Esse endereço é gerenciado por um servidor centralizado, que é responsável pela seleção do servidor que de fato atenderá a requisição dos clientes.

Outro mecanismo de replicação utilizado, que segue uma abordagem oposta ao *cluster*, é a descentralização de servidores, distribuindo-os geograficamente na Internet, a fim de que a redução da carga de acesso não fique apenas em nível de servidor, mas também em nível de rede. Como exemplo de utilização desta técnica podemos citar novamente os servidores da Netscape, que adotam essa distribuição associada à técnica baseada em DNS, conforme descrito anteriormente.

A seguir, destacamos os principais trabalhos que propõem estratégias de acesso a recursos geograficamente replicados, pelo lado do cliente, que são o foco desta dissertação.

3.2 Estratégias de acesso pelo lado do cliente

As técnicas de replicação, consideradas na seção anterior, surgiram como alternativas para resolver o problema de que servidores individuais, quando bastantes acessados, não suportam uma alta carga de requisições, ocasionando em elevados tempos de respostas para os clientes. A alternativa de *cluster* melhora a questão na carga do servidor, todavia não resolve a demora no atendimento ocasionada pela carga nos serviços de rede. Para servidores geograficamente distribuídos, as técnicas baseadas em DNS podem propiciar um melhor balanceamento de carga entre os servidores, porém sem

considerar a carga individual de cada servidor nem a carga em nível de rede. Outro aspecto é que um mesmo servidor pode ser acessado seguidamente a partir de um mesmo cliente, dependendo da freqüência com que o cliente envia novas requisições ao serviço de nomes.

No contexto de servidores geograficamente distribuídos, alguns trabalhos (Yoshikawa *et al.*, 1997; Rastogi, 1999; Vingralek *et al.*, 1999), descritos a seguir, propõem um tratamento específico no acesso aos recursos replicados, pelo lado do cliente. Esses estudos exploraram formas alternativas de controlar a criação e a localização das réplicas, e de selecionar e enviar requisições aos servidores com melhor desempenho, tomando como base critérios como: a distância do servidor, calculada pelo número de pontos de roteamento até o cliente; a carga no servidor, obtida a partir de informações fornecidas pelo próprio servidor; o tráfego na rede, obtido através do envio de sondas aos servidores; e dados estatísticos, obtidos a partir de medições feitas em requisições anteriores.

A abordagem de tratar a replicação pelo lado cliente tem como vantagem a visão geral da rede que pode ser obtida a partir do cliente.

3.2.1 Clientes inteligentes

Yoshikawa *et al.* (1997) argúem que, em muitos casos, prover acesso transparente aos usuários pelo lado do cliente, ao invés do lado do servidor, aumenta a flexibilidade e o desempenho no acesso aos recursos. Entende-se por transparência de acesso, a capacidade da infra-estrutura de *software* do lado do cliente abstrair da aplicação cliente o mecanismo de localização e acesso ao recurso. Eles implementaram uma arquitetura de acesso denominada clientes inteligentes (*smart clients*), para prover acesso a três serviços comuns na Internet: telnet, ftp e chat. A principal vantagem dessa arquitetura é aumentar a flexibilidade, pois os clientes, tomando conhecimento da carga relativa em cada um dos servidores que disponibilizam recursos, podem conectar-se àqueles que possam oferecer o melhor desempenho. Os processos de seleção e conexão, de forma ideal, devem ocorrer sem a intervenção do usuário final.

A camada de clientes inteligentes foi implementada na forma de uma *applet* Java. Quando um usuário deseja fazer uso dessa camada, explicitamente efetua o *download* da *applet* que se integra ao navegador Web. A partir daí, a *applet* passa a gerenciar o acesso

aos servidores sobre os quais detem as informações de réplicas. Alterações no número e na localização das réplicas são passadas dos servidores diretamente à *applet*.

A Figura 3.1 ilustra, de forma simplificada, a arquitetura dos clientes inteligentes proposta por Yoshikawa *et al.* (1997). A *applet* disponibiliza uma interface *NameResolver* através da qual é feita a seleção do servidor a ser utilizado.

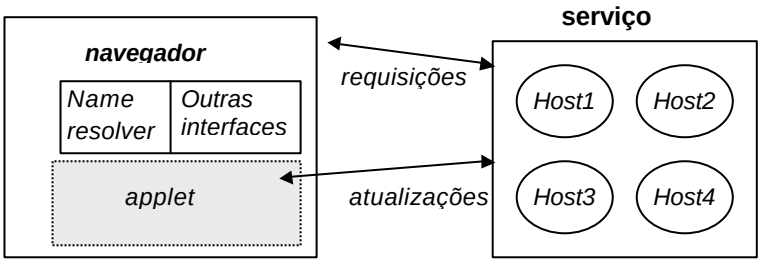

Figura 3.1: Arquitetura básica dos *Smart Clients* proposta por Yoshikawa *et al.* (1997).

Para a seleção do serviço *telnet*, a interface *NameResolver* da *applet* gerencia o número de conexões nos servidores, a partir de informações obtidas destes, através de chamadas periódicas (*polling*) para um programa do tipo CGI (*Common Gateway Interface*), residente em cada servidor. No caso do *ftp*, a seleção se dá de forma randômica. A distribuição do serviço de *chat* utiliza um sistema de arquivos distribuídos, onde cada conexão representa uma entrada em arquivo o qual é sincronizado entre os servidores participantes das salas de *chat*.

3.2.2 *Proxy* inteligente

Rastogi (1999) propõe a implementação de replicação de documentos na Web, de forma transparente aos clientes, incluindo a configuração de réplicas, seleção de servidor e um mecanismo de redirecionamento de requisições, a partir dos clientes. O estudo tomou como premissa atender aos objetivos de reduzir a latência e dar transparência e flexibilidade no acesso pelo lado do cliente.

De forma similar aos clientes inteligentes, Rastogi (1999) propõe que a inteligência para a escolha do servidor resida do lado do cliente. Porém, ao contrário dos clientes inteligentes, o tratamento de acesso é feito num servidor *proxy*. Nesse caso, temos a figura

de um servidor *proxy* inteligente, o qual exigiu uma modificação na camada do protocolo HTTP, particularmente, nos servidores com os quais o *proxy* deveria interagir como parte do processo de seleção. A modificação consistiu na inserção de uma diretiva HTTP, enviada pelo servidor, através da qual o *proxy* retira as informações sobre a existência de réplicas. A escolha da réplica a ser acessada é feita de duas formas. Na primeira, a réplica é escolhida de forma aleatória. Na segunda, quando uma requisição é enviada a uma das réplicas, é medido o intervalo de tempo entre a solicitação e o recebimento do primeiro *byte* da resposta. O tempo obtido é utilizado para atualizar a média dos tempos obtidos de uma dada réplica. Esses tempos são então utilizados para requisições futuras, num processo de escolha baseado em dados estatísticos.

3.2.3 Web++

Vingralek *et al.* (1999) propõem mecanismos de replicação de recursos, na forma de documentos HTML, no lado do servidor, como também estratégias para acesso a esses recursos pelo lado do cliente. O trabalho propõe a arquitetura Web++, que é composta por clientes e servidores Web++. Pelo lado do cliente, o trabalho guarda estreita relação e faz referência à abordagem dos clientes inteligentes. Os clientes são implementados na forma de *applets* assinadas e criptografadas. Os servidores, por sua vez, são implementados através de *servlets*. A arquitetura Web++ está ilustrada na Figura 3.2.

Para cada documento manuseado na arquitetura Web++, os servidores mantêm um diretório de replicação que mapeia cada URL lógica a um conjunto de URLs reais existentes no documento. A consistência é mantida entre os servidores por uma *servlet* integrada ao servidor Web.

Os servidores Web++ criam, destroem e atualizam as informações das réplicas. Os servidores mantêm a consistência propagando suas informações locais entre os demais servidores que participam da arquitetura.

As requisições enviadas pelos clientes Web++ são gerenciadas através de uma *applet* que é integrada ao navegador. Quando um cliente Web++ faz uma primeira requisição para um servidor Web++, o processo de acesso ocorre da seguinte forma:

1. O navegador envia uma chamada HTTP ao servidor onde o recurso está publicado. A requisição sendo endereçada a um servidor da arquitetura fará o *download* automático da *applet* que se integrará ao navegador.
2. A *servlet* intercepta a mensagem e obtém o documento HTML requisitado. Antes do envio para o cliente, ela verifica cada *hyperlink* contido no documento, identificando os que apontam para recursos replicados. Para cada *hyperlink* que aponta para réplicas é inserido um *javascript* que servirá de interface para a *applet* nas futuras requisições aos documentos replicados.
3. A partir deste ponto, as futuras solicitações são interceptadas pela *applet* e a escolha do servidor é feita com base em políticas de seleção aplicadas à lista de réplicas gerenciadas pela *applet*.

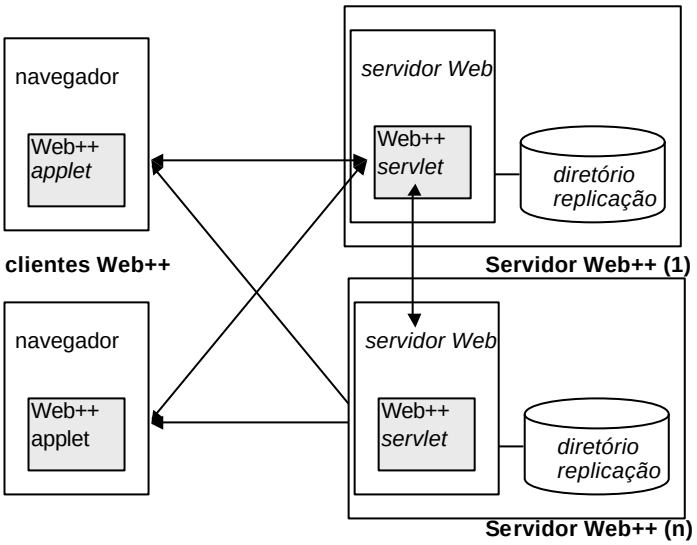

Figura 3.2 – Servidores e clientes na arquitetura Web++ proposta por Vingralek *et al.* (1999).

A partir das informações obtidas das réplicas, a *applet* coleta dados os quais são persistidos no disco local da máquina cliente. A seleção de servidores é feita com base em políticas como: número de pontos de roteamento na rede entre o cliente e o servidor; medições baseadas no envio de pacotes de eco no formato ICMP (*ping round trip time*); e tempo computado no envio e recebimento do primeiro *byte* de um pacote do protocolo HTTP (latência do HTTP).

Na próxima seção, descrevemos alguns trabalhos que, dentro da visão de tratar a replicação pelo lado cliente, propuseram e avaliaram diferentes políticas para seleção de servidores entre aqueles que hospedam réplicas de um mesmo recurso.

3.2.4 Políticas de seleção de servidores

Os trabalhos que tratam a replicação na Web, na questão específica do acesso pelo lado do cliente, tomam como premissa a existência de servidores replicados e geograficamente distribuídos na Internet. Isso está associado ao fato de que somente com essa estrutura de replicação é que se pode obter uma maior flexibilidade no acesso.

Nessa abordagem, várias pesquisas (Amini *et al.*, 2003; Sayal *et al.* 1998; Rodriguez e Biersack, 2002; Hanna *et al.*, 2001; Dykes *et al.*, 2000; Yoshikawa *et al.*, 1997; Crovella e Carter, 1995) têm sido direcionadas para o desenvolvimento de algoritmos e mecanismos de seleção de servidores com conteúdos replicados, com foco, principalmente, em propor arquiteturas para facilitar o redirecionamento das requisições dos clientes, ou simplesmente na avaliação de políticas de escolha de servidor que possam oferecer melhor desempenho.

Quando um mesmo serviço ou cópia de documento existe em vários servidores, escolher o servidor que apresente o melhor tempo de resposta não é uma tarefa simples. Uma das razões é que o desempenho experimentado pelo cliente pode variar bastante, dependendo do servidor que seja selecionado. Ao se escolher um servidor que apresente o melhor tempo de resposta, num dado momento, seu desempenho pode mudar de acordo com o tempo, em virtude da carga na rede ou na carga do servidor.

As principais políticas de seleção estudadas no nosso trabalho estão distribuídas em três categorias: estáticas, estatísticas e dinâmicas. Cada uma dessas categorias é descrita em detalhes a seguir.

3.2.4.1 Políticas estáticas

As políticas estáticas estão baseadas em informações sobre a configuração da rede ou outros recursos de infra-estrutura, tais como o número de saltos de roteamento entre o cliente e o servidor (número de *hops*), núcleos de concentração de roteadores, etc. As

métricas utilizadas para a seleção de servidores levam em conta a capacidade dos recursos, mas não a disponibilidade ou concorrência no acesso a esses recursos.

Entre essas políticas, a mais utilizada é a que toma como base o número de *hops*. Como exemplo, encontramos os trabalhos de Hanna *et al*. (2001), Obraczka e Silva (2000), Sayal (1998) e Crovella e Carter (1995). A política baseada na existência de núcleos de concentração de roteadores também é utilizada nos trabalhos de Hanna *et al*. (2001) e Obraczka e Silva (2000).

Os resultados experimentais descritos por esses trabalhos mostraram que há uma baixa correlação entre os números obtidos pelas políticas estáticas e a seleção dos servidores de melhor desempenho.

3.2.4.2 Políticas estatísticas

As políticas estatísticas estão baseadas em dados previamente coletados, que são submetidos a tratamentos estatísticos. As métricas utilizadas refletem os níveis médios de concorrência na rede e na carga do servidor, num dado período.

Algumas políticas estatísticas foram avaliadas no trabalho de Dykes *et al*. (2000). Nesse trabalho, foram utilizadas políticas baseadas na latência e na largura de banda percebidas no lado do cliente. Essas informações foram obtidas computando-se a mediana dos valores obtidos em requisições anteriores à seleção do servidor. A política baseada na latência selecionava o servidor com a menor latência mediana, e a política baseada na largura de banda selecionava o servidor que apresentasse a maior largura de banda mediana em transferências anteriores.

As conclusões de Dykes *et al.* (2000) foram de que o desempenho obtido pelas políticas estatísticas está diretamente relacionado com os dados e o intervalo de tempo utilizados na fase de calibração. Durante os experimentos, dependendo da variação da latência e largura de banda, um número maior de medições por período de tempo era necessário para que os dados estatísticos melhor refletissem os desempenhos dos servidores no momento da invocação.

3.2.4.3 Políticas dinâmicas

As políticas dinâmicas estão divididas em três subcategorias. Na primeira, são utilizados mecanismos que detectam as condições da rede e carga do servidor, mediante a utilização de sondas (*probes*) na forma de *ping*, *download* de pequenos arquivos, etc, num instante imediatamente anterior ao do envio da requisição. Na segunda, as requisições são enviadas de forma paralela ao conjunto de servidores que disponibilizam réplicas do recurso. Na terceira, o servidor é selecionado randomicamente. Dentro da primeira subcategoria, temos os trabalhos de Hanna *et al.* (2001) e Dykes *et al.* (2001). Na segunda, temos o trabalho de Rodriguez e Biersack (2002). A política de seleção randômica é avaliada no trabalho de Dykes *et al.* (2001).

Hanna *et al.* (2001) apresentam uma estratégia de seleção de servidores baseada em duas técnicas. A primeira, denominada *ping-random*, utiliza os tempos (*round-trip times - RTT)* obtidos pelo envio de uma requisição de eco (*ping*) aos servidores. Uma segunda utiliza *download* de pequenos arquivos. A primeira política foi estruturada num processo em duas etapas. Na primeira etapa, a partir do conjunto total de servidores, um subconjunto com os 5 melhores servidores é isolado. Na etapa seguinte, a seleção de um dos 5 servidores é feita de forma randômica, por um período de 10 dias, após o qual o processo é reiniciado. O subconjunto de 5 servidores é isolado tomando como base os cinco menores tempos obtidos através do envio de uma série de pacotes ICMP (*pings)* dos clientes para o conjunto total de servidores. Hanna *et al.* (2002) concluem que a métrica *RTT* e *download* de pequenos arquivos está melhor correlacionada com os tempos reais de transferência de arquivos, ao invés dos tempos obtidos por medidas estáticas do tipo número de *hop*s. Os servidores selecionados por RTT tiveram, em média, um desempenho melhor que aqueles escolhidos pelas políticas estáticas. Para grandes arquivos, *RTT* não se mostrou como uma métrica suficientemente adequada para a escolha de servidor com o melhor desempenho.

Dykes *et al.* (2001) também avaliaram uma política baseada em *probe*. A técnica utilizada consistia no envio de uma solicitação de conexão TCP para todos servidores. O primeiro a responder ao pedido de conexão era selecionado. Essa política dinâmica foi combinada com uma política estatística baseada na largura de banda, criando uma política híbrida. Essa política considerava um subconjunto de *n* (=3) servidores com a maior largura de banda mediana. Para esse grupo de servidores eram enviadas concorrentemente solicitações de conexão TCP, sendo selecionando aquele que primeiro respondesse. Uma

terceira política dinâmica consistia no envio de pedidos concorrentes de conexão TCP a todos servidores. Após receber a primeira resposta, o algoritmo aguardava a metade do tempo da primeira resposta e verificava se algum outro servidor respondia nesse tempo. Se houvesse mais de uma resposta, era escolhido o servidor, dentre os que responderam, com a menor largura de banda mediana. Dykes *et al*. (2001) avaliaram ainda uma política de seleção randômica. A principal conclusão do trabalho de Dykes *et al*. (2001) foi de que o mecanismo de seleção de servidores, pelo lado do cliente, deve fazer uso de *probes* simples ao invés da utilização de dados estatísticos obtidos em transferências anteriores.

Rodriguez e Biersack (2002) argumentam que, ao invés de escolher o servidor mais rápido entre as réplicas, a experiência do usuário é melhorada, e o desempenho obtido é mais uniforme, ao estabelecer conexões simultâneas com os vários servidores que disponibilizem o mesmo conteúdo. A estratégia por eles apresentada está direcionada ao *download* de documentos e arquivos com tamanhos de centenas de *kilobytes*, contendo software, musica, vídeo ou imagens. Nessa política, o usuário pode efetuar o *download* de diferentes partes de um mesmo documento de forma paralela, a partir de vários servidores. Uma vez recebidas, as partes são remontadas para compor o documento original. A implementação proposta permite sua incorporação a navegadores sem exigir qualquer modificação do lado dos servidores. Para determinar o tamanho inicial do documento, uma requisição HTTP é enviada aos servidores. Essa abordagem foi implementada em algumas aplicações para troca de arquivos na arquitetura ponto-a-ponto, também conhecidas como P2P. Os experimentos realizados por Rodriguez e Biersack mostram que a utilização do acesso paralelo dinâmico permite um desempenho superior ao acesso individual de cada uma das réplicas, mesmo nos casos em que os clientes estão conectados por meio de modems.

Os trabalhos descritos acima tratam a questão do acesso a recursos replicados na Web, na forma restrita de dados de conteúdo (documentos HTML, imagens, etc), ou de serviços tradicionais, tais como *ftp*, *telnet* e *chat*. Na próxima seção abordamos a questão específica do acesso a serviços Web replicados ou funcionalmente equivalentes.

3.3 Estratégias de acesso para serviços Web replicados

Nesta seção, fazemos uma análise do suporte oferecido pelos *frameworks* tradicionais para serviços Web, num contexto do acesso a serviços replicados. Em seguida,

descrevemos trabalhos mais recentes que propõem políticas específicas para a seleção dinâmica de serviços Web.

3.3.1 Acesso através dos *frameworks* tradicionais

A chamada a uma operação de um serviço Web se assemelha ao modo como um método de um objeto distribuído (por exemplo, em CORBA ou DCOM) é acionado. A aplicação cliente conta com uma classe *stub* que tem como papel principal disponibilizar as operações remotas do objeto com o qual se deseja interagir. A implementação, propriamente dita, da operação está no objeto remoto. Na classe *stub*, a implementação da operação contempla as ações necessárias para preparar, enviar e receber uma requisição do lado cliente até o servidor onde o objeto se encontra.

Os *frameworks* que suportam a tecnologia de serviços Web incorporam mecanismos que geram de forma automática, a partir de um documento WSDL de um dado serviço, o conjunto de classes auxiliares, incluindo a classe *stub* e suas interfaces, necessárias para a invocação de um serviço, pela aplicação cliente.

Um documento WSDL pode descrever as várias localizações onde um serviço Web replicado pode ser localizado. Considerando essa possibilidade, analisamos os três principais *frameworks* para serviços Web, disponíveis atualmente, com relação ao suporte que cada um oferece para facilitar o acesso a serviços Web replicados.

O trabalho de análise concentrou-se mais especificamente em explorar como cada um desses *frameworks* interpreta um documento WSDL que tenha em sua estrutura um elemento <service> com vários subelementos <port> contendo diferentes endereços de localização (URLs) para um mesmo serviço Web.

A investigação consistiu na execução do módulo de geração do *stub* e demais classes, submetendo, num primeiro momento, um documento WSDL com apenas um subelemento <port> e, num segundo momento, o mesmo documento WSDL, porém com mais de um subelemento <port>. Os resultados obtidos são apresentados nas seções seguintes. O documento utilizado foi o *inquire.wsdl* que descreve o serviço Web UDDI *Business Registry*[2] (OASIS, 2002). A Figura 3.3 mostra um trecho do documento contendo

[2] Esse serviço é utilizado como estudo de caso na avaliação experimental descrita no capítulo 5.

dois endereços de localização do serviço. Para facilitar a análise das classes geradas, apenas duas das operações disponibilizados por esse serviço foram consideradas.

```
<service name="UDDIWebService">
 <port name="UDDIWebServiceSoapMS" binding="s0:UDDIWebServiceSoap">
    <soap:address location="http://uddi.microsoft.com/inquire.asmx" />
 </port>
 <port name="UDDIWebServiceSoapIB" binding="s0:UDDIWebServiceSoap">
    <soap:address location="http://www-3.ibm.com/services/uddi/inquiryapi" />
 </port>
 </service>
```

Figura 3.3: Segmento de um documento WSDL com dois elementos <port>.

Os *frameworks* analisados foram:

- AXIS (versão 1.0), do grupo Apache (Apache, 2002);
- *Java Web Service Developer Pack* – JWSDP (versão 1.1), da SUN (SUN, 2003);
- .NET (versão 1.0), da Microsoft (Microsoft, 2002a).

Os resultados da análise de cada um dos três *frameworks* são descritos a seguir.

3.3.1.1 AXIS

O *framework* AXIS disponibiliza a classe utilitária *org.apache.axis.wsdl.WSDL2Java* que, a partir de um documento WSDL, faz a geração automática de uma classe *stub*, uma classe de localização (*locator*), e da interface para o serviço a ser acessado. A Figura 3.4 ilustra as classes geradas pelo AXIS a partir da interpretação do WSDL onde está especificado apenas um subelemento <port>. São elas: *UDDIWSSoapLocator*, *UDDIWebServiceStub*, e a interface *IUDDIWebService*, O endereço de localização do serviço é um atributo da classe *locator*.

A classe *stub* gerada expõe as operações disponibilizadas pelo serviço Web, as quais estão declaradas na Interface *UDDIWebService*. A classe de localização *(locator*) disponibiliza dois métodos de nome *getUDDIWebService()*, a partir dos quais pode ser obtida uma instância de um objeto da classe *stub*. No primeiro método, o endereço de destino é aquele configurado no atributo da classe, definido a partir do WSDL. No

segundo, pode-se especificar um outro endereço na instanciação do *stub*. Esse método segue um padrão de nome que corresponde ao atributo *name* do elemento <port> no documento WSDL. A aplicação cliente instancia um objeto *locator*, e na chamada da operação *getUDDIWebService()*, obtém a referência do objeto *stub*. A partir daí, a aplicação cliente pode invocar qualquer uma das operações disponibilizadas pelo serviço.

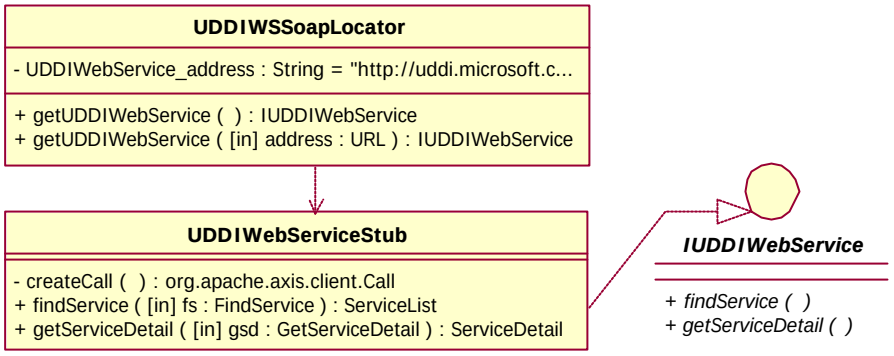

Figura 3.4: Classes e interfaces geradas de um documento WSDL com um elemento <port> pelo *framework* AXIS.

A classe *Call*, utilizada pela classe *stub*, abstrai da aplicação cliente as camadas inferiores (SOAP, transporte HTTP, etc) que estão implementadas no *framework*.

Ao submetermos a descrição do serviço com mais de um subelemento <port> ao AXIS, o processo de geração altera a estrutura da classe l*ocator*, disponibilizando um método para cada elemento <port> encontrado. A classe *stub* permanece com a mesma estrutura. A Figura 3.5 apresenta a nova estrutura da classe *locator*. A aplicação cliente instancia um objeto *locator*. A partir daí, ela deverá chamar uma das quatro operações disponibilizadas para obter uma instância da classe *stub*. Cada método tem um atributo associado que aponta para o endereço correspondente ao método selecionado. Com a nova estrutura, ainda é possível a aplicação cliente especificar em qual endereço o serviço pode ser acionado. Para cada uma das operações da classe *locator* há uma outra correspondente que recebe uma URL como parâmetro, permitindo a chamada do serviço em endereço diferente daquele definido como padrão nos atributos da classe. Essa funcionalidade é implementada através da operação *getUDDIWebServiceMS([in] add:URL)*. As outras três operações foram omitidas na Figura 3.5.

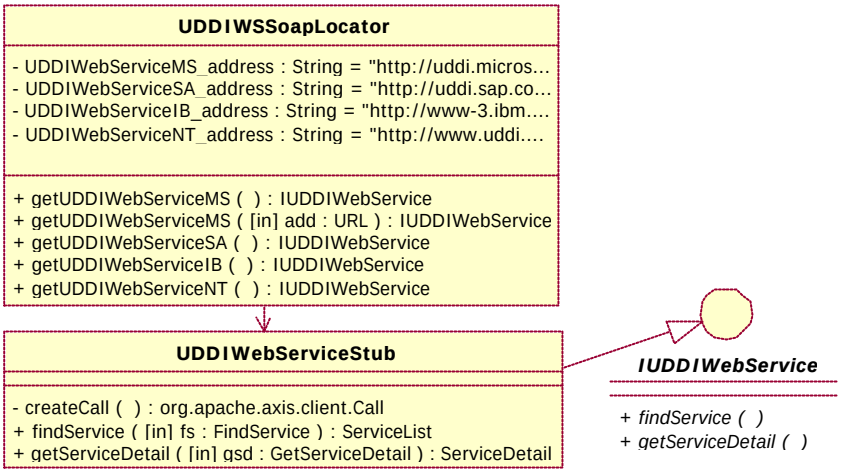

Figura 3.5: Classes e interfaces geradas de um documento WSDL com quatro elementos <port> pelo *framework* AXIS.

3.3.1.2 JWSDP

O *framework* JWSDP também disponibiliza classes utilitárias que automatizam o tratamento de um documento WSDL. A classe principal é a *com.sun.jwsdp.launcher.WsCompileExecutable*. A Figura 3.6 apresenta o resultado da interpretação do WSDL com apenas um endereço de localização do serviço pelo JWSDP.

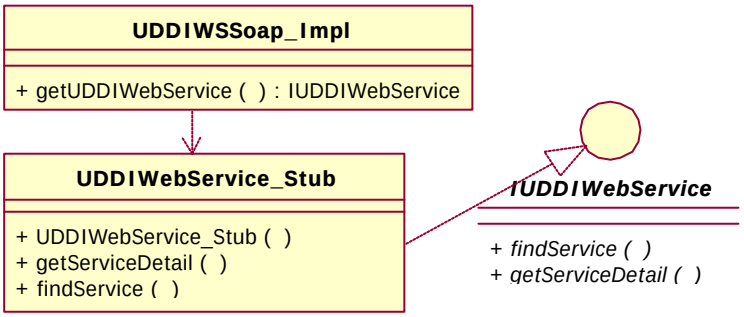

Figura 3.6: Classes e interfaces geradas de um documento WSDL com um elemento <port> pelo *framework* JWSDP.

O processo de tratamento de um documento WSDL gera um número de classes semelhante aquele gerado pelo AXIS. Temos a classe *UDDWSSoap_Impl*, que tem um papel semelhante à classe *locator* no AXIS. Essa classe é responsável pela instanciação do objeto *stub*. A informação da localização (URL) do serviço é passada de forma estática no construtor da classe *stub*.

Para um documento WSDL que contém vários pontos de localização do serviço, o JWSDP gera uma classe *stub* para cada um dos subelementos <port> existentes no WSDL.

Cada classe *stub* leva o nome do atributo *name* que identifica o subelemento <port> no elemento <service>. A Figura 3.7 ilustra o diagrama com as classes geradas. Caberá a aplicação cliente instanciar um objeto da classe *UDDIWSSoap_Impl* e, com a referência obtida, utilizar um dos quatro métodos disponíveis, nesta classe, para obter um objeto da classe *stub* correspondente ao endereço onde o serviço deve ser chamado. As estruturas das classes *stub* são idênticas. Os métodos que se repetem, em cada *stub*, foram omitidos na Figura 3.6. Os nomes dos métodos da classe *Impl* e das classes *stub* seguem os nomes dados no atributo *name* de cada um dos subelementos <port> presentes no WSDL.

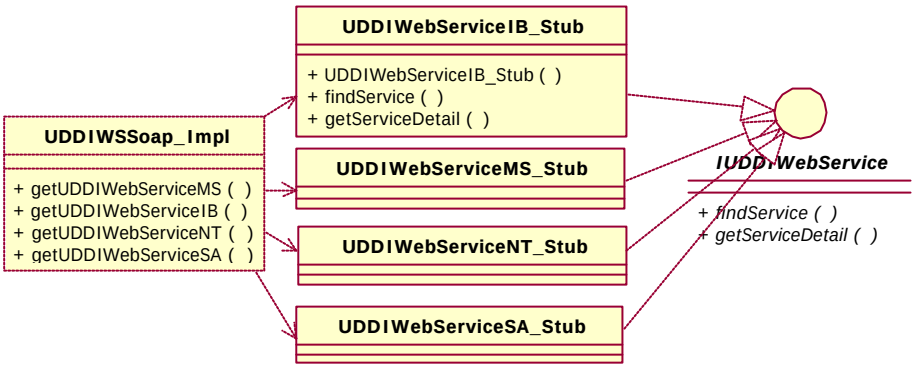

Figura 3.7: Classes e interfaces geradas de um documento WSDL com quatro elementos <port> pelo *framework* JWSDP.

3.3.1.3 .NET

O *framework* .NET disponibiliza duas formas para a geração de classes auxiliares. A primeira utiliza a aplicação WSDL.EXE, que é externa à IDE[3] do *framework*. Dado um documento WSDL, essa aplicação gera a classe *stub* que poderá ser incorporada na estrutura de um projeto do ambiente.NET.

No contexto da própria IDE do *framework*, a geração da classe *stub* pode ser feita pela adição de uma referência Web, na forma de uma URL, para a localização do documento WSDL. O acesso e a leitura do documento WSDL irá gerar a classe *stub*, a qual é automaticamente inserida ao projeto em curso. Em ambas as formas, pelo utilitário WSDL.EXE ou utilizando a IDE, a estrutura da dasse *stub* gerada é a mesma. A Figura 3.8 ilustra o resultado obtido a partir da análise de um documento WSDL contendo um único endereço de um serviço. Temos apenas uma classe que desempenha o papel de *stub* para acesso ao serviço. Não há disponibilização de uma interface para o serviço. De forma

[3] IDE – *Integrated Development Environment*.

similar ao *framework* JWSDP, o endereço de localização é atribuído estaticamente no construtor da classe.

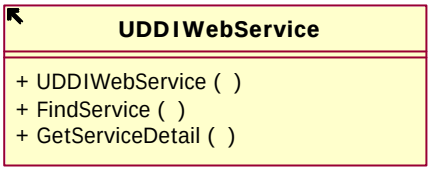

Figura 3.8: Classe gerada de um documento WSDL com um elemento <port> pelo *framework* .NET.

No tratamento de uma descrição de serviço com mais de uma URL de localização, a geração da classe *stub* também segue o mesmo comportamento visto no JWSDP, ou seja, é implementada uma classe *stub* para cada um dos pontos de localização do serviço. O construtor de cada uma das classes incorpora o endereço presente em cada subelemento <port> do WSDL. A aplicação cliente instancia de forma direta o objeto *stub* de acordo com o endereço onde o serviço deve ser buscado. O padrão de nome que as classes seguem não guarda relação com as informações do WSDL. A Figura 3.9 mostra as classes geradas.

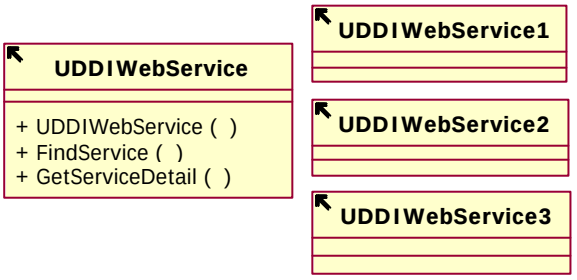

Figura 3.9: Classes geradas de um documento WSDL com quatro elementos <port> pelo *framework* .NET.

3.3.1.4 Discussão

Nos três *frameworks* analisados, observamos dois comportamentos diferenciados no processo de geração de classes auxiliares, na análise de um documento WSDL que inclua vários endereços para um mesmo serviço. No primeiro, é gerado um conjunto de classes específico para cada endereço. No segundo, é gerado um único conjunto de classes, com a classe *locator* oferecendo diferentes opções de acesso para cada endereço. Em ambos os casos, cabe à aplicação decidir para qual endereço envia uma chamada de operação. Portanto, nenhum dos três *frameworks* analisados explora alternativas que possam dar uma

maior transparência e dinamicidade às aplicações clientes no acesso a um serviço replicado, especialmente que se refere à localização e à escolha da réplica do serviço que possa oferecer o melhor desempenho.

3.3.2 Acesso através da seleção dinâmica de servidores

Nesta seção, apresentamos três trabalhos recentes que visam melhorar o desempenho no acesso a serviços Web replicados através de políticas dinâmicas de seleção de servidores.

3.3.2.1 *WebTransact-EM*

Azevedo *et al.* (2003) propõem um mecanismo para a seleção dinâmica de serviços Web baseada em critérios de qualidade e parâmetros de custo monetário. O processo da seleção dinâmica de serviços Web está inserido dentro de uma arquitetura de serviços denominada *WebTransact-EM,* que implementa um modelo para execução dinâmica de serviços Web semanticamente equivalentes. Dos critérios considerados para a escolha dos serviços que devem ser executados, alguns devem ser informados pelo próprio provedor do serviço. Entre eles está o custo de inicialização do serviço, que corresponde ao tempo necessário para que o serviço seja acionado, e o tempo médio da execução de cada operação do serviço. Para a seleção e execução de um serviço, Azevedo *et al.* (2002) propõem três modos: privilegiar o custo monetário, privilegiar o menor tempo de resposta (desempenho) ou privilegiar a melhor relação custo/desempenho. No modo de privilegiar o menor tempo de execução, o trabalho adota a política de invocação dos serviços em paralelo. Assim, os serviços são invocados em paralelo e apenas a primeira resposta obtida é devolvida à aplicação cliente (as demais respostas são descartadas).

O trabalho não discute nenhuma outra política de invocação nem oferece nenhuma evidência de que a invocação em paralelo seja a alternativa de melhor desempenho. A localização dos serviços é realizada manualmente. Um processo de localização mais automatizado é apresentado como sugestão de trabalho futuro.

3.3.2.2 *ServiceGlobe*

Keidl *et al.* (2002) propõem uma abordagem semelhante à de Azevedo *et al.* (2003), no que se refere à invocação em paralelo de serviços que oferecem funcionalidades equivalentes. O processo de invocação dinâmica está inserido dentro de uma arquitetura de serviços denominada *ServiceGlobe*. Nesta arquitetura, um serviço especializado é responsável pela descoberta e invocação dos serviços equivalentes. A descoberta é realizada a partir dos registros UDDI. Uma vez descobertos os serviços de funcionalidades equivalentes, a invocação aos mesmos pode ser conduzida de três modos diferentes:

- Um único serviço é invocado e, em caso de falha, invoca-se um outro serviço equivalente;
- Apenas uma parte dos serviços existentes é chamada em paralelo. Um parâmetro define a quantidade de serviços a serem invocados.
- Todos os serviços são invocados em paralelo.

No modo em que um serviço é invocado isoladamente, não há uma regra definida para a seleção do servidor a ser utilizado, apontando para algo semelhante a um processo de seleção aleatório. Não há um foco claro, no trabalho, em privilegiar o tempo de resposta, visto que nos modos onde ocorrem as chamadas em paralelo existe uma parametrização para permitir o recebimento das respostas de todos os serviços invocados, e não apenas daquele que responda primeiro. O trabalho também não apresenta nenhuma evidência com relação à efetividade dos processos de invocação adotados, ou seja, qual dos modos ofereceria o melhor desempenho na prática. Por fim, existem algumas restrições com respeito ao tempo máximo em que a resposta pode ser obtida inerentes ao modelo proposto. Se um ou mais serviços não responderem no tempo máximo estabelecido, outros serviços podem ser chamados, o que pode resultar num tempo de resposta demasiadamente longo para a aplicação cliente.

3.3.2.3 Seleção baseada em agentes móveis e consultas por RPC

Recentemente, Padovitz *et al.* (2003) propuseram mecanismos para a seleção dinâmica de serviços Web, com foco na obtenção de informações sobre os serviços em uma etapa anterior à invocação propriamente dita. Eles propõem um sistema, que está em fase de desenvolvimento, para recuperar informações dos provedores dos serviços de

acordo com restrições e especificações definidas pelo cliente, tais como tempo de resposta, disponibilidade e qualidade de serviço. A partir das informações obtidas, o sistema decide qual serviço Web deverá ser invocado. Para a busca das informações nos provedores, duas abordagens são propostas, uma baseada em RPC (*Remote Procedure Call*) e outra baseada em agentes móveis. Na abordagem com RPC, as requisições são enviadas aos provedores em paralelo. Na abordagem com agentes móveis, os agentes são enviados aos provedores de serviços de duas formas: na primeira, um agente é enviado a cada um dos provedores e na outra, um único agente circula entre os provedores dos serviços a serem avaliados. Cada agente leva consigo as restrições definidas pelo cliente. Por ser um trabalho ainda incipiente, não foi feita nenhuma avaliação das abordagens propostas. A localização dos serviços é realizada de forma manual.

3.4 Sumário

Neste capítulo, descrevemos várias estratégias de replicação na Web, com maior ênfase nos trabalhos que exploraram o acesso aos recursos replicados, incluindo serviços Web, pela visão do lado do cliente. Apresentamos, também, um trabalho de investigação, conduzido com três *frameworks* tradicionais, com o objetivo de analisar o tratamento que cada um faz ao processar um documento WSDL que descreva várias localizações para um serviço Web replicado.

No capítulo seguinte, apresentamos um *framework* que dá suporte à invocação transparente de serviços Web replicados, baseado em diferentes políticas de seleção de servidores.

Capítulo 4

RWS – Um *Framework* para Invocação de Serviços Web Replicados

Este capítulo apresenta o framework RWS, cujo objetivo é viabilizar a invocação transparente de serviços Web replicados baseada em diferentes políticas de seleção de servidores.

Ao considerarmos a existência de serviços Web replicados, vem à tona a necessidade de se localizar e tratar adequadamente as múltiplas réplicas do serviço. Uma vez identificadas as réplicas, outra questão importante é decidir para qual servidor, dentre os que disponibilizam réplicas do serviço, uma chamada de operação deve ser enviada. Essas duas questões, localização e seleção das réplicas, podem ser implementadas na forma de um *framework*, dando às aplicações clientes transparência no processo de localização e acesso ao serviço.

Neste capítulo, apresentamos o *framework* RWS (*Replicated Web Services*) (Silva e Mendonça, 2004), que tem como objetivo atender às questões aqui discutidas referentes à localização e, principalmente, à invocação transparente das réplicas de um serviço Web replicado. Na próxima seção, discutimos a questão da localização das réplicas de um serviço através da sua descrição WSDL. Em seguida, damos uma visão geral da estrutura do *framework* AXIS, sobre o qual o *framework* RWS foi baseado. Por fim, descrevemos em mais detalhes o *framework* proposto, incluindo sua estrutura, o processo de tratamento das réplicas, e as políticas de seleção de servidores consideradas.

4.1 Localização das réplicas

O acesso a recursos replicados no lado cliente exige, entre outras informações, a identificação e a localização das réplicas. No contexto de serviços Web, informações sobre as réplicas que estão disponíveis para um determinado serviço, bem como informações sobre a localização de cada uma, podem ser obtidas diretamente a partir do documento WSDL definido para o serviço.

Num documento WSDL, as informações necessárias para a localização e a invocação de um serviço estão descritas no elemento <service>. Esse elemento inclui um elemento <port> com a descrição do tipo de protocolo de ligação (HTTP, SMTP, etc) através do qual o serviço pode ser acessado. <port> por sua vez inclui o elemento <address> com a descrição do endereço do serviço na Internet.

Um mesmo serviço Web pode ser invocado via diferentes protocolos de ligação. Por esse motivo, a especificação WSDL prevê que um elemento <service> pode conter zero ou mais elementos do tipo <port>. No entanto, a especificação não impõe nenhuma restrição quanto ao valor do elemento <address> de cada elemento <port>. Portanto, através da inclusão de múltiplos elementos <port> também é possível definir diferentes endereços para um mesmo serviço. No caso de serviços Web replicados, esses valores poderiam representar os endereços das suas respectivas réplicas. A Figura 3.3 mostra um exemplo de um segmento do documento WSDL com a descrição de dois endereços que apontam para o mesmo serviço.

Escolher a melhor réplica de um serviço replicado não é uma tarefa trivial. Variações na latência e na capacidade de transmissão da rede, e na capacidade de conexão e processamento dos clientes, entre outras, são fatores que podem ser decisivos para determinar a melhor réplica a ser invocada pela aplicação cliente. Para explorar estas características, torna-se necessário dar um tratamento adequado à informação sobre a existência de réplicas contidas no documento WSDL de um serviço. Uma vez obtida essa informação, a mesma poderá ser incorporada diretamente à aplicação, ou a camadas próximas a ela, como nas classes auxiliares geradas pelo *framework* de serviços Web utilizado.

Considerada a existência das réplicas, e com os seus respectivos endereços devidamente identificados, o que interessará à aplicação é que a requisição enviada ao serviço replicado seja atendida da melhor forma possível, não importando para qual réplica a solicitação seja encaminhada. Para tornar a escolha da réplica transparente para a aplicação, o ideal é que as camadas auxiliares, na forma de *stub*, ou na forma de um servidor *proxy* inteligente, incorporem mecanismos que busquem a invocação do modo mais eficiente. Nesse contexto, propomos o *framework* RWS¸ que permite a geração automática das camadas que estarão ligadas à aplicação cliente, incluindo uma classe *stub*, com a incorporação da informação necessária para escolher e invocar, automaticamente, de acordo com diferentes critérios, as réplicas de um serviço Web replicado.

O *framework* RWS foi implementado como uma extensão do *framework* AXIS (Apache, 2002). Assim, primeiramente daremos uma visão geral da estrutura do AXIS e em seguida descrevemos como ele foi estendido para incorporar as novas funcionalidades descritas na seção 4.3.

4.2 Visão geral do *framework* AXIS

O *framework* AXIS provê um conjunto de classes necessárias para o desenvolvimento e execução de serviços Web na plataforma Java 2, em ambos os lados, cliente e servidor. Do lado do servidor, o AXIS oferece toda a infra-estrutura de software necessária à integração com um *container* Java, para a disponibilização e execução de serviços Web. Do lado do cliente, o AXIS disponibiliza as camadas de *software* necessárias para o empacotamento, envio e recebimento de chamadas de operações aos serviços, como também as ferramentas utilitárias para a geração das classes auxiliares que vão estar mais diretamente integradas à aplicação.

A Figura 4.1 mostra uma visão simplificada da estrutura do AXIS do lado do servidor. A Figura 4.2 ilustra a estrutura do lado do cliente, destacando as classes que estão mais próximas da aplicação e os serviços adicionais necessários ao empacotamento e transporte das chamadas de operações.

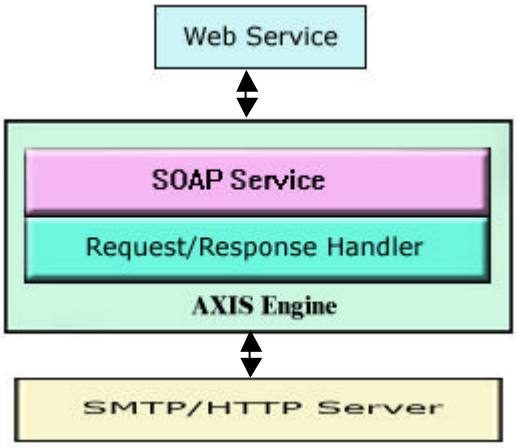

Figura 4.1: Estrutura básica do AXIS no lado do servidor (Apache, 2002).

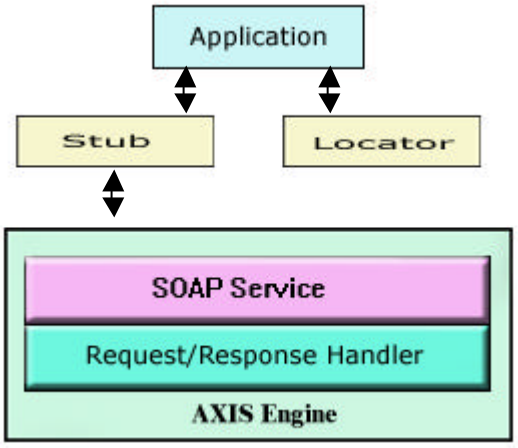

Figura 4.2: Estrutura básica do AXIS no lado do cliente (Apache, 2002).

Uma aplicação, ao utilizar a estrutura tradicional do AXIS, deve inicialmente interagir com a classe *locator*, para obter a localização do serviço e, em seguida, com a classe *stub*, para efetuar a chamada de uma operação do serviço cuja interface é disponibilizada pela casse *stub*, de forma local. A partir daí, a classe *stub* interage com os demais serviços do AXIS, que serão responsáveis por, entre outras tarefas, serializar os parâmetros de chamada na forma de mensagens SOAP, enviar as mensagens através do protocolo HTTP, receber e desempacotar a resposta a ser devolvida à aplicação. Esse processo está ilustrado, de forma simplificada, no diagrama de seqüência da Figura 4.3. A operação *getServiceDetail*, utilizada no diagrama, é uma operação disponibilizada pelo serviço *UDDI Business* Registry, que foi utilizado como estudo de caso na avaliação experimental descrita no próximo capítulo.

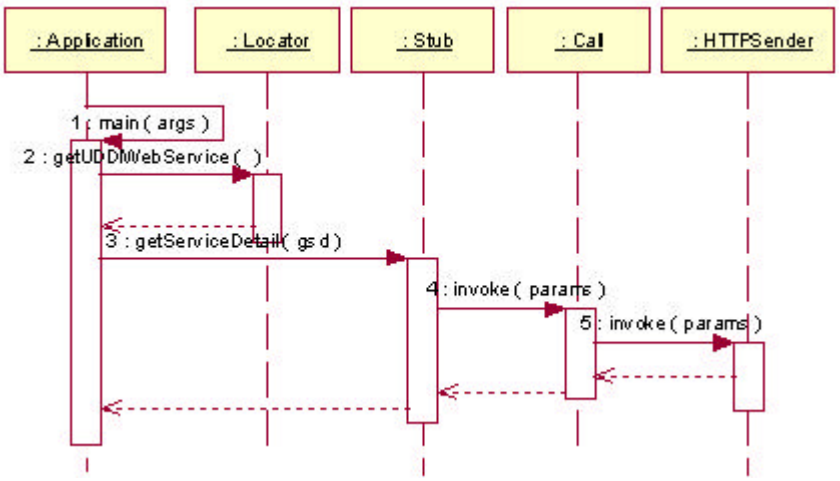

Figura 4.3: Processo de invocação de um serviço web utilizando o *framework* AXIS.

4.3 O *framework RWS*

Na implementação do RWS, estendemos o *framework* AXIS no processo da geração das classes auxiliares que vão estar ligadas diretamente à aplicação cliente, além de alterar o comportamento dessas classes na invocação do serviço. A opção pelo AXIS deve-se ao fato de que o mesmo implementa um processo de geração da classe *stub* mais próximo ao modelo aqui proposto, e também pelo fato do seu código fonte ser totalmente aberto.

O *framework* RWS foi implementado acrescentando dois novos componentes, o Gerente de Seleção *(SelectionManager)* e o Gerente de Invocação (*InvocationManager*), à arquitetura da aplicação cliente originalmente utilizada pelo AXIS, conforme mostra a Figura 4.4. Com o tratamento dado ao documento WSDL, no que se refere à localização, é facultado à aplicação cliente especificar um endereço para acessar um serviço, ou delegar ao objeto *stub* a escolha de um entre os vários disponíveis. Na segunda alternativa, o comportamento da classe *stub* é modificado com a incorporação de políticas de seleção de réplicas, onde a definição sobre para qual réplica será enviada a requisição do serviço poderá ser feita com base em mecanismos que busquem aquele de melhor desempenho. O conjunto formado pela aplicação, a classe *stub* e as demais classes compõem uma estrutura de cliente inteligente que se assemelha ao modelo proposto por Vingralek *et al.* (1999) e Yoshikawa *et al.* (1995). As diferentes políticas de invocação são aplicadas pelo *stub* a partir dos parâmetros fornecidos pela aplicação cliente. Desse modo, o envio da requisição ao serviço passa de uma forma estática, antes definida apenas pela aplicação consumidora, para uma forma dinâmica, que permite a seleção do melhor servidor, no momento da invocação, entre aqueles que hospedam réplicas do serviço.

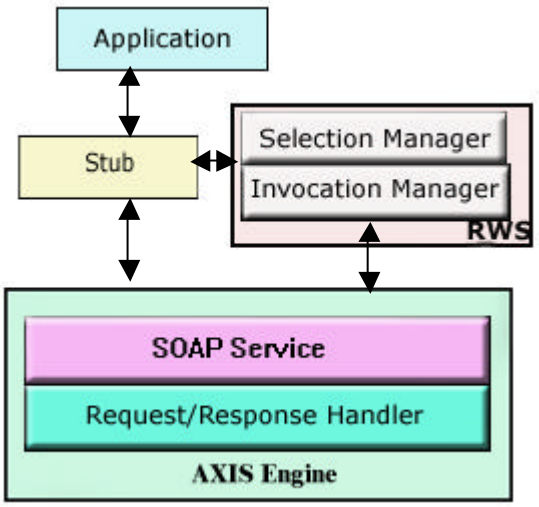

Figura 4.4: Estrutura do *framework* RWS.

O RWS utiliza os serviços do AXIS responsáveis pela execução do processamento de mais baixo nível, como por exemplo, o empacotamento SOAP e o transporte sobre HTTP.

O trabalho de planejamento e implementação do *framework* RWS foi dividido em duas etapas. Na primeira, adotamos uma forma diferenciada em relação ao comportamento dos *frameworks* tradicionais na interpretação das informações de localização de um serviço extraídas de um documento WSDL, alterando o comportamento das classes utilitárias no processo de geração da classe *stub*. Num segundo momento, implementamos novas classes que suportaram a etapa de avaliação de políticas de invocações de serviços Web, conduzida neste trabalho.

Nas próximas seções, descrevemos a implementação do processo de tratamento das réplicas e das diferentes políticas de invocação incorporadas ao *framework* RWS

4.3.1 Implementação do processo de tratamento das réplicas

No *framework* AXIS, a classe *stub*, além do papel tradicional de expor os métodos que compõem a interface do serviço remoto, também parametriza as classes das camadas inferiores (serialização/desserialização XML, empacotamento SOAP, transporte HTTP, etc) com a localização do serviço obtida a partir da sua descrição WSDL.

O RWS trata a identificação dos endereços alternativos, que disponibilizam réplicas para um mesmo serviço, obtidos a partir de seu documento WSDL, de forma transparente para a aplicação cliente. Para isso, foi necessáriao modificar a classe *WSDL2Java*, integrante do AXIS. A nova classe, denominada *WSDL2JavaR*, trata a existência de múltiplos endereços para um serviço através de um processo diferenciado de geração das classes auxiliares que serão incorporadas à aplicação cliente.A Figura 4.5 ilustra, de forma simplificada, o processo de interpretação do WSDL e geração das classes auxiliares no *framework* RWS.

A classe de localização (*locator)*, gerada originalmente pelo AXIS, não é mais gerada. Temos agora a classe *SelectionManager*, cujo comportamento principal é gerenciar o processo de seleção de servidores nas várias políticas de invocação suportadas, passando à classe *stub* um endereço selecionado para onde a chamada de uma operação do serviço replicado deve ser encaminhada. Uma vez instanciado um objeto dessa classe, os endereços onde o serviço pode ser buscado estarão presentes no atributo *replicaList*. A

Figura 4.6 mostra um fragmento da classe *stub* gerada pelo RWS a partir da descrição WSDL do *UDDI Business Registry*. Destacamos a operação *setPolicy()*, através da qual uma instância da classe *stub* pode ser informada sobre qual política de invocação deverá ser realizada no acesso ao serviço Web.

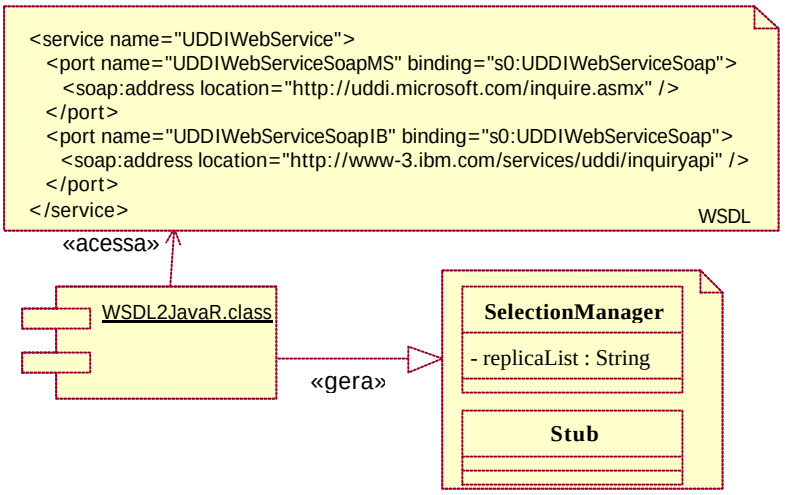

Figura 4.5: Tratamento das réplicas e geração de classes auxiliares no *framework* RWS.

Para realizar uma chamada a uma das operações do serviço replicado, a aplicação cliente deve instanciar um objeto da classe *stub*. Se nenhuma parametrização adicional for realizada, a chamada será enviada ao servidor cujo endereço consta em primeiro lugar na lista de réplicas mantida pela instância da classe *SelectionManager*. A invocação será feita de forma direta a esse servidor, sem a aplicação de nenhuma política de seleção específica. Adicionalmente, fica facultado à aplicação cliente especificar um outro endereço para a invocação do serviço, através da operação *setTargetEndpoint(address:URL)*, também gerada na classe *stub* pelo RWS.

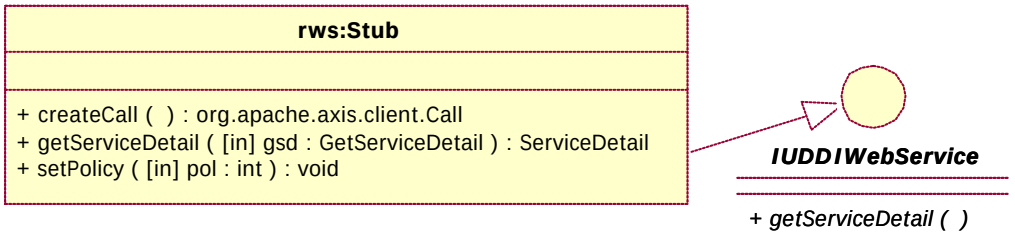

Figura 4.6: Exemplo de classe *stub* gerada a partir de um documento WSDL pelo *framework* RWS.

4.3.2 Implementação das políticas de seleção de servidores

Além do tratamento do endereço de localização das réplicas de um serviço Web replicado, o RWS foi estruturado de forma a suportar a implementação de diferentes políticas de seleção de servidores, através das quais poderíamos avaliar diferentes estratégias de escolha prévia de um servidor, antes do envio de uma chamada de operação.

Para viabilizar a escolha do servidor, o comportamento da classe s*tub* foi modificado para permitir a interação com uma instância da classe *SelectionManager* antes do envio da chamada de operação ao serviço remoto. Esse processo de seleção prévia somente não é utilizado na aplicação da política Paralela. Nesse caso, cabe à instância da classe *stub* obter a lista de endereços das diversas réplicas, a partir da instância da classe *SelectionManager*, e então acionar uma instância da classe *InvocationManager*, a qual ficará encarregada de realizar as invocações em paralelo. De acordo com a política de seleção definida, o objeto da classe *SelectionManager* instancia as classes necessárias para realizar a seleção do servidor, ressalvado o caso da política Paralela.

As políticas de seleção de servidores implementadas no contexto do *framework* RWS são as seguintes: política Randômica, política Paralela, política HTTPing, política Melhor Última e política Melhor Mediana. Nas próximas seções, descrevemos em detalhes a implementação de cada uma dessas políticas.

4.3.2.1 Política Randômica

Com esta política, procuramos avaliar o desempenho experimentado pela estratégia de selecionar de forma aleatória um servidor dentre o conjunto de servidores que disponibilizam réplicas para um dado serviço Web replicado, o servidor para o qual será enviada uma chamada de operação. A Figura 4.7 mostra um diagrama simplificado das classes envolvidas na implementação da política Randômica, e a Figura 4.8 ilustra como elas interagem em tempo de execução.

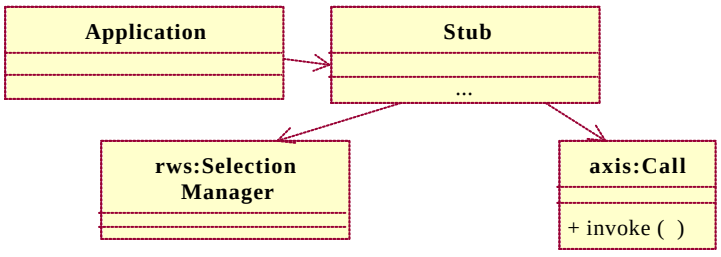

Figura 4.7: Diagrama simplificado das classes envolvidas na implementação da política Randômica.

Nessa política, a aplicação cliente informa ao objeto *stub* que a seleção será feita de forma aleatória. A escolha do servidor é efetuada pela classe *SelectionManager*, a partir da geração de um número randômico dentro do intervalo da quantidade de endereços disponíveis para o serviço. Em seguida, a chamada da operação é enviada ao servidor escolhido, através de um objeto da classe *Call*, que representa uma chamada de operação na infra-estrutura originalmente provida pelo AXIS. A política de seleção de servidores de forma aleatória também foi aplicada por Dykes *et al.* (2000).

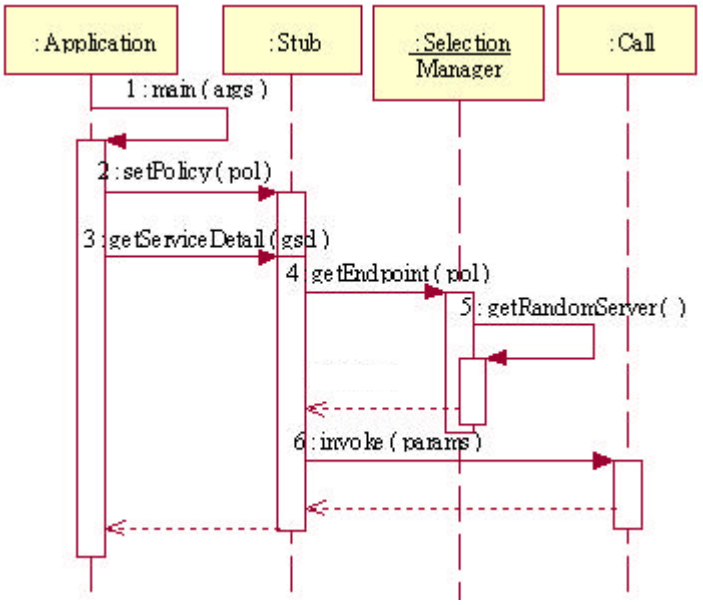

Figura 4.8: Processo de invocação na política Randômica.

4.3.2.2 Política Paralela

Esta política tem como objetivo avaliar o desempenho do envio de uma chamada de operação de um serviço Web, de forma concorrente, a todos os servidores que disponibilizem uma réplica do serviço.

Na implementação dessa política, duas classes desempenham papéis principais: a classe *ServiceInvocationManager*, que tem o papel de gerenciar a instanciação e execução concorrente de objetos da classe *ServiceInvocationThread*, que por sua vez, tem o papel de enviar a chamada de uma operação a cada servidor. A Figura 4.9 ilustra um diagrama simplificado das classes envolvidas na implementação dessa política.

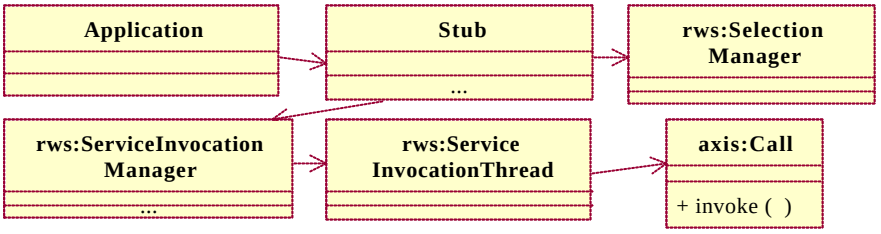

Figura 4.9: Diagrama simplificado das classes envolvidas na implementação da política Paralela.

Na execução desta política, a aplicação cliente informa ao objeto *stub* que a chamada da operação será enviada a todos os servidores de forma paralela. A chamada é encaminhada através de múltiplas *threads*. A quantidade de *threads* instanciadas e executadas equivale ao número de endereços onde o serviço está disponível. Para cada *thread* é instanciado um novo objeto da classe *Call*. A *thread* que recebe a resposta do servidor em primeiro lugar a repassa à instância da classe *InvocationManager*, que por sua vez a repassa ao objeto *stub*, daí chegando à aplicação cliente. As demais *threads* são interrompidas e as respostas parciais, quando recebidas, são ignoradas.

Esta política de invocação em paralelo é uma modificação do método proposto por Rodriguez e Biersack (2002), utilizado para *download* de arquivos. É também uma implementação do processo de invocação em paralelo proposto por Azevedo *et al.* (2003) e Keidl *et al.* (2002). A seqüência de interações entre as classes que implementaram a política Paralela está ilustrada nos diagramas das Figuras 4.10 e 4.11.

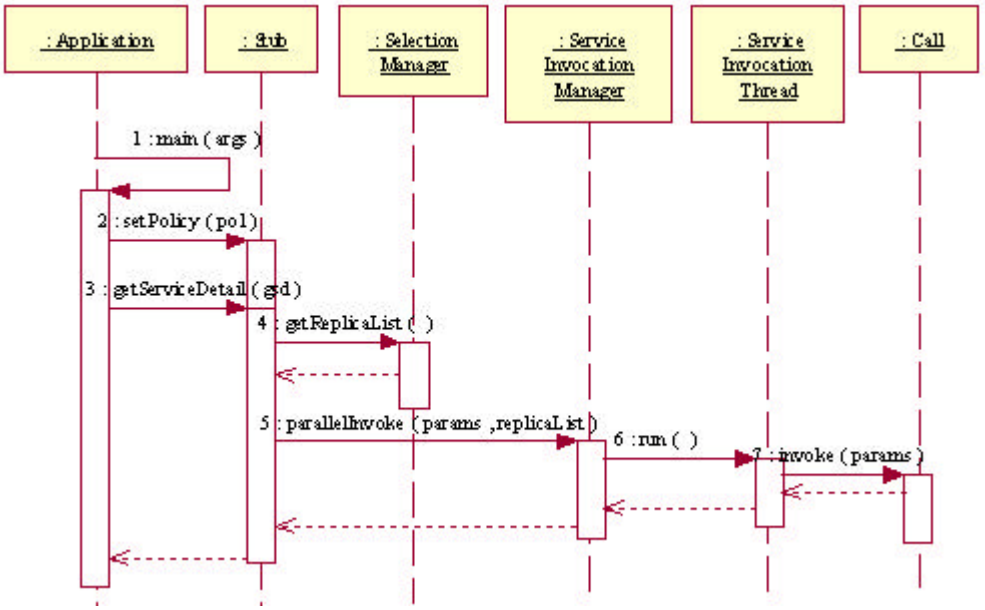

Figura 4.10: Processo de invocação na política Paralela.

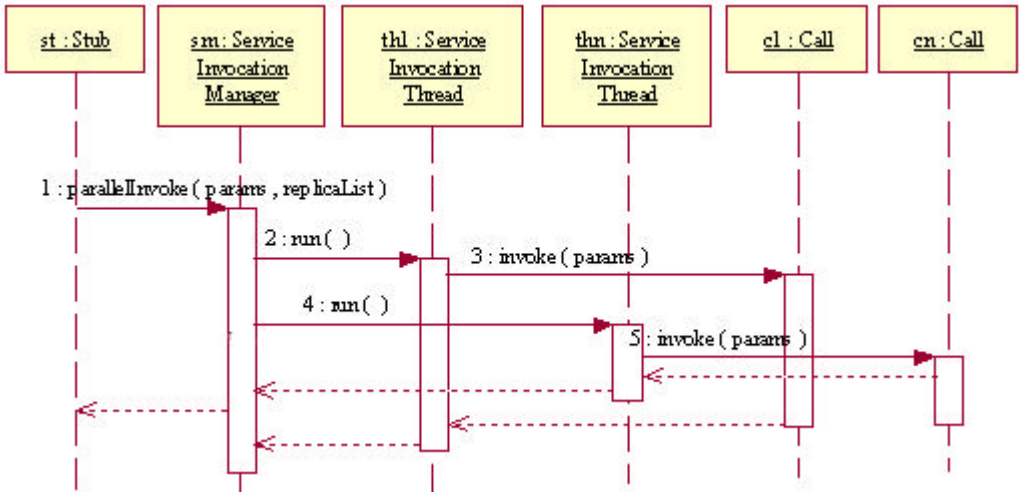

Figura 4.11: Processo de invocação na política Paralela (cont.).

4.3.2.3 Política HTTPing

Esta política de invocação é baseada num processo de seleção dinâmica, no qual é avaliado, num primeiro momento, o comportamento em nível de rede e de serviço HTTP de cada um dos servidores que hospedam uma réplica do serviço Web, antes do envio propriamente dito da chamada da operação. Com esta política, procuramos avaliar a hipótese de que o servidor que apresenta o melhor nível de serviço de rede e HTTP, no instante imediatamente anterior ao envio da chamada, possa atender a uma invocação do serviço de forma mais rápida. Essa hipótese está associada aos resultados positivos obtidos por Dykes *et al.* (2000) na seleção de servidores por técnica semelhante, no contexto da requisição de documentos HTML e imagens.

A política HTTPing é implementada como uma modificação do método que foi proposto por Dykes *et al.* (2000). Naquele trabalho, foi levado em conta o comportamento em nível de rede relativo apenas à troca dos pacotes iniciais no estabelecimento de uma conexão TCP, através de um mecanismo de sonda denominado *TCPing*. A motivação para a implementação da política HTTPing está relacionada com a necessidade de avaliar não somente os níveis de serviço de rede, mas também o nível de carga no serviço HTTP de cada servidor. Uma outra razão foi para contornar algumas limitações de natureza técnica, em virtude, por exemplo, da ausência de resposta, pela maioria dos servidores na Internet, aos pacotes ICMP enviados na forma de *pings*. Além disso, é comum a utilização de servidores *proxy*, do lado do cliente, com a função de controle de acesso à Internet, o que normalmente implica no bloqueio do envio/recebimento de pacotes utilizados pela técnica

TCPing. Esse cenário esteve presente nas duas estações clientes utilizadas nos experimentos descritos no capítulo 5.

A implementação da política HTTPing consiste no envio de uma requisição do tipo HTTP HEAD[1], no instante imediatamente anterior ao envio da chamada da operação ao serviço. A requisição HTTP é enviada de forma concorrente a todos os servidores que hospedam o serviço Web replicado. Aquele que primeiro responder à requisição é selecionado para a invocação do serviço.

Duas classes adicionais foram implementadas para essa política. A classe *HTTPingHead* tem o papel de gerenciar a instanciação e controlar a execução de objetos da classe *HTTPingThread*. Esta segunda classe tem como papel o envio de uma requisição HTTP HEAD para cada um dos servidores com réplicas disponíveis. A Figura 4.12 ilustra o diagrama simplificado das classes envolvidas na implementação desta política.

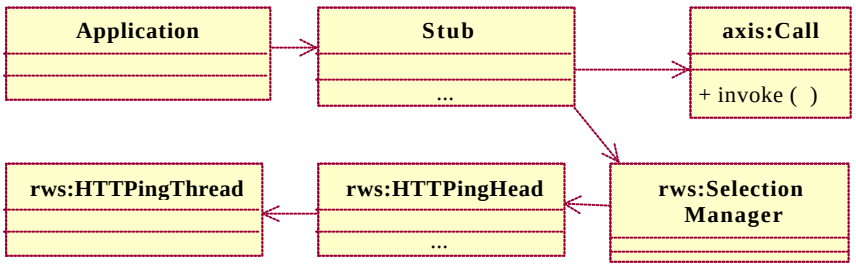

Figura 4.12: Diagrama simplificado das classes envolvidas na implementação da política HTTPing.

Durante a execução, o objeto da classe *stub*, devidamente parametrizado pela aplicação cliente, interage com uma instância da classe *SelectionManager*, que por sua vez aciona uma instância da classe *HTTPingHead*, a qual, de posse da lista de servidores que oferecem uma réplica do serviço, instancia objetos da classe *HTTPingThread* de acordo com a quantidade de servidores existentes para o serviço em questão. Esses objetos enviam uma requisição HTTP HEAD a cada um dos servidores. O primeiro servidor a responder é selecionado. Essa informação é passada pelo objeto da classe *HTTPingHead*, através da instância da *SelectionManager*, até o objeto *stub*, que em seguida envia a chamada da operação ao servidor selecionado. Esse processo é ilustrado na Figura 4.13.

[1] HTTP HEAD – um dos tipos de mensagens suportadas pelo protocolo HTTP. Com essa mensagem, obtém-se as informações de data e hora de atualização de um documento cuja expressão de caminho é passada como parâmetro.

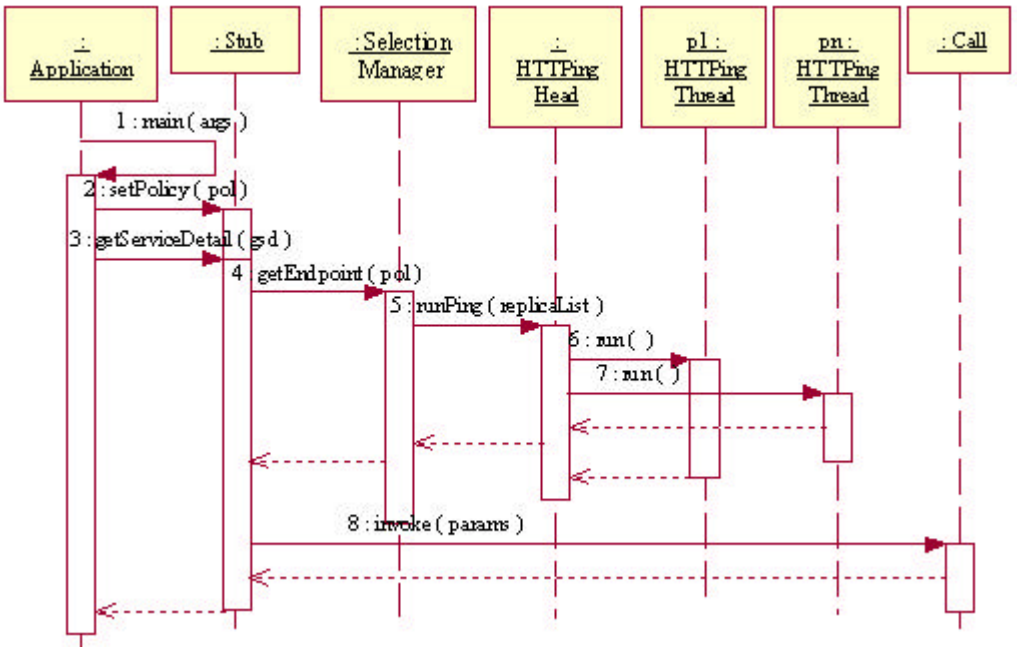

Figura 4.13: Processo de invocação na política HTTPing.

4.3.2.4 Política Melhor Última

Esta política é um das duas políticas do *framework* RWS que, juntamente com a política Melhor Mediana, realizam a seleção de servidores considerando dados históricos de invocações anteriores. Para sua implementação, foram criadas classes adicionais que têm como papel o armazenamento e tratamento dos dados históricos utilizados na escolha dos servidores para o envio das chamadas futuras. A interação com as classes adicionais é realizada pela aplicação cliente no registro dos dados obtidos nas chamadas de operação de um serviço, e pelas classes auxiliares, quando recuperam dados históricos para a seleção do servidor a ser contatado. A Figura 4.14 ilustra o diagrama simplificado das classes envolvidas na implementação das políticas Melhor Última e Melhor Mediana. As classes *HistoryLog* e *HistoryRecord* são uma abstração de um arquivo randômico na linguagem Java, onde são armazenadas as informações referentes aos resultados de invocações anteriores feitas a cada um dos servidores que contêm réplicas do serviço.

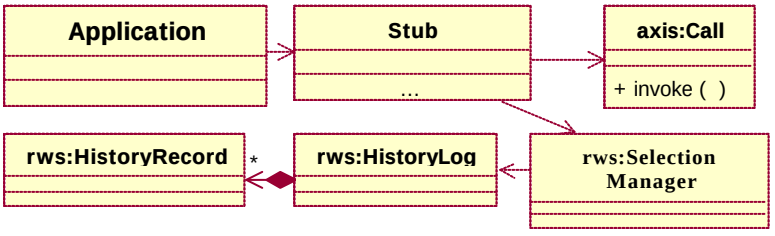

Figura 4.14: Diagrama simplificado das classes envolvidas na implementação das políticas Melhor Última e Melhor Mediana.

O método *getBestLastEndpoint()*, definido na classe *HistoryLog*, retorna o endereço do servidor que teve o melhor desempenho entre as últimas invocações enviadas a cada um dos servidores participantes do conjunto de réplicas. A escolha é baseada na melhor largura de banda obtida, calculada considerando o tempo e o tamanho das respostas recebidas para uma determinada operação disponibilizada pelo serviço.

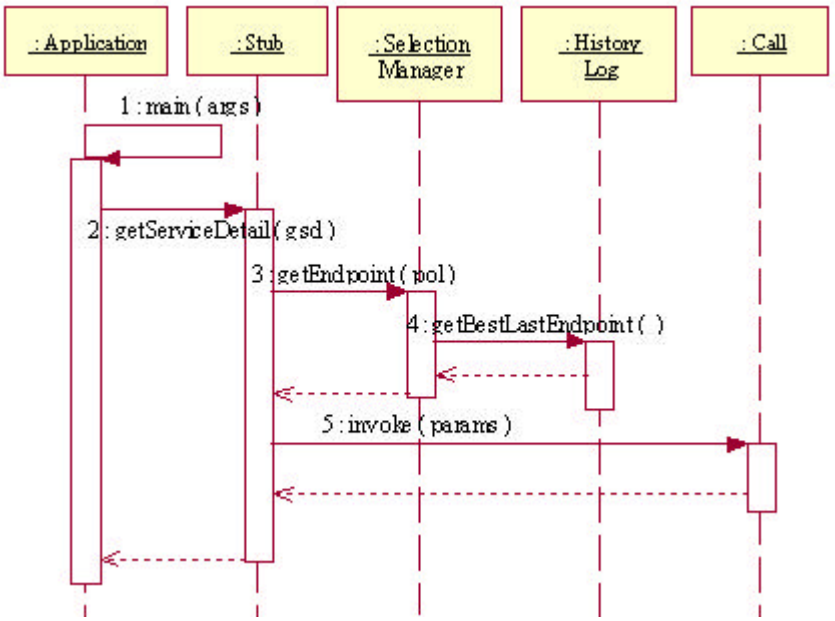

Figura 4.15: Processo de invocação na política Melhor Última.

A aplicação cliente, ao receber a resposta de uma chamada ao serviço realizada com sucesso, faz o registro dos dados necessários para uso estatístico posterior. Ao solicitar uma nova invocação ao serviço, a aplicação cliente aciona o objeto da classe *stub*, o qual aciona um objeto da classe *SelectionManager*, que por sua vez aciona uma instância da classe *HistoryLog*, para finalmente selecionar um servidor de acordo com o critério descrito acima. A Figura 4.15 ilustra a seqüência de eventos envolvida na aplicação da política Melhor Última.

4.3.2.5 Política Melhor Mediana

Esta política é uma generalização da política anterior, e consiste na seleção do servidor com o melhor desempenho mediano, calculado a partir dos dados históricos das k últimas invocações anteriores. O valor de k é um parâmetro fornecido pela aplicação cliente na utilização desta política. No caso de $k = 1$, a política Melhor Mediana se reduz à

política Melhor Última. Esta política é uma adaptação da política estatística proposta por Dykes *et al.* (2000).

O diagrama das classes envolvidas na sua implementação é semelhante ao representado na Figura 4.14. A diferença está na interação com a classe *HistoryLog*, que agora é através do método *getBestMedianEndpoint*(). Este método retorna o endereço do servidor que ofereceu a melhor largura de banda mediana entre as *k* últimas invocações bem sucedidas registradas no histórico. A razão pela escolha da mediana está detalhada na seção que trata a metodologia utilizada na avaliação experimental, descrita no próximo capítulo. A Figura 4.16 mostra a seqüência de eventos envolvida na aplicação da política Melhor Mediana.

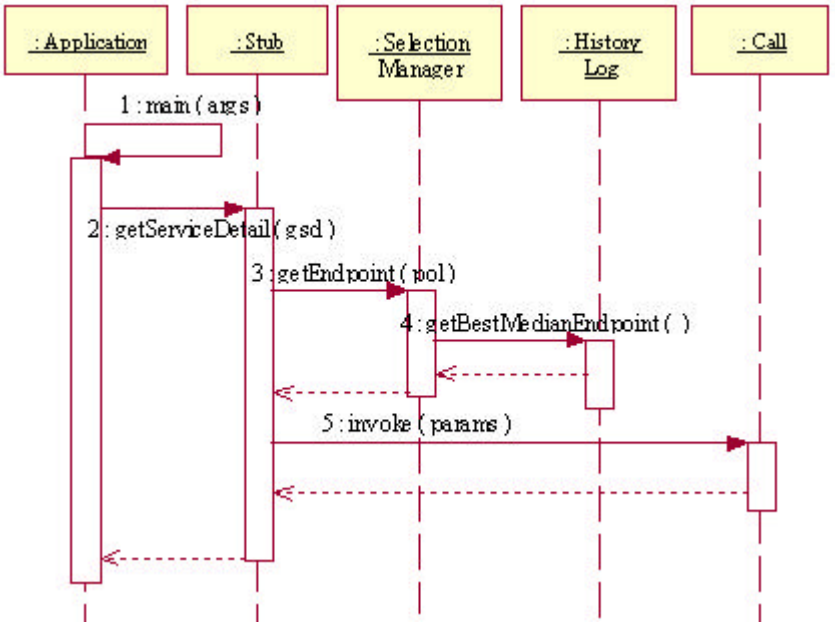

Figura 4.16: Processo de invocação na política Melhor Mediana.

4.4 Sumário

Neste capítulo, apresentamos o *framework* RWS, que é uma extensão do *framework* AXIS para dar suporte ao processo de invocação de serviços Web replicados de forma transparente para as aplicações clientes. Descrevemos como o *framework* implementa o tratamento dos endereços de localização das réplicas, a partir de um documento WSDL, bem como as classes adicionais introduzidas para as diferentes políticas de seleção de servidores consideradas.

O próximo capítulo descreve como as políticas de seleção de servidores incorporadas ao *framework* RWS foram avaliadas experimentalmente utilizando um cenário real da Internet.

Capítulo 5

Avaliação Experimental

Este capítulo descreve a metodologia utilizada para a avaliação experimental do framework RWS, bem como a análise dos resultados obtidos.

No capítulo anterior, apresentamos o *framework* RWS, que incorpora diversas políticas para a invocação transparente de serviços web replicados. Neste capítulo, descrevemos a metodologia utilizada para a condução da avaliação experimental dessas políticas num cenário real da Internet, e discutimos os principais resultados obtidos através de uma análise quantitativa e qualitativa.

5.1 Metodologia

A avaliação consistiu na invocação periódica de uma operação de um serviço Web replicado, a partir de uma aplicação cliente estruturada sobre o *framework* RWS e que fazia uso alternado das cinco políticas de seleção de servidor descritas no capítulo 4. Os dados referentes ao resultado de cada invocação foram armazenados e posteriormente analisados, o que permitiu uma avaliação do desempenho oferecido por cada política sob diferentes cenários. Os experimentos foram realizados no período de 12 a 30 de dezembro de 2003, 24 horas por dia. Para efeito de análise foram considerados apenas os dados coletados nos dias úteis.

5.1.1 Aplicação cliente

A aplicação desenvolvida para os experimentos tinha dois requisitos básicos: efetuar chamadas periódicas a uma operação disponibilizada pelo serviço Web replicado, e registrar as informações resultantes de cada uma invocação (servidor escolhido, tempo de resposta, situação de falha, etc) em dois arquivos. Um arquivo continha os dados históricos utilizados pelas políticas estatísticas, e o outro continha os dados que seriam utilizados na análise dos resultados dos experimentos. A classe principal da aplicação cliente foi denominada *ClientGSD*.

Nos experimentos, as chamadas ao serviço replicado feita pela aplicação cliente deveriam ser executadas dentro de num tempo máximo permitido. Para isso, a aplicação cliente teve que ser executada sob controle externo de outra aplicação, a qual ficaria encarregada de monitorar o tempo de execução das chamadas e, caso o tempo máximo fosse ultrapassado, registrar uma ocorrência no arquivo de log referente à falha por tempo expirado (*time-out*). Para atender esse requisito, implementamos algumas classes adicionais. A classe *ExecClientGSD* teve como papel iniciar a execução da aplicação cliente, dentro de outra máquina virtual Java (JVM), e efetuar o registro das informações no arquivo de log no caso da aplicação cliente não concluir no tempo máximo estipulado. As classes *ExecTimeout* e *EndExec*, por sua vez, tinham como papel o controle do tempo de execução da aplicação cliente. A Figura 5.1 apresenta um diagrama UML representando as três classes utilizadas como parte da aplicação cliente e seus relacionamentos. As classes *TimerTask* e *Timer* fazem parte do pacote *java.util*, e a classe *Runtime*, do pacote *java.lang*.

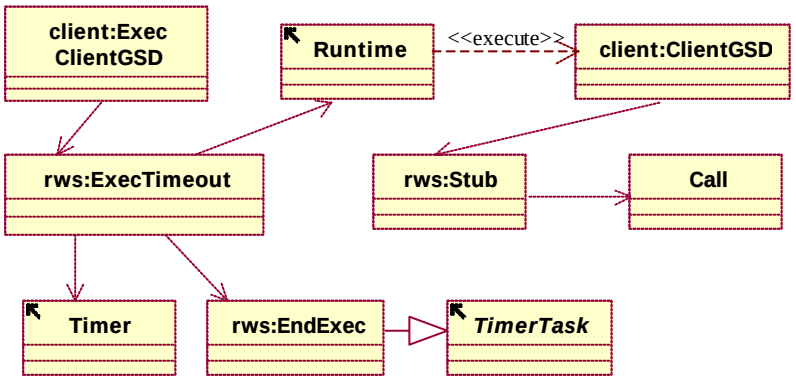

Figura 5.1: Diagrama das classes utilizadas pela aplicação cliente.

A Figura 5.2 ilustra o diagrama de seqüência para a execução da aplicação cliente. O diagrama contempla apenas os passos necessários para iniciar a aplicação cliente, não representando a seqüência completa da chamada de uma operação. Vale ressaltar que a medição dos tempos de resposta é feita dentro da aplicação cliente. A mensagem de número 9, representada no diagrama pela operação *run*, só ocorre no caso da aplicação cliente ultrapassar o tempo máximo definido para a sua execução.

5.1.2 Serviço Web utilizado

Para os experimentos, utilizamos o serviço Web *UDDI Business Registry*, que está disponível em quatro servidores localizados em três continentes, conforme descrito no capítulo 2. Cada réplica do serviço foi invocada através da mesma operação, *GetServiceDetail*, que retorna os detalhes armazenados pelo serviço UDDI sobre um ou mais serviços cujas chaves de identificação são passadas como parâmetro. Como as estruturas retornadas com os detalhes de um serviço têm aproximadamente o mesmo tamanho, variando o número de chaves passadas como parâmetro foi possível obter um controle preciso sobre o tamanho das respostas recebidas pela aplicação cliente. Essa propriedade permitiu uma maior flexibilidade na definição dos tamanhos de resposta utilizados nos experimentos, os quais são descritos na seção 5.1.4.

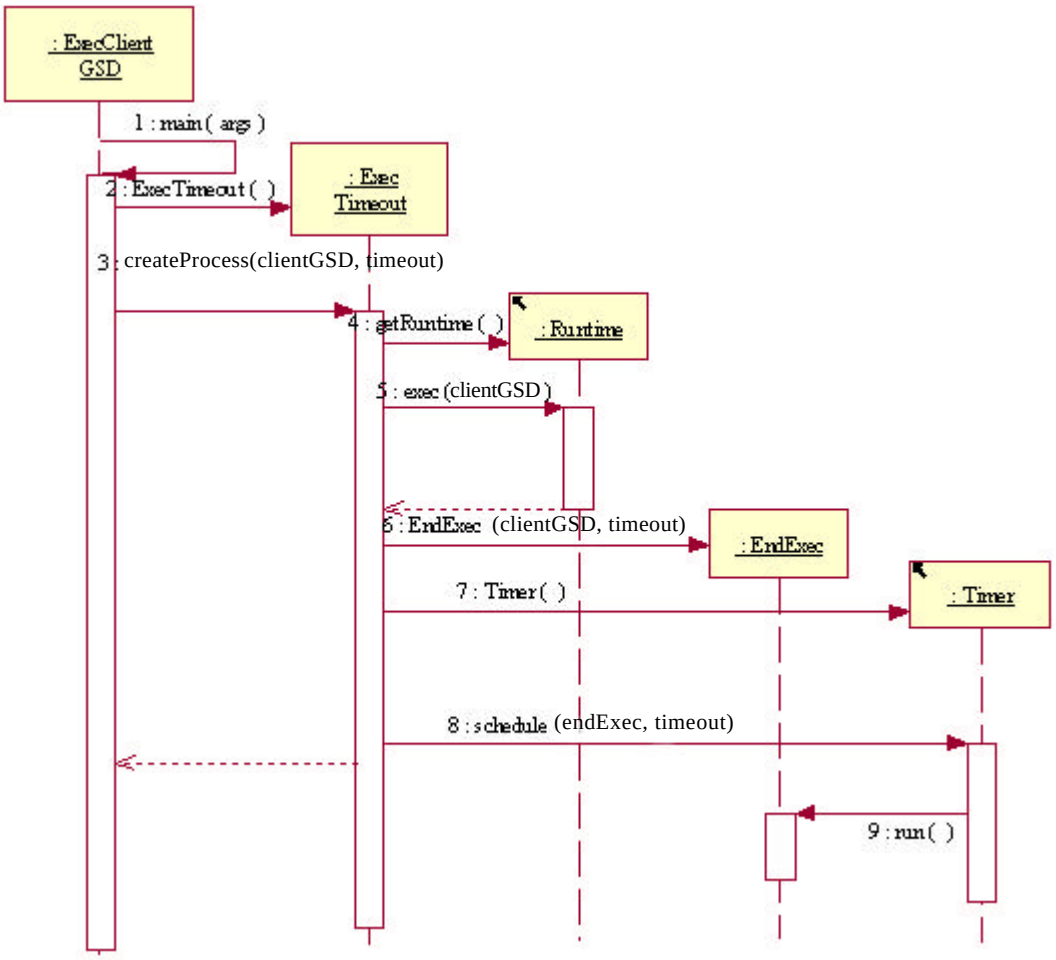

Figura 5.2: Diagrama de seqüência ilustrando o controle de execução da aplicação cliente.

5.1.3 Clientes e servidores

Como clientes, utilizamos duas estações de trabalho que executaram, de forma dedicada, a aplicação descrita na seção 5.1.1. As duas estações estavam localizadas em Fortaleza e conectadas à Internet através de diferentes *backbones* de comunicação.

Uma estação estava instalada no laboratório do Mestrado em Informática Aplicada, da Universidade de Fortaleza (UNIFOR), utilizando uma conexão através da Rede Nacional de Pesquisa – RNP. A outra estava nas dependências do Banco do Nordeste (BNB), utilizando uma conexão via EMBRATEL. Os equipamentos que executaram a aplicação tinham as seguintes características:

- Estação UNIFOR/RNP: conexão à Internet E1 (2Mbps), processador Pentium III 700 MHz, 192 MB de RAM, HD IDE 20 GB, sistema operacional Windows 2000 Professional, servidor proxy Squid;
- Estação BNB/EMBRATEL: conexão à Internet 2xE1 (4Mbps), processador Pentium III 800 MHz, 256 MB, HD SCSI 18 GB, sistema operacional Windows 2000 Server, servidor proxy Microsoft Proxy.

A Figura 5.3 ilustra a localização geográfica dos clientes no Brasil e dos quatro servidores com réplicas do serviço em três continentes.

Os servidores que hospedam as quatro réplicas do serviço UDDI *Business Registry* estão localizados nas seguintes URLs:

- “http://www-3.ibm.com/services/uddi/inquiryapi" (IBM)
- "http://uddi.microsoft.com/inquire.asmx" (MS)
- "http://www.uddi.ne.jp/ubr/inquiryapi" (NTT)
- "http://uddi.sap.com/UDDI/api/inquiry/" (SAP)

5.1.4 Sessões e Ciclos

Para permitir a avaliação dos resultados sob diferentes cenários, as invocações ao serviço replicado foram distribuídas em sessões, que por sua vez, foram divididas em ciclos. Cada sessão compreendeu a realização de cinco ciclos, de acordo com o detalhamento dado a seguir. Dentro de cada ciclo foram aplicadas, seqüencialmente, as cinco políticas incorporadas ao *framework* RWS, na seleção do servidor que receberia uma chamada ao serviço.

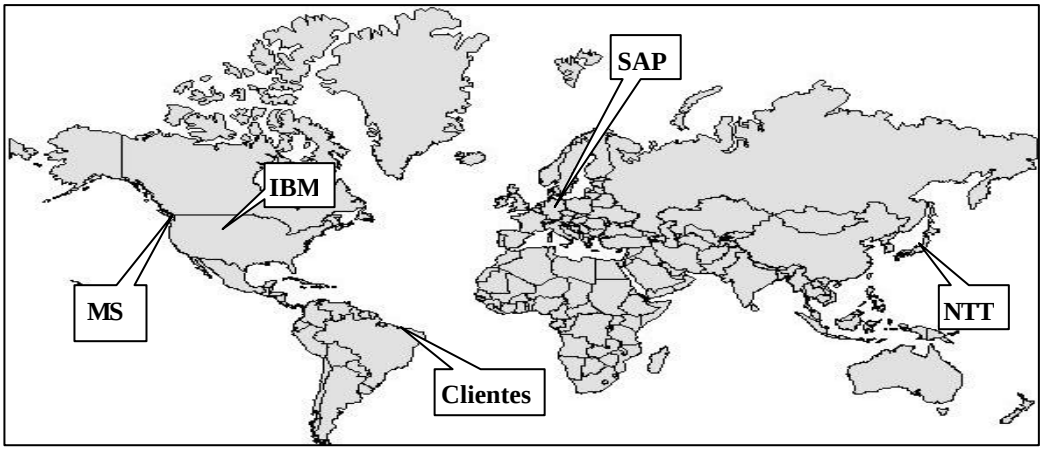

Figura 5.3: Localização geográfica dos clientes e servidores utilizados nos experimentos.

A organização das invocações às réplicas do serviço, em ciclos, permitiu-nos observar o processo de seleção e o efeito no tempo de resposta de acordo com variações no tamanho da resposta e no período do dia de cada invocação. Dentro de cada ciclo estipulamos tempos máximos em que as requisições teriam que ser atendidas. Caso a resposta não fosse recebida no tempo esperado, a chamada à operação seria tratada com a exceção de tempo expirado (*time-out*). Essa necessidade estava associada ao fato de algumas chamadas poderem demorar muito além do tempo médio de resposta, o que poderia inviabilizar a execução de um número mínimo sessões ao longo do dia. Na etapa de calibração do experimento, as chamadas às operações dos serviços foram efetuadas sem qualquer controle de tempo, nos mesmos horários da coleta efetiva dos dados. A partir desse período livre de controle de tempo, foram identificados os tempos máximos, médios, medianos e mínimos de resposta, a partir dos quais foram definidos os tempos máximos que a aplicação cliente poderia aguardar pela resposta, dentro de cada ciclo.

Os parâmetros definidos para cada um dos cinco ciclos de uma sessão estão representados na tabela 5.1. O tempo máximo de espera refere-se ao tempo de uma chamada completa da operação do serviço, ou seja, envolvendo o envio da requisição e o posterior recebimento da resposta pela aplicação cliente. A definição de um tamanho e tempo máximo para a resposta de uma chamada ao serviço foi a alternativa utilizada para que pudéssemos trabalhar sob condições de rede aproximadamente constantes durante a execução de cada ciclo.

Ciclo	Número de chaves requisitadas	Tamanho da resposta (bytes)	Tempo máximo para resposta (seg)
1	1	9K	80
2	10	87K	200
3	20	174K	320
4	30	260K	440
5	40	347K	560

Tabela 5.1: Parâmetros dos cinco ciclos aplicados no experimento.

5.1.5 Políticas de seleção avaliadas

A aplicação cliente fez uso de diferentes políticas de seleção de servidores nas invocações ao serviço replicado, registrando o tempo decorrido entre o envio de cada requisição e o recebimento de suas respostas. Além do tempo de resposta, os seguintes dados também foram coletados pela aplicação cliente: o servidor escolhido, a operação executada, a data e hora de chamada da operação, o número de bytes enviados e recebidos, entre outros.

As políticas de seleção avaliadas são aquelas descritas no Capítulo 4, que resumimos a seguir:

- Randômica - A escolha do servidor é feita de forma aleatória. Em seguida, a chamada da operação é feita apenas ao servidor escolhido.
- Paralela - A chamada da operação é feita de forma concorrente a todos os servidores com réplicas do serviço, com uso de *threads*. A primeira resposta obtida é repassada à aplicação, com outras respostas parciais sendo descartadas.
- HTTPing - Esta política utiliza um mecanismo adicional que envia uma mensagem de sonda a todos os servidores, de forma concorrente. A chamada da operação é efetuada ao servidor que primeiro responder à mensagem sonda.
- Melhor Última – Nesta política, a seleção do servidor leva em conta a invocação mais recentemente efetuada a cada uma das réplicas. A informação é obtida a partir de um arquivo de log mantido pela aplicação cliente e que armazena dados das invocações anteriores. O servidor

selecionado é aquele que apresenta o melhor tempo de resposta na última invocação efetuada para o mesmo ciclo.

- Melhor Mediana – Nesta política, a seleção é feita com base no histórico das *k* últimas invocações efetuadas em ciclos equivalentes, a cada um dos servidores. Aquele que apresentar melhor resultado mediano entre as últimas invocações consideradas é selecionado. Nos experimentos, utilizamos o valor de $k = 6$.

5.1.6 Métrica de desempenho

A métrica utilizada para avaliação das políticas no experimento foi o tempo de resposta percebido pela aplicação cliente. A motivação para adotarmos essa medida foi com o objetivo de refletir todo o tempo experimentado pelo cliente em qualquer uma das políticas de invocações, visto que esse é o tempo de interesse na utilização do serviço. O tempo de resposta utilizado compreendeu o instante imediatamente anterior à chamada da operação até o recebimento dos dados pela aplicação cliente. Nas políticas baseadas em sonda ou dados estatísticos, o *overhead* para escolha do servidor também foi considerado dentro do tempo total observado em cada invocação. Essa métrica também foi utilizada em Dykes *et al*. (2000), sendo que naquele trabalho foi necessária uma normalização dos tempos de resposta obtidos em virtude dos objetos requisitados serem de tamanhos significantemente diferentes.

Os tempos de resposta foram analisados considerando a mediana como medida de tendência central. A mediana foi utilizada tanto no cálculo dos dados estatísticos, usados pela política Melhor Mediana, quanto na própria avaliação dos tempos de resposta observados para cada uma das cinco políticas. A decisão pela mediana foi decorrente dos picos nos tempos de resposta observados durante todo o experimento.

Uma análise dos dados foi realizada com a aplicação do método estatístico Escores Z (Martins, 2001), para a identificação dos pontos de alta variabilidade (*outliers*), dentro de cada ciclo. Os valores de escores utilizados como limites ficaram no intervalo de +3 a –3 para o tratamento dos dados. Na avaliação, os tempos de resposta com escores acima e abaixo do intervalo estabelecido estariam de alguma forma fora do conjunto típico das observações colhidas na amostra de todos os ciclos/políticas ao longo do período experimental. É importante ressaltar que apenas tempos com escores acima do intervalo

estipulado foram evidenciados. Concluímos, assim, que a média da amostra, pela sua própria natureza de ser sensível aos extremos, não refletia o comportamento típico da maioria dos tempos de resposta obtidos. A mediana evidenciou-se como a medida mais adequada para avaliação, uma vez que foi pouco afetada pelos altos tempos de resposta considerados atípicos, a partir do conjunto de dados analisados.

5.2 Análise de Resultados

Nesta seção, apresentamos uma análise quantitativa e qualitativa dos resultados obtidos a partir da avaliação experimental das políticas de invocação implementadas no *framework* RWS.

5.2.1 Análise quantitativa

Para a análise quantitativa, identificamos os períodos de maior e menor tráfego nos dois clientes, e avaliamos o comportamento das políticas separadamente em cada período. Consideramos tanto os valores absolutos do desempenho obtido em cada política quanto o ganho relativo das políticas de melhor desempenho em relação à política Randômica.

5.2.1.1 Números dos experimentos

Os números obtidos mostram um alto grau de disponibilidade na interação com os servidores durante toda a avaliação, em ambos os clientes. A Tabela 5.2 (a) apresenta uma síntese das sessões realizadas no cliente Unifor. Das 38.511 invocações encaminhadas aos servidores, houve apenas 312 com erros. Isso representa 0,81% do total de invocações, distribuindo-se em 0,43% em *time-outs* e 0,38% em outros erros. A Tabela 5.2 (b) mostra o resumo das sessões no cliente BNB, onde ocorreram apenas 148 ocorrências de falhas, de um total de 50.985 invocações. Os percentuais de erros ficaram em níveis ainda mais baixos que os obtidos no cliente Unifor. Foram 0,13% em *time-outs* e 0,16% em outras falhas, totalizando 0,29%.

Número de Sessões	Chamadas ao Serviço		
	Total	***Time-outs***	**Outras falhas**
861	38.511	165	147

(a)

Número de Sessões	Chamadas ao Serviço		
	Total	***Time-outs***	**Outras falhas**
1.133	50.985	65	83

(b)

Tabela 5.2: Números das sessões realizadas no cliente Unifor (a) e cliente BNB (b).

Do conjunto de sessões realizadas no cliente Unifor, a de maior duração levou cerca de 62 minutos, enquanto a de menor duração levou em torno de 7 minutos. As sessões de maior duração concentraram-se, diariamente, no horário das 15h às 17h. Já no cliente BNB, a de maior duração foi de 33 minutos e a menor de 8 minutos. O período em torno das 13h concentrou as sessões de maior duração.

A Tabela 5.3 nos dá uma visão do desdobramento desses números nos cinco ciclos realizados em cada sessão, no cliente Unifor.

Ciclos		Chamadas de Operação do Serviço		
Número	**Duração Total (seg)**	**Total**	***Time-out***	**Outras falhas**
1	48.946	7.740	64	20
2	84.428	7.716	27	22
3	128.520	7.701	25	24
4	165.029	7.684	23	36
5	205.660	7.670	26	45
Totais	632.583	38.511	165	147

Tabela 5.3: Números dos ciclos realizados no cliente Unifor.

A Tabela 5.4 mostra os números dos cinco ciclos realizados em cada sessão no cliente BNB.

Ciclos		Chamadas ao Serviço		
Número	**Duração Total (seg)**	**Total**	***Time-out***	**Outras falhas**
1	44.623	10.197	7	7
2	90.652	10.197	8	11
3	144.344	10.197	13	11
4	201.847	10.197	19	21
5	255836	10.197	18	33
Totais	737.302	50.985	65	83

Tabela 5.4: Ciclos realizados no cliente BNB.

5.2.1.2 Efeitos dos períodos do dia

No cliente Unifor, o tráfego observado aumenta significativamente pela manhã e só volta a diminuir no período da noite. No cliente BNB, embora haja um aumento visível no tráfego durante o dia, observamos que no período noturno há ainda um tráfego intenso.

Traçamos, inicialmente, um gráfico do desempenho das políticas em cada cliente, por período do dia, com o objetivo de identificar em que momentos ocorriam variações significativas nos tempos de resposta. As Figuras 5.4 e 5.5 ilustram os resultados obtidos, no dois clientes, considerando a mediana dos tempos de resposta ao longo das 24 horas do dia. Os gráficos são relativos aos dados obtidos no ciclo 5, onde foram obtidos os maiores tempos de resposta nas chamadas ao serviço. Na identificação dos períodos de maior e menor tráfego, levamos em conta apenas as políticas de melhor desempenho.

No cliente Unifor, observamos que há um aumento visível nos tempos de resposta após as 7h, e uma redução para os mesmo níveis anteriores somente no horário próximo às 23h. Observamos ainda uma alta variabilidade nos tempos de respostas, mesmo para as políticas de melhor desempenho. Às 8h, por exemplo, algumas políticas já apresentam tempos de resposta até 6 segundos maiores que os tempos obtidos às 7h, cujo patamar só é novamente alcançado por volta das 18:30h. Os períodos de maiores picos estão concentrados no horário próximo às 15h. No cliente BNB, considerando apenas as políticas de melhor desempenho, observamos que o intervalo com os maiores tempos de resposta está no período entre 8h e 17h, aproximadamente. Os períodos de picos, para essas políticas, concentram-se em torno das 11h e 15h.

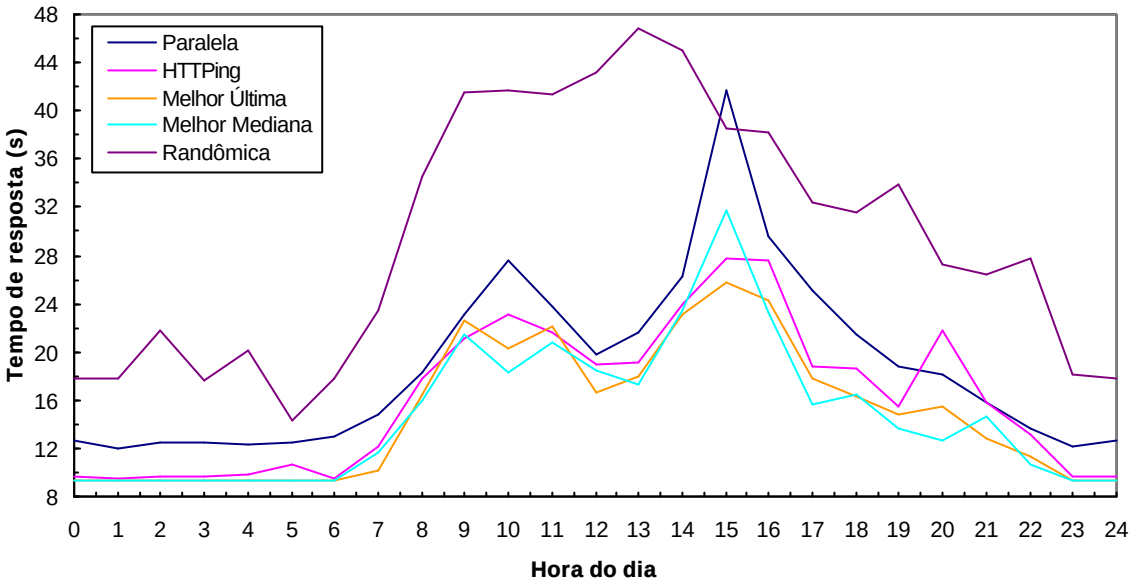

Figura 5.4: – Efeitos do período do dia no ciclo 5 (cliente Unifor).

De modo geral, observamos que os tempos de resposta obtidos no cliente BNB estão em patamares superiores ao do cliente Unifor. No entanto, o cliente BNB apresenta uma maior escalabilidade no tratamento das respostas durante o período de maior tráfego. Isto pode ser observado pelo fato de que os tempos de respostas obtidos nesse cliente apresentam uma menor variação, quando comparados aos tempos observados no cliente Unifor. No cliente BNB, a amplitude entre o ponto mais baixo e mais alto da curva de melhor desempenho é de aproximadamente 5 segundos. Já no cliente Unifor, essa amplitude fica em torno de 16 segundos.

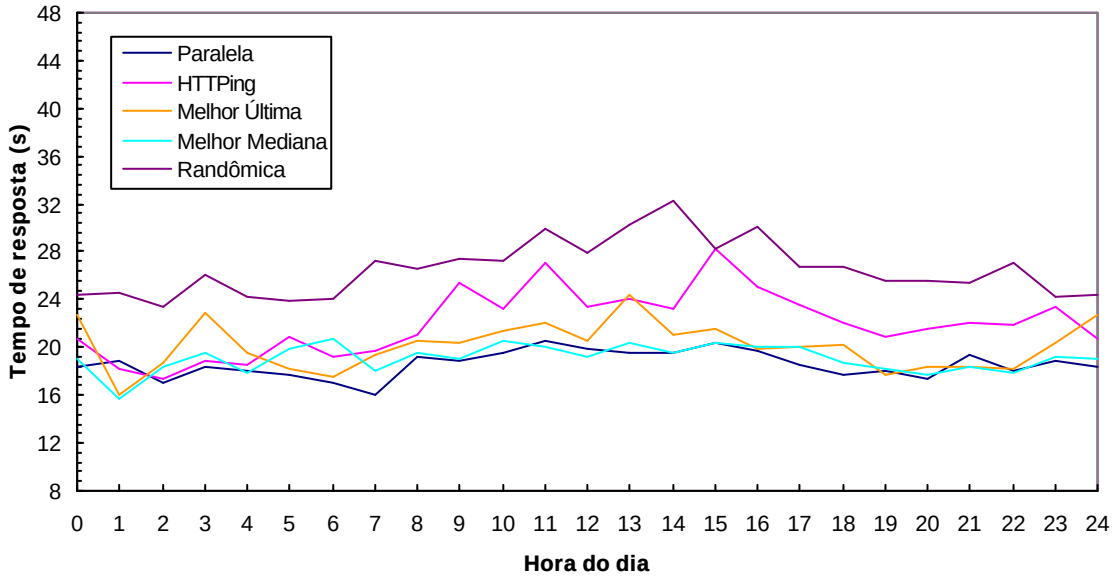

Figura 5.5: – Efeitos do período do dia no ciclo 5 (cliente BNB).

O coeficiente de variação (CV) de Pearson (Martins, 2001) permite-nos comparar, relativamente, a variação observada nos conjuntos dos tempos de resposta observados. A Tabela 5.5 mostra os coeficientes de variação para cada uma das cinco políticas avaliadas. De acordo com as regras empíricas de interpretação do coeficiente de variação (Martins, 2001), os tempos de resposta obtidos no cliente Unifor apresentam elevada dispersão (CV > 30%), enquanto no BNB os tempos apresentam média dispersão (15% < CV < 30%).

Política	**Cliente**	
	Unifor	**BNB**
Paralela	62%	18%
HTTPing	57%	25%
Melhor Última	65%	27%
Melhor Mediana	60%	26%
Randômica	61%	29%

Tabela 5.5: Coeficientes de variação dos tempos de respostas no ciclo 5.

5.2.1.3 Desempenho dos servidores

Para a avaliação do desempenho das políticas em ambos os clientes, primeiramente, traçamos o desempenho dos servidores individualmente, o que pode ser observado nas Figuras 5.6 e 5.7.

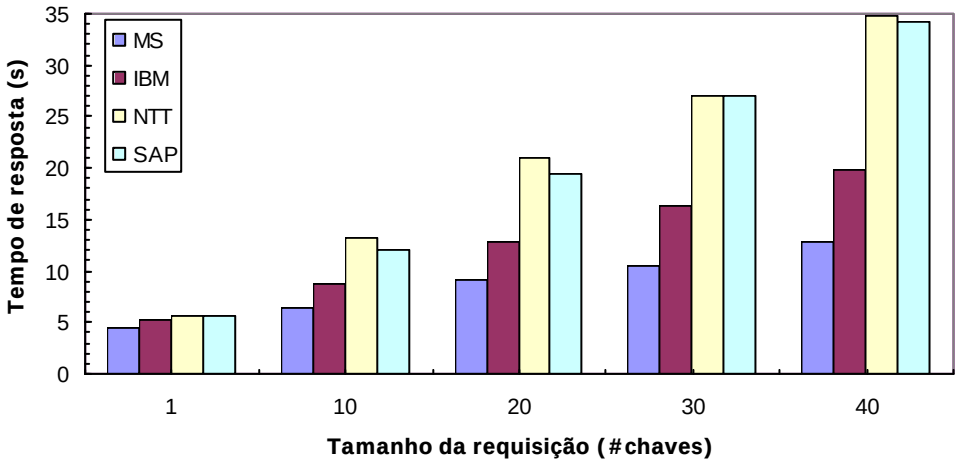

Figura 5.6: Tempos medianos de resposta dos servidores (cliente Unifor).

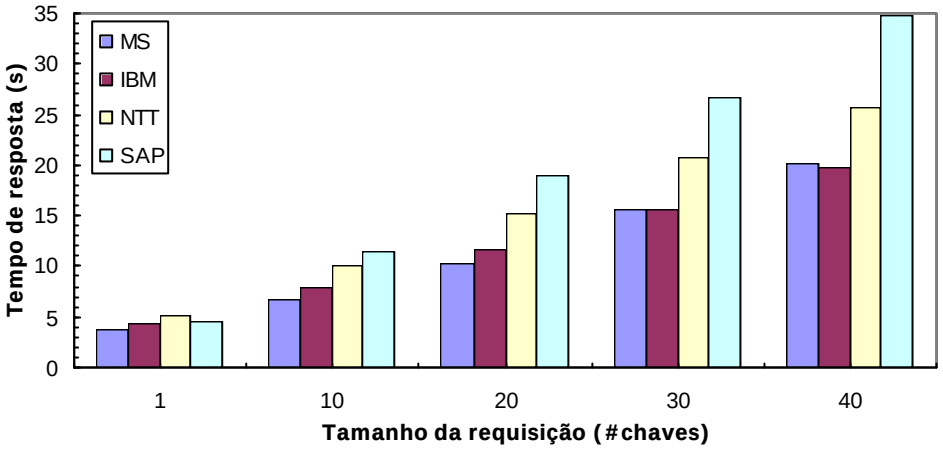

Figura 5.7: Tempos medianos de resposta dos servidores (cliente BNB).

A avaliação do desempenho dos servidores compreendeu o exame dos tempos medianos de resposta obtidos na invocação do serviço, em cada um dos cinco ciclos, onde cada ciclo corresponde ao tamanho da requisição enviada ao servidor. No cliente Unifor, os servidores NTT e SAP apresentam os maiores tempos medianos de resposta, ambos bem próximos. No cliente BNB, temos os servidores Microsoft e IBM próximos, enquanto NTT e SAP se distanciam à medida que o tamanho da requisição cresce. De forma geral, uma classificação de desempenho dos servidores, nos dois clientes, ficaria na seguinte ordem: MS, IBM, NTT e SAP. Os tempos de resposta dos servidores MS e IBM estão em menores níveis em todos os ciclos. Os servidores NTT e SAP têm os maiores tempos de resposta.

5.2.1.4 Desempenho das políticas por período

Os gráficos das Figuras 5.4 e 5.5 mostraram o desempenho das políticas ao longo do dia. Observamos que as curvas das políticas de melhor desempenho cruzam-se em certos períodos, não permitindo uma visão clara de qual delas oferece o melhor desempenho global. As Figuras 58 a 5.11 comparam as políticas, nos dois clientes, dando uma visão global do desempenho de cada uma. Os resultados das políticas estão agrupados dentro de dois períodos, que denominamos de comercial (maior utilização) e não-comercial (menor utilização), identificados para cada cliente conforme discutido na seção 5.2.1.2.

As Figuras 5.8 e 5.9 ilustram a distribuição do desempenho das políticas nos períodos comercial e não-comercial, respectivamente para o cliente Unifor. Nesse cliente, o desempenho das políticas no período não-comercial apresenta um ganho em torno de 50% em relação ao período comercial.

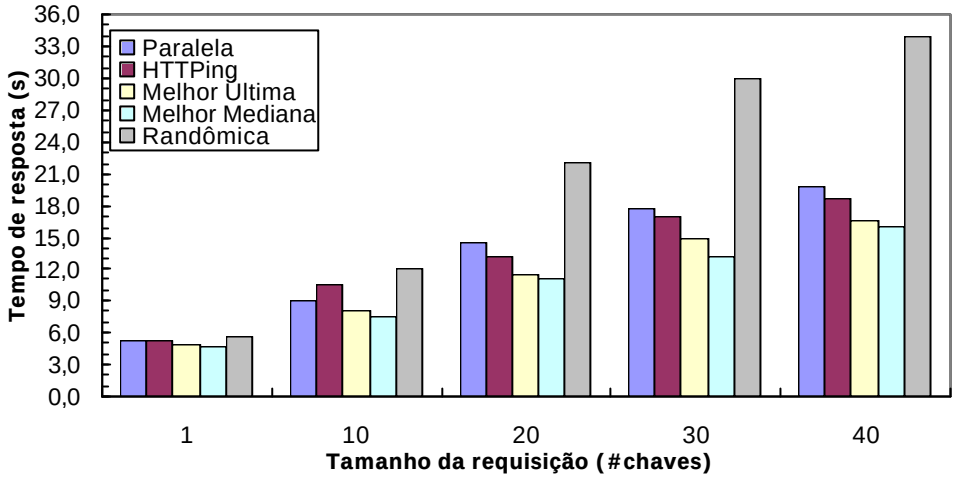

Figura 5.8: Tempo mediano de resposta das políticas no período comercial (cliente Unifor).

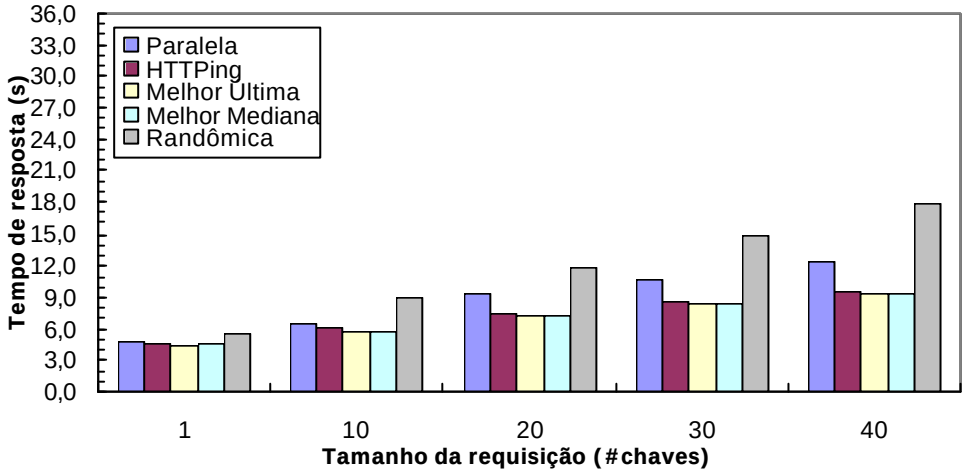

Figura 5.9: Tempo mediano de resposta das políticas no período não-comercial (cliente Unifor).

As Figuras 5.10 e 5.11, por sua vez, mostram a distribuição do desempenho das políticas, nos dois períodos, para o cliente BNB. Nesse cliente, observamos que os tempos medianos de resposta nos períodos comercial e não-comercial estão bem próximos, o que está em conformidade com os moderados coeficientes de variação apresentados para esse cliente na tabela 5.4.

Os tempos de resposta do cliente BNB, mesmo no período não comercial, permanecem em níveis maiores do que os tempos de resposta no cliente Unifor, no mesmo período. Uma diferença a considerar é que o cliente BNB utiliza uma conexão à Internet de natureza comercial, enquanto o cliente Unifor é atendido por uma conexão à Internet de natureza acadêmica.

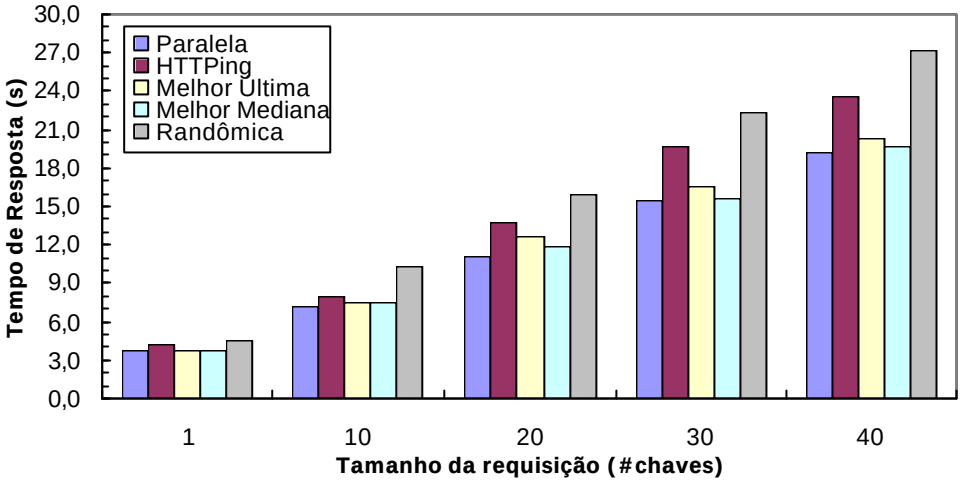

Figura 5.10: Tempo mediano de resposta das políticas no período comercial. (cliente BNB).

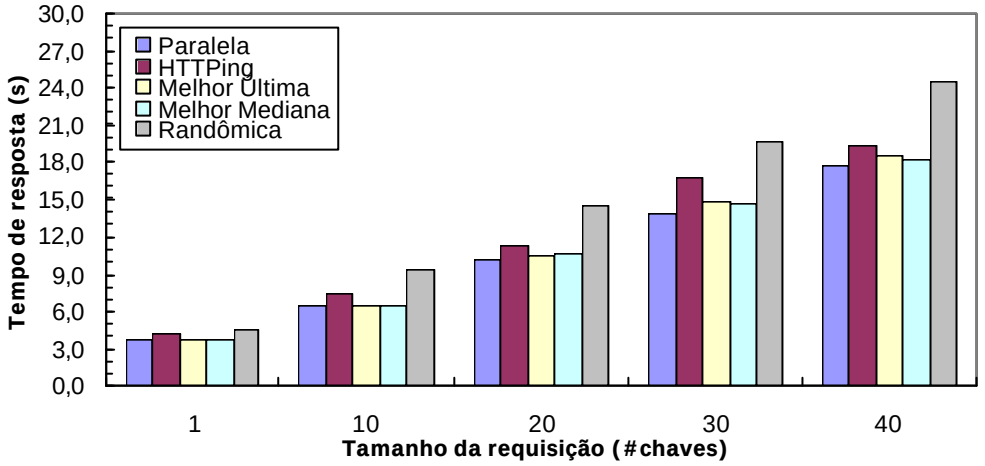

Figura 5.11: Tempo mediano de resposta das políticas no período não-comercial (cliente BNB).

Todas as cinco políticas, nos períodos comercial e não-comercial, apresentam um padrão de desempenho similar no que se refere ao crescimento dos tempos de respostas à medida que os tamanhos das requisições e, conseqüentemente, das respostas, aumentam. Uma característica a se ressaltar é que as três políticas de melhor desempenho, em ambos os clientes, obtêm tempos de resposta bem próximos no período não-comercial.

Para melhor visualizar as diferenças de desempenho entre políticas, traçamos os gráficos de ganho relativo entre elas, em termos do tempo mediano de resposta de cada uma, utilizando como referência a política Randômica, que apresentou o pior desempenho nos dois clientes em todos os ciclos.

As Figuras 5.12 e 5.13 ilustram o ganho relativo das demais políticas em relação à política Randômica, no cliente Unifor, nos períodos comercial e não-comercial, respectivamente. Nesse cliente, observamos claramente que no período comercial o ganho relativo entre as políticas é mais acentuado que no período não-comercial, com o melhor caso acontecendo no ciclo 3 (requisição de 30 chaves), onde a política Melhor Mediana apresenta tempos de resposta até 2,3 vezes mais rápidos que a política Randômica.

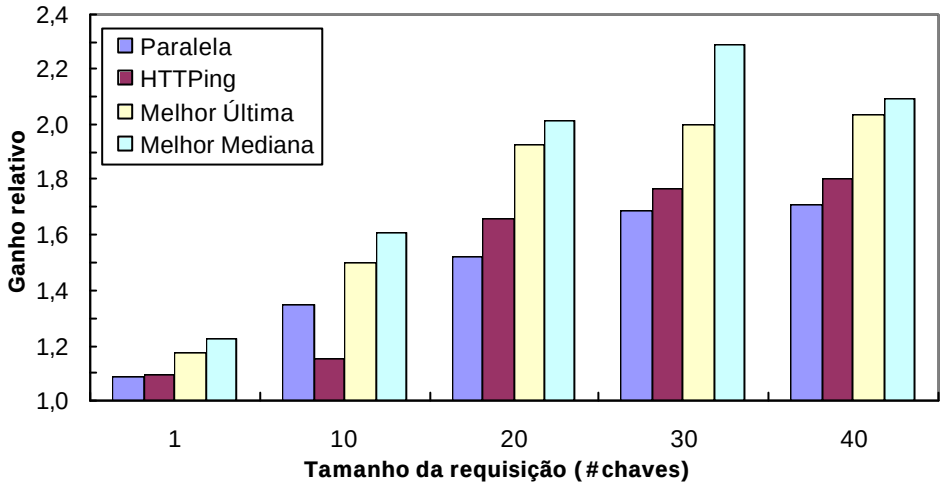

Figura 5.12: Ganho relativo das políticas em relação à política Randômica no período comercial (cliente Unifor).

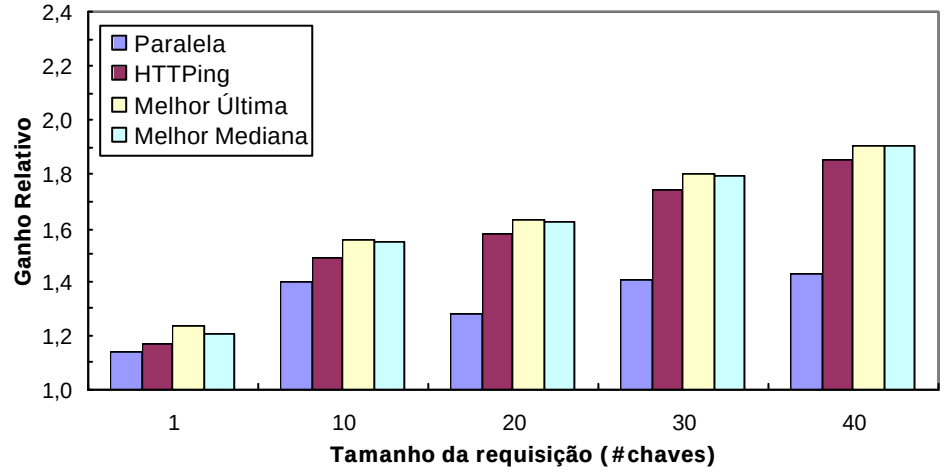

Figura 5.13: Ganho relativo das políticas em relação à política Randômica no período não-comercial (cliente Unifor).

As Figuras 5.14 e 5.15 ilustram o ganho relativo das políticas em relação à política Randômica no cliente BNB, nos dois períodos. Nesse cliente, a diferença do ganho relativo entre as melhores políticas é menor que no cliente Unifor. O destaque, mais uma vez, é a

política Paralela, que obtém um ganho global superior em relação às demais políticas. Esse ganho é mais acentuado nos ciclos 2 e 3 do período comercial, e nos ciclos 3, 4 e 5 do período não-comercial. No período não-comercial, no ciclo 2, há um empate em termos de ganho relativo entre a política Paralela e as duas estatísticas.

Considerando a análise no período de maior utilização da rede nos dois clientes, observamos que a política Paralela apresenta resultados de desempenho opostos de um cliente para outro. Ela é a melhor política no cliente BNB, mas ocupa a posição de segunda pior política no cliente Unifor, atrás apenas da política Randômica.

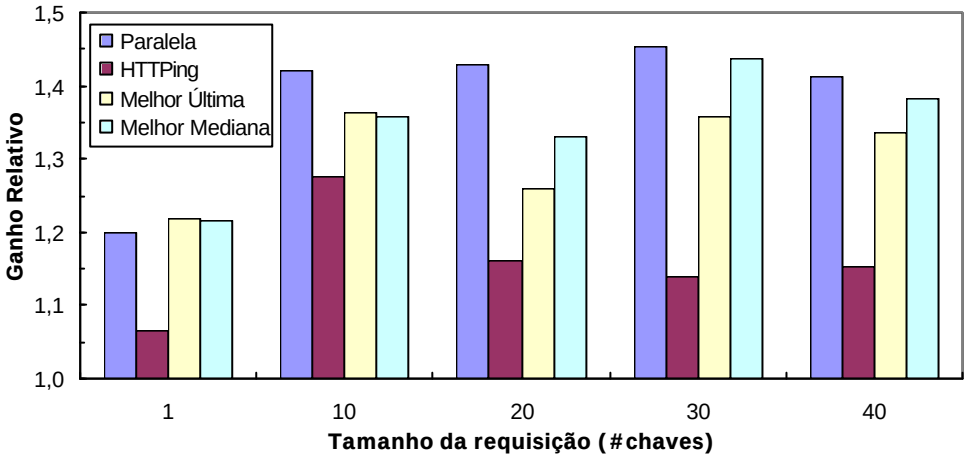

Figura 5.14: Ganho relativo das políticas em relação à política Randômica no período comercial (cliente BNB).

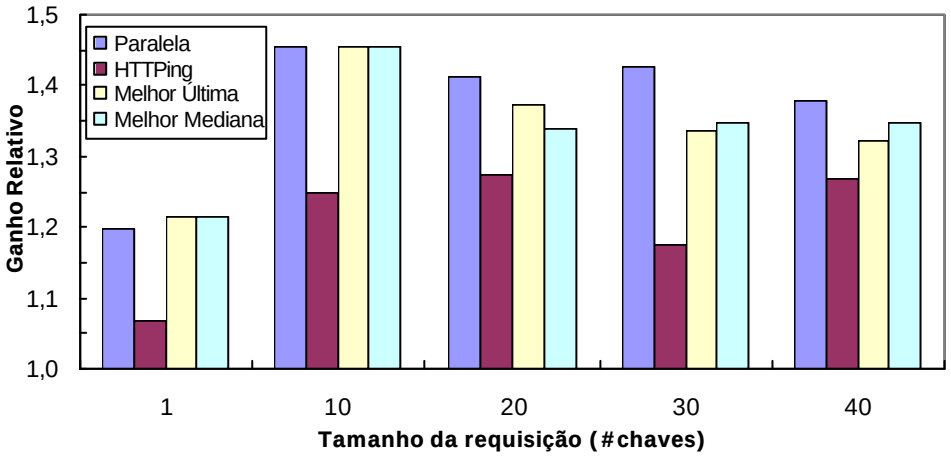

Figura 5.15: Ganho relativo das políticas em relação à política Randômica no período não-comercial (cliente BNB).

A política HTTPing apresenta desempenho acima das políticas estatísticas, em ambos os clientes. Porém, quando relacionada com a Paralela, o seu desempenho é diferenciado de um cliente para outro, sendo inferior no cliente BNB e superior no cliente Unifor.

A política Randômica é a que apresenta o pior desempenho independentemente do cliente, do ciclo e do período. Mostrando-se, portanto, como uma política não adequada para a invocação de serviços Web geograficamente replicados. A escolha randômica do servidor, como era esperado, ora seleciona servidores que apresentam baixos tempos de resposta, ora seleciona aqueles com alta latência nas respostas, o que prejudica sobremaneira o seu desempenho global. O desempenho observado para a política Randômica corresponde ao resultado obtido por Dykes *et al*. (2000), ao avaliar o acesso a imagens e documentos HTML, através de uma política similar.

As políticas estatísticas apresentaram, na maioria dos casos, desempenhos muito próximos. Como ambas as políticas são baseadas em respostas anteriores, levando em conta os melhores tempos de resposta, elas em geral selecionavam os servidores com o melhor desempenho. Na maioria das vezes, essas políticas faziam a seleção entre os dois servidores que historicamente tinham melhor desempenho, ou seja, MS e IBM. O percentual de escolha de cada servidor por cada uma das políticas avaliadas é analisado na próxima seção.

5.2.2 Análise qualitativa

Nesta seção, analisamos qualitativamente o desempenho individual de cada política nos dois clientes. Para isso, observamos o percentual de seleção dos servidores obtidos por cada uma delas, e correlacionamos esses resultados com o desempenho dos servidores apresentados anteriormente.

Mostramos, ainda, a distribuição acumulada dos tempos de resposta obtidos com cada política, em ambos os clientes.

5.2.2.1 Política Paralela

A Figura 5.16 apresenta os percentuais de seleção dos quatro servidores efetuados pela política Paralela no cliente Unifor, nos cinco ciclos. O maior percentual de escolha

está associado ao servidor MS, seguido, em um distante segundo lugar, pelo servidor IBM. Um aspecto interessante na aplicação dessa política, no cliente Unifor, é o nível de seleção do servidor SAP no ciclo 1, que fica em torno de 2,1%. O servidor NTT foi selecionado poucas vezes, alcançando o nível mais alto, de apenas 0,8%, no ciclo 1. Os níveis de seleção do servidor SAP e NTT não guardam relação com o desempenho individual observado para esses dois servidores.

A política Paralela, no cliente Unifor, mostrou-se como a segunda política de mais baixo desempenho, ficando apenas à frente da Randômica. Embora tenha selecionado, na maioria das vezes, o servidor MS, que possui o melhor desempenho global, o efeito da concorrência das chamadas enviadas aos servidores em paralelo ocasionava uma carga significativa no lado do cliente, elevando assim os tempos de resposta observados.

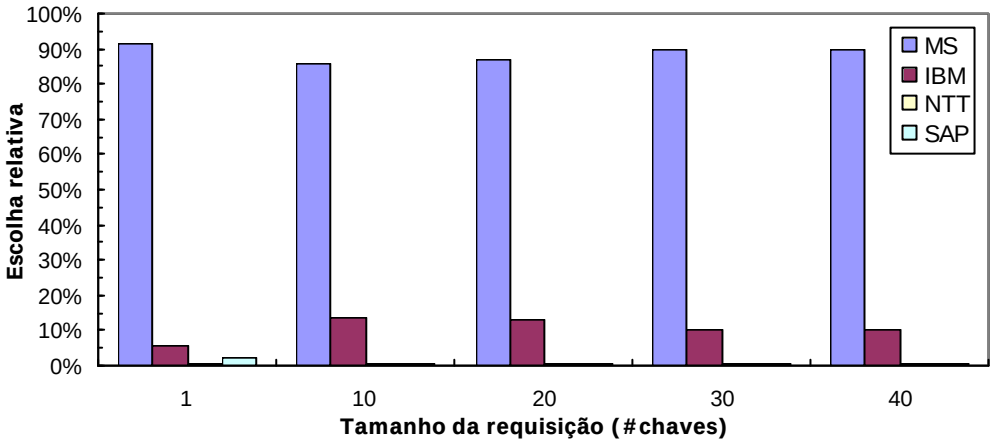

Figura 5.16: Seleção de servidores na política Paralela (cliente Unifor).

A Figura 5.17 ilustra o percentual de escolha de servidores da política Paralela no cliente BNB, a qual teve o melhor desempenho entre as cinco políticas avaliadas nesse cliente, em todos os ciclos. No ciclo 1, o servidor MS foi selecionado em 99,6% das invocações. Esse padrão vai sendo modificado ciclo a ciclo, observando-se, no ciclo 5, que os níveis de seleção distribuem-se, principalmente, entre dois servidores: IBM, com 49,6%, e MS com 42,0%. O servidor NTT foi selecionado em apenas 8,4% das invocações. O servidor SAP não foi selecionado nenhuma das vezes.

O percentual de escolha obtido pela política Paralela no cliente BNB reflete mais proximamente as diferenças de desempenho observadas entre os servidores.

No cliente BNB, a capacidade da conexão em lidar com uma carga maior em nível de rede (*througput*) faz da política Paralela a de melhor desempenho. Isto porque a concorrência no recebimento das respostas dos servidores é bem suportada nesse cliente.

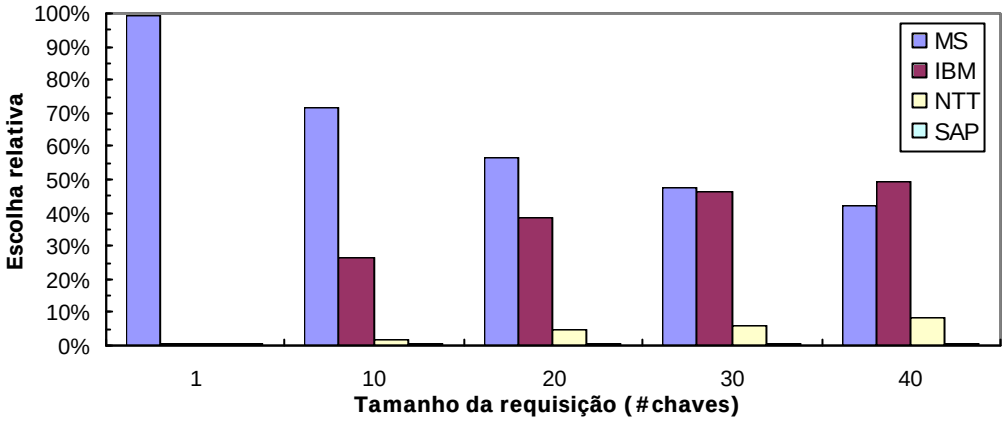

Figura 5.17: Seleção de servidores na política Paralela (cliente BNB).

5.2.2.2 Política HTTPing

A política HTTPing apresentou comportamento diferenciado no processo de seleção de servidores nos dois clientes. Entre as cinco políticas avaliadas, ela ocupou a terceira e quarta posições de desempenho nos clientes Unifor e BNB, respectivamente. Esses resultados estão relacionados com o fato de que nem sempre o servidor selecionado pela política HTTPing oferecia o melhor tempo de resposta. Isso ocorreu com maior freqüência no cliente Unifor.

Da forma aplicada no presente trabalho, a seleção por sonda teve resultados diferentes daqueles obtidos em Dykes *et al*. (2000), no que se refere à seleção do melhor servidor. Ressaltamos que aqui foi utilizado um método de sondagem diferente. Naquele estudo, a seleção do servidor por sonda constituiu-se na medição do tempo necessário ao estabelecimento de uma conexão TCP, avaliando a carga na rede, sem levar em conta nenhum aspecto de carga em nível de servidor. No presente trabalho, optamos pelo envio da sonda na forma de um pacote HTTP HEAD pelas razões técnicas já mencionadas no Capítulo 4. Além disso, diferentemente do trabalho de Dykes *et al*. (2000), com o uso da política HTTPing objetivamos considerar a carga dos servidores até o nível do serviço HTTP, no instante antecedente ao envio da chamada ao serviço.

No processo de seleção através da política HTTPing, observamos que o servidor que geralmente tinha o melhor tempo de resposta da operação invocada, nem sempre respondia à sonda em menor tempo. Contrariamente, o servidor que respondia mais rapidamente à sonda, nem sempre era o mais rápido na resposta à chamada da operação, levando, às vezes, a expirar o tempo máximo de resposta. Considerando que não temos controle sobre as variações externas aos clientes, o primeiro fato poderia estar relacionado com a carga da rede e/ou carga do servidor no momento da chamada ao serviço. Quanto ao segundo, observamos que a questão está mais relacionada com a capacidade do servidor em processar uma chamada de operação do serviço Web.

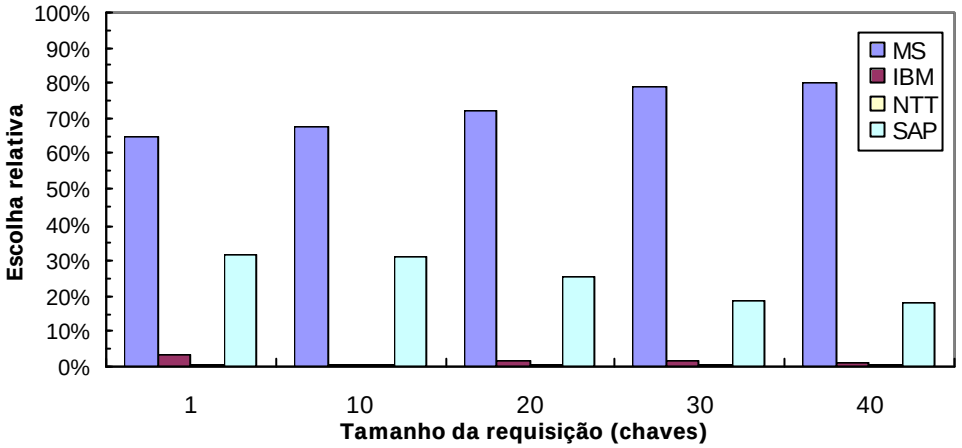

Figura 5.18: Seleção de servidores na política HTTPing (cliente Unifor).

A Figura 5.18 apresenta a seleção de servidores efetuada pela política HTTPing no cliente Unifor. Podemos observar que o percentual de escolha de alguns servidores está em ordem oposta aos seus desempenhos individuais apresentados na seção 5.2.3. Por exemplo, muito embora o servidor SAP tenha sido escolhido 18,9% e 18,3% das vezes, nos ciclos 4 e 5, respectivamente, essa seleção não significou receber a resposta em menor tempo, uma vez que o servidor SAP apresentou o pior desempenho individual.

O menor tempo de resposta do servidor SAP, após a seleção por HTTPing, é ainda bem superior ao menor tempo obtido dos demais servidores selecionados em outras políticas, dentro do mesmo ciclo e período do dia, mesmo descontando o custo do tempo de envio da sonda.

No cliente BNB, o padrão de seleção foi mais compatível com o desempenho dos servidores. Os dois servidores que apresentaram historicamente o melhor desempenho, MS e IBM, foram os que tiveram o maior percentual de seleção. A Figura 5.19 ilustra esses resultados.

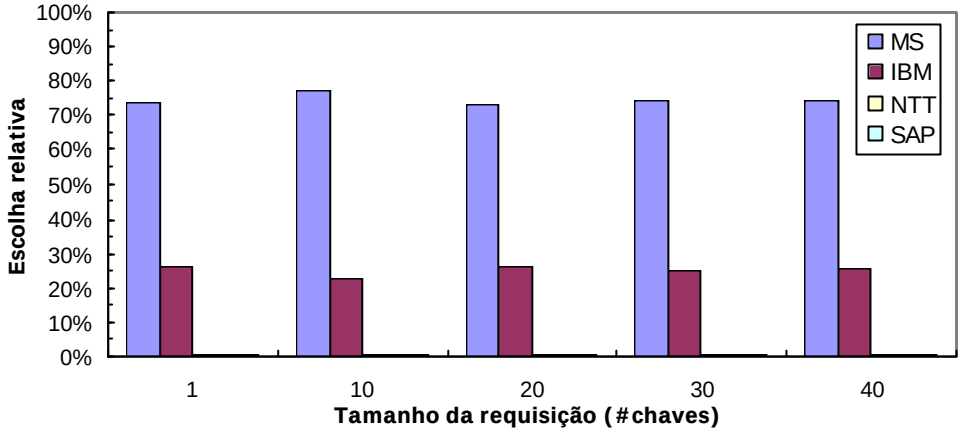

Figura 5.19: Seleção de servidores na política HTTPing (cliente BNB).

5.2.2.3 Política Melhor Última

No cliente Unifor, as políticas estatísticas mostraram-se com desempenho melhor que as demais políticas. O que podemos observar através do gráfico da Figura 5.20 é que o resultado da política Melhor Última guarda estreita semelhança com o desempenho da política Paralela nesse cliente. Outra vez, o servidor MS é selecionado o maior número de vezes, contribuindo para que o tempo mediano observado entre todas as chamadas de operação, dentro de cada um dos cinco ciclos, ficasse em níveis mais baixos. Com o servidor MS tendo sido escolhido com percentuais igualmente elevados nas políticas Melhor Última e Paralela, fica a questão do que estaria causando a diferença de desempenho entre as duas políticas nesse cliente. A resposta está no fato de que, na política Melhor Última, na grande maioria das vezes apenas o servidor MS é escolhido e invocado, enquanto na Paralela há concorrência no recebimento dos pacotes dos demais servidores. No ciclo 1, os desempenhos das duas políticas se aproximam em virtude da baixa concorrência, decorrente do tamanho reduzido das respostas nesse ciclo.

No cliente BNB, observamos que as políticas estatísticas, embora não tenham tido o melhor desempenho, permaneceram em níveis bem próximos aos resultados obtidos com a política Paralela. A Figura 5.21 mostra o percentual de seleção de servidores da

política Melhor Última, nesse cliente. Da mesma forma observada no cliente Unifor, podemos verificar que os níveis percentuais das duas políticas (Figuras 5.17 e 5.21) guardam estreita relação entre si. A pequena diferença existente entre os dois clientes é decorrente do desempenho da política Paralela, que no cliente BNB lida melhor com a concorrência no recebimento das respostas, ao contrário do que ocorre no cliente Unifor.

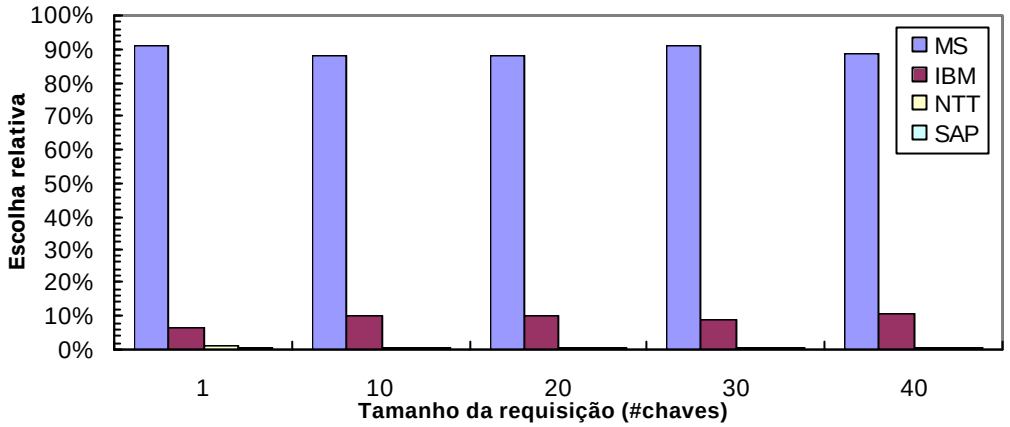

Figura 5.20: Seleção de servidores na política Melhor Última (cliente Unifor).

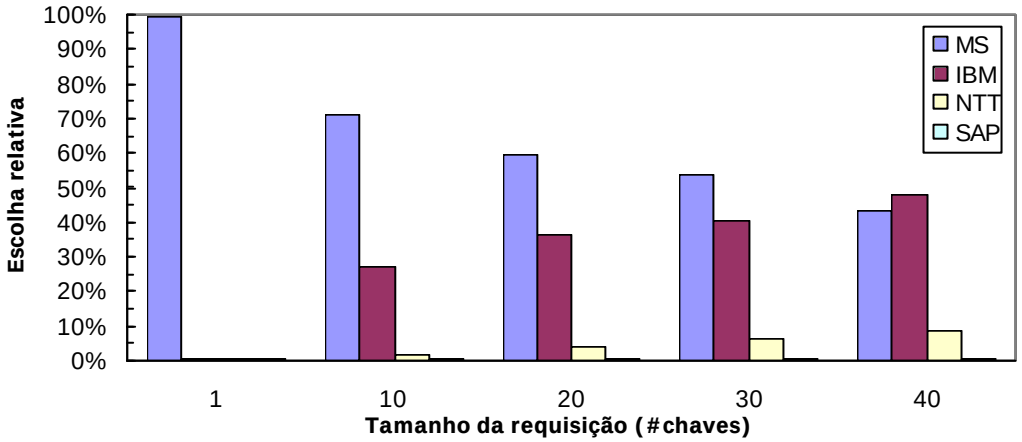

Figura 5.21: Seleção de servidores na política Melhor Última (cliente BNB).

5.2.2.4 Política Melhor Mediana

As Figuras 5.22 e 5.23 apresentam os percentuais de seleção de servidores da política Melhor Mediana, nos dois clientes. Nelas, novamente podemos observar o comportamento similar das políticas estatísticas com a relação à política Paralela.

No cliente Unifor, o nível de seleção do servidor MS na política Melhor Mediana alcançou pontos bem próximos a 100% em todos os ciclos. O servidor NTT não foi selecionado nenhuma vez, e o servidor SAP foi selecionado duas vezes apenas, nos

ciclos 1 e 4. No cliente BNB, o destaque fica para o aumento percentual de escolha do servidor NTT. O servidor MS também fica com níveis próximos a 100% e o servidor SAP

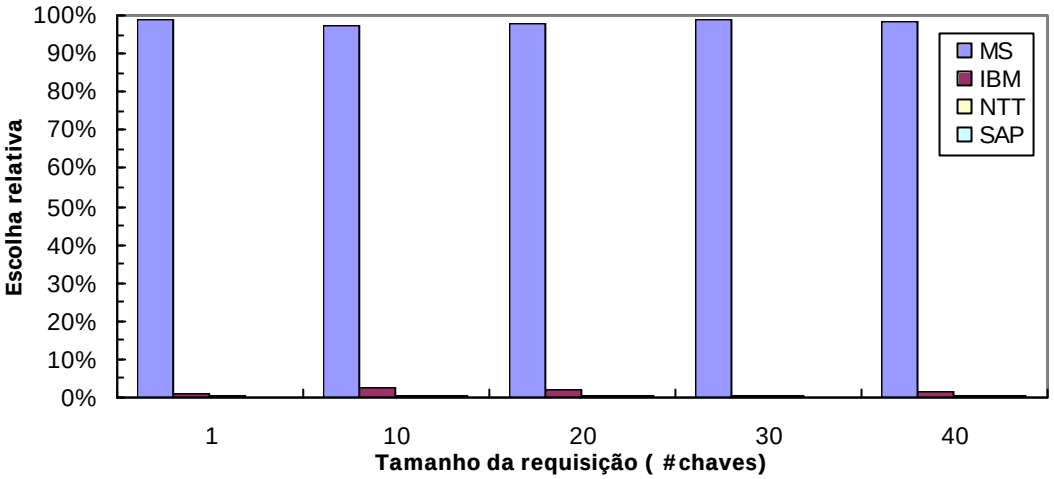

Figura 5.22: Seleção de servidores na política Melhor Mediana (cliente Unifor).

não é selecionado nenhuma vez em nenhum dos ciclos.

5.2.2.5 Distribuição acumulada dos tempos de respostas

Na identificação dos efeitos dos períodos do dia, analisamos o comportamento das políticas, através do tempo mediano de resposta, obtido para cada intervalo de uma hora. Considerando os tempos de resposta como um todo, as políticas apresentam, em determinados instantes, tempos de resposta bastante elevados, ocasionados pelo tempo de conexão, latência do recebimento do primeiro pacote, tempo de recebimento dos demais pacotes e carga no servidor. Mesmo os servidores com melhor desempenho, em alguns momentos, levam minutos, ao invés de segundos, para atender uma chamada de operação, de acordo com o tamanho de resposta esperado.

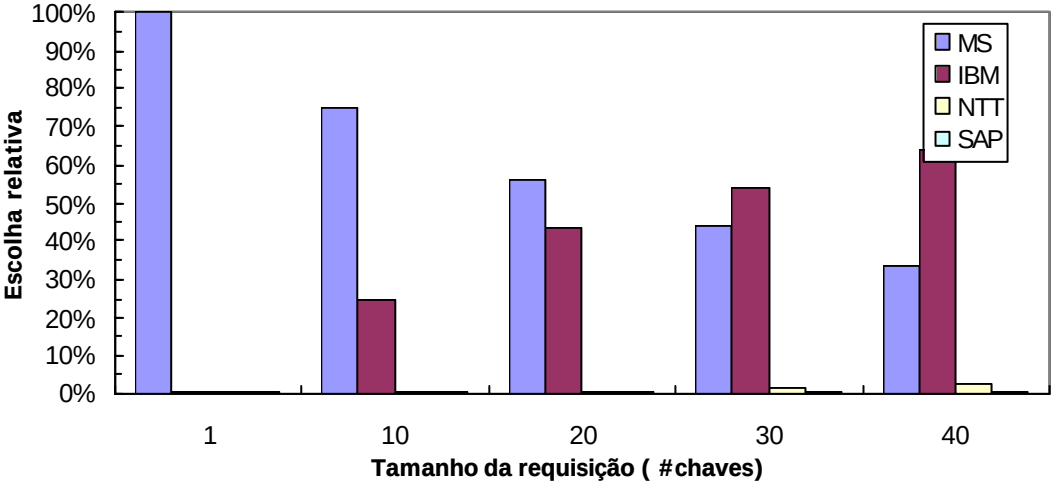

Figura 5.23: Seleção de servidores na política Melhor Mediana (cliente BNB).

Para identificar as políticas com melhor desempenho sob a ótica da faixa total de tempos de respostas observados, computamos a função de distribuição acumulada (Downing e Clark, 2000) do total dos tempos de respostas obtidos em cada política, no ciclo 5. A função de distribuição acumulada permite avaliar, de uma forma mais geral, o desempenho das políticas em termos dos intervalos de tempos de respostas obtidos ao longo de todo o experimento. As figuras 5.24 e 5.25 apresentam a distribuição acumulada dos tempos de respostas obtidos para cada política nos dois clientes. Podemos observar os mesmos padrões de desempenho apresentados na seção 5.2.1.4, com as políticas estatísticas apresentando, no cliente Unifor, melhores tempos de respostas que as demais políticas, e, no cliente BNB, a política paralela produzindo resultados bem próximos aos das políticas estatísticas. A política Randômica, em ambos os clientes, mais uma vez, apresenta o pior desempenho.

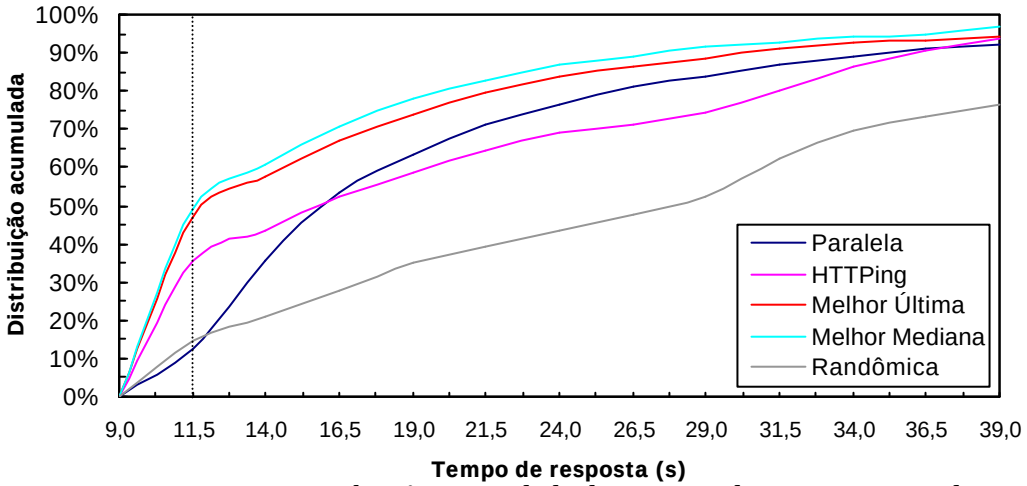

Figura 5.24: Distribuição acumulada dos tempos de respostas no ciclo 5 (cliente Unifor).

A função de distribuição acumulada também permite conduzir uma avaliação mais criteriosa de qual política apresenta os melhores tempos de respostas em dado nível, entre os vários intervalos de tempo observados. Por exemplo, na Figura 5.24, referente ao ciclo 5 do cliente Unifor, podemos constatar que 50% dos tempos obtidos com as políticas estatísticas foram iguais ou inferiores a 11,5 segundos, enquanto a terceira melhor política, HTTPing, tem apenas 35% dos seus tempos obtidos dentro desse intervalo.

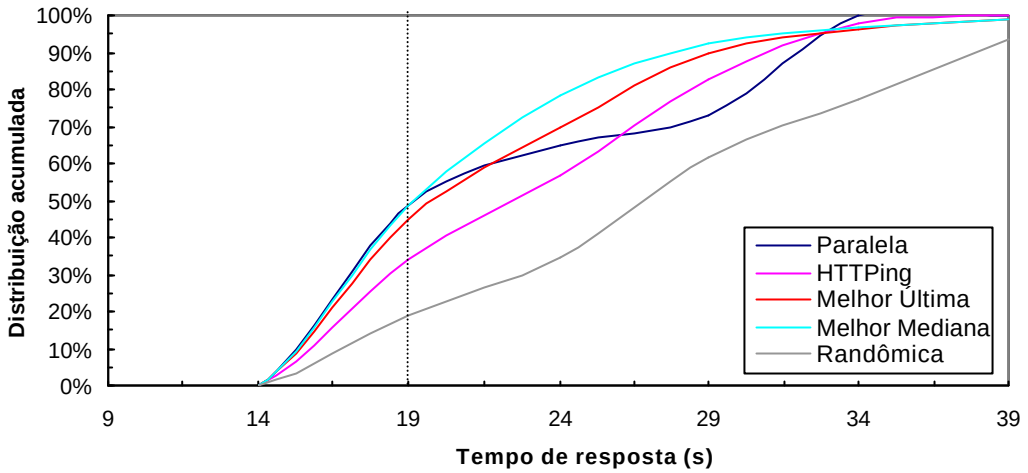

Figura 5.25: Distribuição acumulada dos tempos de respostas no ciclo 5 (cliente BNB).

Já na Figura 5.25, referente ao ciclo 5 do cliente BNB, observa-se, por exemplo, que 50% dos tempos obtidos com as políticas Melhor Mediana e Paralela foram iguais ou inferiores a 19 segundos, ficando as políticas Melhor Última e HTTPing com apenas 45% e 35% dos seus tempos, respectivamente, obtidos dentro desse intervalo.

5.3 Sumário

Neste capítulo, descrevemos a avaliação experimental conduzida, utilizando as cinco políticas de invocação, implementadas no contexto do *framework* RWS. Apresentamos a metodologia empregada na condução dos experimentos, além de uma análise quantitativa e qualitativa dos resultados obtidos.

No capítulo seguinte, apresentamos as conclusões deste trabalho, suas contribuições e sugestões de trabalhos futuros.

Capítulo 6

Conclusão

Este capítulo apresenta as conclusões deste trabalho, incluindo suas principais contribuições, resultados obtidos e propostas para trabalhos futuros.

6.1 Contribuições e resultados

Este trabalho apresentou o *framework* RWS para a invocação transparente de serviços Web replicados. O *framework* incorpora diversas políticas para seleção e invocação de réplicas, de modo a tornar o acesso ao serviço replicado mais eficiente no lado do cliente. As políticas implementadas (randômica, dinâmica paralela, dinâmica HTTPing, estatística Melhor Última e estatística Melhor Mediana) foram avaliadas empiricamente, no ambiente real da Internet. Os experimentos envolveram a invocação de um serviço Web geograficamente replicado em três continentes, utilizando cada uma das políticas implementadas, a partir de dois clientes com diferentes configurações de rede e distribuição de carga de trabalho ao longo do dia. Os resultados mais importantes dos experimentos são sumarizados abaixo.

As políticas estatísticas se apresentaram como a melhor alternativa quando não se leva em consideração características específicas do cliente, a partir de onde a seleção é realizada. Essa conclusão se contrapõe ao resultado obtido por Dykes *et al* (2000), no contexto do acesso a réplicas de recursos tradicionais da Web, onde a política dinâmica de sonda oferecia o melhor desempenho em todos os cenários avaliados. Os desempenhos obtidos pelas duas políticas estatísticas, Melhor Última e Melhor Mediana, ficaram bem próximos, em termos da mediana dos tempos de resposta, em ambos os clientes. No entanto, mais experimentos ainda são necessários para estimar se uma das duas políticas (ou mesmo possíveis variações da política Melhor Mediana com diferentes valores de k) pode vir a oferecer um desempenho significativamente melhor do que a outra. É importante ressaltar que o desempenho das políticas estatísticas depende da existência de dados recentes sobre a situação de cada servidor que hospeda uma réplica do serviço Web. Nos experimentos, este problema foi contornado

invocando separadamente todos os servidores no início de cada ciclo. Os resultados dessas invocações eram então armazenados nas bases históricas, que assim mantinham informações atualizadas sobre o desempenho de cada servidor. Uma solução mais realista envolveria o monitoramento periódico dos servidores, possivelmente por uma entidade externa ao *framework* de invocação, de forma que as bases históricas possam refletir com maior precisão mudanças recentes ocorridas tanto em nível de servidor quanto em nível de rede.

A política paralela apresentou o melhor desempenho entre todas as políticas no cliente onde a largura de banda era suficiente para absorver a concorrência no recebimento das respostas dos quatro servidores. Por outro lado, no cliente onde a largura de banda era mais restrita houve uma degradação gradual no desempenho dessa política na medida em que aumentou o tamanho das respostas recebidas dos servidores. De todo modo, o experimento envolveu apenas quatro servidores. Experimentos adicionais são necessários para determinar a partir de que ponto a relação entre a largura de banda disponível e a concorrência no recebimento das respostas pode inviabilizar a utilização dessa política.

Vale salientar que, nas avaliações de políticas de seleção de servidores realizados no contexto de outros trabalhos, o tempo de acesso medido levava em conta fatores tais como o tempo de transmissão na rede, a carga nos servidores e o tempo de recuperação dos recursos solicitados (documentos, imagens, etc). Nos experimentos conduzidos no contexto deste trabalho, além dos itens acima, o tempo de acesso medido envolveu ainda os custos associados às camadas de software responsáveis pela infra-estrutura de suporte aos serviços Web, tanto no lado do cliente quanto no lado dos servidores. Esses custos compreendem, por exemplo, o tempo de serialização e desserialização das mensagens SOAP, na forma de documentos XML, em ambos os lados, e o tempo de processamento das mensagens SOAP pelo servidor de aplicações, no lado dos servidores.

6.2 Trabalhos futuros

A avaliação empírica de políticas de acesso a serviços Web replicados, utilizada neste trabalho, pode servir como ponto de partida para a condução de novos experimentos envolvendo outros serviços, possivelmente replicados em um número maior de servidores, no

sentido de investigar até que ponto os resultados obtidos neste trabalho podem ser generalizados. Um outro fator que também deve ser considerado é a ampliação do número de clientes, com variações não apenas nas suas características de conexão, mas também nas suas localizações geográficas.

Com relação às políticas de invocação, uma possível linha de trabalho futuro seria explorar novas políticas ou ainda combinar as políticas existentes, criando políticas híbridas, a exemplo do que foi proposto por Hanna *et al* (2001) e Dykes *et al* (2000).

No que se refere ao *framework* RWS, a atual estrutura de classes poderia ser expandida, e novas classes adicionadas à camada de seleção e invocação das réplicas, de modo que as políticas de invocação implementadas possam ser disponibilizadas na forma de um serviço *proxy*. Isto permitiria estender as vantagens oferecidas pelo *framework* RWS a várias aplicações clientes, sem a necessidade de alterar seu código fonte. Outro benefício desta abordagem seria que as aplicações teriam a possibilidade de compartilhar as mesmas bases históricas na aplicação das políticas estatísticas.

Da forma com foi implementado, na atual versão do *framework* RWS, o processo de atualização dos endereços das réplicas de um serviço é feito através de consultas periódicas aos registros UDDI. Uma manutenção mais automática da lista de réplicas poderia ser implementada utilizando o novo serviço de notificação de atualizações recentemente definido pelo padrão UDDI, na sua versão 3.0 (Bellwood *et al*, 2003).

Por fim, uma outra linha de pesquisa seria avaliar o desempenho das políticas de seleção de servidor, no contexto do *framework* RWS, através de simulação. Isto possibilitaria uma maior flexibilidade na elaboração dos cenários de avaliação, bem como um maior controle nas variáveis envolvidas. Por outro lado, os resultados obtidos através de simulação estão fortemente relacionados com o grau de precisão com que as variáveis de controle definidas nos cenários de avaliação refletem o ambiente real, o que poderá comprometer a qualidade da análise haja vista a alta variabilidade de condições de rede na Internet.

Referências Bibliográficas

ALMANAC, 2002, Computer Industry Almanac Inc. Disponível em: http://www.c-i-a.com/pr032102.htm. Acesso em: julho/2004.

AMINI, L., SHAIK, A., SCHULZRINNE, H., 2003, *Modeling Redirection in Geographically Diverse Server Sets,* Proceedings of the Twelfth International Conference on World Wide Web, Budapeste, Hungria, Maio.

ANDREWS, T., CURBERA, F., DHOLAKIA, H., *et al*., 2003, *Business Process Execution Language for Web Services Version 1.1*. Disponível em: http://www-106.ibm.com/developerworks/library/ws-bpel/. Acesso em: julho/2004.

ATKINSON, B., DELLA-LIBERA, G., HADA, S., *et al*., 2002, *Web Services Security (WS-Security)*. Disponível em: http://www-106.ibm.com/developerworks/webservices/library/ws-secure/. Acesso em: julho/2004.

AUSTIN, D., BARBIR, A., FERRIS, C., *et al.*, 2002, Web Services Architecture Requirements. Disponível em: http://www.w3.org/TR/2002/WD-wsa-reqs-20020819#IDAIO2IB. Acesso em: julho/2004.

AZEVEDO, V., PIRES, P. F., MATTOSO, M., 2003, *WebTransact-EM: Um Modelo para a Execução Dinâmica de Serviços Web Semanticamente Equivalentes*, Simpósio Brasileiro de Sistemas Multimídia e Web, WEBMÍDIA2003, Salvador, Brasil.

APACHE, 2002, *The Apache SOAP Project*, Versão 1.0, Junho. Disponível em: http://ws.apache.org/axis/. Acesso em: julho/2004.

BELLWOOD, T., CLÉMENT, L., RIEGEN, C., *Universal Description, Discovery and Integration version 3.0,* Outubro. Disponível em: http://uddi.org/pubs/uddi_v3.htm#_Toc53709323. Acesso em: julho/2004.

BERNERS-LEE, T., GETTYS, J., NIELSEN, H. F.,1996, *Replication and Caching Position Statement*. Disponível em: http://www.w3.org/Propagation/Activity.html. Acesso em: julho/2004.

BIRON, P. V, MALHOTRA, A., 2001, *XML Schema Part 2: Datatypes*, W3C Recommendation 2, Maio. Disponível em: http://www.w3.org/TR/xmlschema-2/#schema. Acesso em: julho/2004.

BIZTALK, 2002, *Microsoft BizTalk Server*. Disponível em: http://www.microsoft.com/biztalk/techinfo/default.asp. Acesso em: julho/2004.

BRYAN, D., DRALUK, V., EHNEBUSKE, D., GLOVER, T, *et al*., 2002, *Universal Description, Discovery and Integration version 2.04,* Julho. Disponível em: http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm. Acesso em: julho/2004.

BOX, D., EHN EBUSKE, D., KAKIVAYA, A., *et al*., 2000, *Simple Object Access Protocol (SOAP) 1.1,* W3C Note, Maio. Disponível em: http://www.w3.org/TR/soap. Acesso em: julho/2004.

BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, M., *et al*., 2000, *Extensible Markup Language (XML), 1.0 Second Edition*, W3C Recommendation, Outubro. Disponível em: http://www.w3.org/TR/REC-xml. Acesso em: julho/2004.

CABRERA, F. L., COPELAND, G., COX, W., *et* al, 2003, *Web Services Coordination (WS-Coordination)*. Disponível em: http://www-106.ibm.com/developerworks/library/ws-coor/. Acesso em: julho/2004.

CABRERA, F. L., COPELAND, G., COX, W., *et* al, 2002, *Web Services Transaction (WS-Transaction)*. Disponível em: http://www-106.ibm.com/developerworks/library/ws-transpec/. Acesso em: julho/2004.

CAULDWELL, P., CHAWLA, R., CHOPRA, V., *et al*., 2001, *Professional XML Web Services*, Editora Wrox Press, Birminghan, Estados Unidos.

CHAPELL, D., JEWELL, T. (2002) "Java Web Services", Editora O'Reilly, 1a. Edição, Março.

CISCO, 2003, *Cisco's Distributed Director e Cisco's Local Director*. Disponível em: http://www.cisco.com/warp/public/cc/pd/cxsr/dd/index.shtml. Acesso em: julho/2004.

CONTI, M., KUMAR, M., DAS, S. K., SHIRAZI, B. A., 2002a, *Quality of Service Issues in Internet Web Services*, IEEE Transactions on Computers, vol. 51, no. 6, Junho.

CONTI, M., GREGORI, E., LAPENNA, W., 2002b, *Replicated Web Services: a Comparative Analisys of Client-based Content Delivery Policies*, Web Engineering and Peer-to-Peer Computing Networking 2002, Pisa, Itália, Maio.

COM, 1992, *Component Object Model*. Disponível em: http://www.microsoft.com/Com/default.asp. Acesso em: julho/2004.

CORBA, 1991, *Commom Object Request Broker Architecture*. Disponível em: http://www.omg.org/gettingstarted/history_of_corba.htm. Acesso em: julho/2004.

CHRISTENSEN, E., CURBERA, F., MEREDITH, G., *et al*., 2001, *Web Services Description Language*, Version 1.1, W3C Note 15 Março. Disponível em: http://www.w3.org/TR/wsdl. Acesso em: julho/2004.

CROVELLA, M. E., CARTER, R. L., 1995, *Dynamic Server Selection in the Internet*, Proceedings of the Third IEEE Workshop on the Architecture and Implementation of High Performance Comunication Subsystems, Connecticut, Estados Unidos, Agosto.

CURBERA, F., KHALAF, R., MUKHI, N., TAI, S., *et al*., 2003, *The Next Step in Web Services*, Communications of the ACM, Outubro, Vol. 46, nº 10.

DAMANI, O., CHUNG, Y., KINTALA, C., WAN, Y., 1997, *ONE-IP: Techniques for Hosting a Service on a Cluster of Machines*, Proceedings of the Sixth International World Wide Web Conference, California, Estados Unidos, Abril.

DAML, 2000, *DARP Agent Markup Language*. Disponível em: http://www.daml.org/. Acesso em: julho/2004.

DCOM, 1996, *Distributed Componente Object* Model. Disponível em: http://www.microsoft.com/com/tech/dcom.asp. Acesso em: julho/2004.

DIAS, D. M., KISH, W., MUKHERJEE, R., TEWARI, R., 1996, *A Scalable and Highly Available Web Server*, Proceedings of the Fortieth-first IEEE Computer Society International Conference, California, Estados Unidos, Fevereiro.

DINGLE, A., PARL, T., 1996, *Web Cache Coherence*, Proceedings of the Fifth International World Wide Web Conference, Paris, França, Maio. Disponível em: http://www5conf.inria.fr/fich_html/papers/P2/Overview.html. Acesso em: julho/2004.

DYKES, S., ROBBINS, A. K., JEFFERY, L.C., 2000, *An Empirical Evaluation of Client-side Server Selection Algorithms*, IEEE INFOCOM 2000, The Conference on Computer Communications, Tel Aviv, Israel, Março.

DNS, 1987, *Domain Names - Concepts And Facilities*, Request for Comments: 1034, novembro. Disponível em: http://www.ietf.org/rfc/rfc1034.txt. Acesso em: julho/2004.

DOWNING, D., CLARK, J., 2000, *Estatística Aplicada*, Editora Saraiva, São Paulo.

ELFWING, R., PAULSSON, U., LUNDBERG, L., 2002, *Performance of SOAP in Web Service Environment Compared to CORBA*, Proceedings of the Ninth Asia-Pacific Software Engineering Conference (APSEC'02), Queensland, Austrália, Dezembro.

FALLSIDE, D. F., 2001, *XML Schema Part 0: Primer*, W3C Recommendation, Maio. Disponível em: http://www.w3.org/TR/xmlschema-0/. Acesso em: julho/2004.

HANNA, M.K., NATAJARAN, N., LEVINE, N.B., 2001, *Evaluation of a Novel Two-Step Server Selection Metric*, Proceedings of the IEEE International Conference on Network Protocols, Califórnia, Estados Unidos, Novembro.

HUNT, G., GOLDSZMIDT, G., MUKHERJEE, R., 1998, *Network Dispatcher: A Connection Router for Scalable Internet Services*, Proceedings of the Seventh International World Wide Conference, Brisbane, Austrália, Abril.

IBM, 2002, *UDDI Business Registry*. Disponível em: http://www-306.ibm.com/software/solutions/webservices/uddi/. Acesso em: julho/2004.

IIOP, 1996, *Commom Object Request Broker Architecture - Internet InterORB Protocol*, disponível em: http://www.omg.org/technology/documents/formal/corba_iiop.htm.

J2EE, 1999, *Java 2 Platform Enterprise Edition*. Disponível em: http://java.sun.com/j2ee/index.jsp. Acesso em: julho/2004.

KEIDL, M., SELTZSAM, S., KEMPER, A., 2002, *Flexible and Reliable Web Service Execution*, Proceedings of the First Workshop on Application Development using XML Web-Service Technology, Alemanha, Julho.

KREGER, H., 2001, *Web Services Conceptual Architecture (WSCA 1.0)*. Disponível em: http://www-3.ibm.com/software/solutions/webservices/documentation.html. Acesso em: julho/2004.

KWAN, T. T., MCGRATH, R. E., REED, D. A., 1995, *NCSA's Word Wide Web Server: Design and performance*, IEEE Computer, Novembro.

LITOUI, M., 2002, *Migrating to Web Services – Latency and Scalability*, Proceedings of the Fourth International Workshop on Web Site Evolution (WSE' 02), Montreal, Canadá, Outubro.

MARKATOS, E. P., CRETE, H., 1996, *Main Memory Caching of Web Documents*, Proceedings of the Fifth International World Wide Web Conference, Paris, França, Maio. Disponível http://www5conf.inria.fr/fich_html/papers/P1/Overview.html. Acesso em: julho/2004.

MARTINS, G. A., 2001, *Estatística Geral e Aplicada*, Editora Atlas, São Paulo.

MICROSOFT, 2002a, *Visual Studio .NET e Web Service*s, versão 7.0. Disponível em: http://www.microsoft.com/net/ e http://msdn.microsoft.com/webservices/. Acesso em: julho/2004.

MICROSOFT, 2002b, *UDDI Business Registry*. Disponível em: http://uddi.microsoft.com. Acesso em: julho/2004.

NTT, 2002, *UDDI Business Registry*. Disponível em: http://www.ntt.com/uddi/index-e.html. Acesso em: julho/2004.

OASIS, 2002, *UDDI Version 2 WSDL Service Interface Descriptions*. Disponível em: http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv2. Acesso em: julho/2004.

OBRACZKA, K., SILVA, F., 2000, *Network Latency Metrics for Server Proximity*, Proceedings of the IEEE Globecom, San Francisco, Estados Unidos, Dezembro.

OMG, 1989, *Object Management Grou*. Disponível em http://www.omg.org. Acesso em: julho/2004.

PADOVITZ, A., KRISHNASWAMY, S., LOKE, S.W., 2003, *Towards Efficient Selection of Web Services*, Second International Joint Conference on Autonomous Agents and Multi-agent Systems, New York, Estados Unidos, Julho.

PAPAZOGLOU, M. P., GEORGAKOPOULOS, D., 2003, *Service-Oriented Computing: An Introduction*, Communications of the ACM, volume 46, Outubro.

RASTOGI, K. K., 1999, *Replication of Documents in the World Wide Web*, Indian Institute of Technology, Kanpur, Índia, Março, Dissertação de Mestrado.

RDF, 2000, *Resource Description Framework*. Disponível em: http://www.w3.org/RDF/#overview. Acesso em: julho/2004..

RMI, 1997, *Java Remote Method Invocation*. Disponível em: http://java.sun.com/products/jdk/rmi/index.jsp. Acesso em: julho/2004.

RODRIGUEZ, P., BIERSACK, E.W., 2002, *Dynamic Parallel Access to Replicated Content in the Internet*, IEEE/ACM Transactions on Networking, vol. 10, nº 4, Agosto.

SAP, 2002, *UDDI Business Registr*. Disponível em: http://uddi.sap.com. Acesso em: julho/2004.

SAYAL, M. , BREITBART, Y., SCHEUERMANN, P., VINGRALEK, R., 1998, *Selection Algoritms for Replicated Web Servers*, Proceeding of the Workshop on Internet Server Performance, Wisconsin, Estados Unidos, Junho.

SCHEMERS, R. J., 1995, *lbmnamed: a Load Balancing Name Server*, Proceedings of the Ninth Systems Administration Conference, California, Estados Unidos, Setembro.

SILVA, J. A. F., MENDONÇA, N. C., 2004, *Uma Avaliação Empírica de Políticas de Invocação para Serviços Web Replicados*, 22º Simpósio Brasileiro de Redes de Computadores, Rio Grande do Sul, Brasil, Maio.

SNELL, J., TIDWELL, D., KULCHENKO, P., 2001, *Programming Web Services with SOAP*, O'Reilly, 1ª Edição, Dezembro.

SUN, 2003, *Java Web Services Developer Pack* , versão 1.0, SUN Microsystems. Disponivel em: http://java.sun.com/webservices/downloads/webservicespack.html. Acesso em: julho/2004.

TABOR, R., 2001, *Microsoft .NET XML Web Services*, Editora SAMS, Dezembro.

VINGRALEK, R., BREITBART, Y., SAYAL, M., SCHEUERMANN, P., 1999, *Web++: A System for Fast and Reliable Web Service*, Proceedings of the USENIX Annual Technical Conference, California, Estados Unidos, Junho.

W3C, 1994, *World Wide Web Consortium*. Disponível em: http://www.w3.org/Consortium/. Acesso em: julho/2004.

YOSHIKAWA, C., CHUN, B., ESATHMA, P., VAHDAT, A., ANDERSON T., CULLER, D., 1997, *Using Smart Clients to Build Scalable Services*, Proceedings of the USENIX Annual Technical Conference, California, Estados Unidos, Janeiro.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo