Failover transparente ao cliente em Serviços Web: Uma extensão ao WS-Addressing

José Ricardo da Silva[1*] **e Irineu Sotoma**[1]

[1]Departamento de Computação e Estatística – Universidade Federal de Mato Grosso do Sul (UFMS)
Caixa Postal 549 – 79070-900 – Campo Grande – MS – Brazil

contato@josericardo.eti.br, isotoma@dct.ufms.br

Abstract. *In spite of the great number of works on fault tolerance in web services, none of them, to the best knowledge of the authors, establishes a client-transparent failover mechanism without proxies. The main contribution of this paper is an extension to the WS-Addressing standard which allows the specification of replicas of a service and defines the way endpoints shall interact. Additionally, evaluation of a prototype, based on the Axis2 web services framework, that implements a subset of the extension is also given.*

Resumo. *Apesar do grande número de trabalhos sobre tolerância a falhas em Serviços Web baseados em técnicas de replicação, até onde vai o conhecimento dos autores, nenhum deles realiza o failover de maneira transparente ao cliente sem utilizar proxies. A principal contribuição deste artigo é uma extensão ao padrão WS-Addressing de Serviços Web, que permite a especificação de réplicas de um serviço e define a forma como as interações entre os endpoints devem ocorrer. Adicionalmente, a avaliação de um protótipo, baseado no framework Axis2 e que implementa um subconjunto da extensão, também é fornecida.*

1. Introdução

Arquiteturas Orientadas a Serviços (SOA) [Huhns and Singh 2005, Papazoglou and Georgakopoulos 2003] têm sido adotadas nos mais variados cenários. A possibilidade de criar sistemas independentes de plataforma e de linguagem de programação e a possibilidade de integração de sistemas de maneira transparente fornecem novas oportunidades aos desenvolvedores. Conforme a adoção de SOAs for aumentando, como é previsto [Birman 2006], e sistemas críticos em termos de dependabilidade passem também a utilizar Serviços Web[1], mecanismos para garantir o funcionamento confiável destes sistemas deverão ser criados e padronizados. Em caso de falhas críticas, apenas técnicas de replicação podem garantir o acesso aos dados críticos, além de permitir uma reação coordenada [Birman 2006, Birman et al. 2004]. Assim sendo, a primeira melhoria necessária ao conjunto atual de especificações relativas

*Pesquisa desenvolvida com suporte financeiro da Fundação de Apoio ao Desenvolvimento do Ensino, Ciência e Tecnologia do Estado de Mato Grosso do Sul (Fundect), proc. 41/100.270/2006 e suporte parcial do CNPq proc. 620171/2006-5.

[1]O termo "Serviço Web" neste texto é utilizado para referenciar serviços baseados em padrões do *World Wide Web Consortium* (W3C) e da *Organization for the Advancement of Structured Information Standards* (OASIS), como o SOAP, a *Web Services Definition Language* (WSDL) e o restante de padrões WS-*.

a Serviços Web é a definição de mecanismos que permitam a utilização de serviços replicados.

Trabalhos anteriores sobre tolerância a falhas (TF) em Serviços Web podem ser divididos, basicamente, em duas categorias: os que utilizam replicação (ativa [Schneider 1990], passiva [Budhiraja et al. 1993] ou uma variação, como a replicação semi-passiva [Défago and Schiper 2004], por exemplo) e os que utilizam implementações diferentes da mesma interface de serviço (*n-Versões* [Avizienis 1985]).

Ortogonalmente às abordagens de TF se encontra a transparência da solução de TF ao cliente. Apesar do inegável valor das propostas que não são transparentes ao cliente, na prática é mais fácil adotar uma solução transparente. Isto pode ser dito porque, especificamente no caso de Serviços Web, é grande a heterogeneidade entre os clientes.

O principal objetivo da tecnologia de Serviços Web é permitir um alto nível de interoperabilidade entre as soluções. Para tornar isto possível, a arquitetura possui padrões abertos como alicerce. Apesar da existência de vários padrões já estabelecidos pelo W3C e pela OASIS, o tópico de TF ainda não foi padronizado. Há várias propostas (ver Seção 4.1) para adicionar TF às soluções, entretanto, até onde vai o nosso conhecimento, todas utilizam soluções *ad-hoc* para tratar a questão do *failover*[2]. A única proposta que estende um padrão é a de Fang et al., que estende a WSDL [W3C 2007]. Esta proposta, entretanto, apresenta algumas deficiências que esperamos ter sanado com este trabalho (ver Seção 4).

A maioria das arquiteturas de TF propostas não podem ser consideradas transparentes ao cliente, pois nestes casos ou a implementação do serviço (lógica de negócio) precisa ser modificada para tratar as falhas e interações com as réplicas ou o *failover* é tratado por código do *middleware* de TF. A primeira abordagem é indesejável, pois aumenta o nível de acomplamento entre a solução de TF e a implementação do serviço. No segundo caso, o desenvolvedor que consome o serviço precisa utilizar a API do *middleware* de TF, dificultando a adição de TF a serviços legados e, em alguns casos, reduzindo o nível de interoperabilidade das soluções, contrariando uma das premissas fundamentais de Serviços Web.

Por outro lado, as soluções transparentes de que temos conhecimento (ver Seção 4) não podem ser consideradas totalmente tolerantes a falhas. Nestes casos a transparência é alcançada através da inserção de um *proxy*, responsável pelas interações com as réplicas. Contudo, erros de hardware e software ainda podem ocorrer nos *proxies*, além de partições na rede. Assim, uma alternativa simples para atingir um maior nível de TF nas interações com um serviço é tornar o cliente ciente da replicação do mesmo. Para que seja possível fazer isto de maneira transparente, propomos a extensão de um padrão amplamente difundido [Weerawarana et al. 2005, Hansen 2007, Foster et al. 2008]: o WS-Addressing, principalmente pelo fato de ser este o padrão responsável pelo endereçamento de *endpoints*[3]. A extensão permite a especificação de réplicas de um serviço e fica encapsulada no WS-Addressing. Por este motivo, pode ser considerada transparente ao cliente: o único componente a ser modificado na pilha de Serviços Web é o módulo responsável

[2]Tratamento da quebra da réplica primária de um serviço.

[3]O WS-Addressing define um *endpoint* como sendo uma entidade, processador ou recurso referenciável ao qual mensagens podem ser endereçadas [W3C 2006a].

pelo processamento do WS-Addressing[4]. Uma vez que o código responsável pelo tratamento do WS-Addressing, tanto no cliente quanto no servidor, atenda à especificação do WS-Addressing em conjunto com a extensão proposta, será possível realizar o *failover* de maneira transparente, independentemente da plataforma de Serviços Web utilizada.

O restante deste artigo é organizado da seguinte maneira: a Seção 2 descreve a extensão ao WS-Addressing. A Seção 3 detalha a criação do protótipo e sua avaliação. A Seção 4 revisita algumas propostas de TF, correlacionando-as com a extensão proposta e, por fim, a Seção 5 conclui o artigo e fornece algumas sugestões de trabalhos futuros.

2. Estendendo o WS-Addressing

O WS-Addressing é um padrão que visa a fornecer um mecanismo de endereçamento de Serviços Web independente de protocolo de transporte. O padrão define um conjunto de propriedades que são utilizadas para referenciar Serviços Web e facilitar o endereçamento de *endpoints* nas mensagens [W3C 2006a]. A criação do WS-Addressing foi necessária porque o SOAP não define uma maneira padrão de se especificar o destino de uma mensagem, como retornar uma resposta e nem para onde enviar mensagens de erro. Antes da criação do WS-Addressing, todas estas informações eram delegadas ao protocolo na camada de transporte (o HTTP, por exemplo). Esta Seção define como o WS-Addressing pode ser estendido para tornar possíveis a especificação de réplicas e o tratamento de falhas (*failover*).

2.1. Mensagens de erro relevantes definidas pelo WS-Addressing

Além de ser o padrão responsável pela definição dos mecanismos de endereçamento de *endpoints*, é o WS-Addressing que define [W3C 2006b], também, as duas mensagens de erro pertinentes à extensão:

- *Destination Unreachable*: recebida pelo consumidor do serviço quando o *endpoint* de destino não pode ser alcançado através da rede.
- *Endpoint Unavailable*: enviada ao consumidor quando o fornecedor não é capaz de processar a requisição devido a uma falha temporária ou permanente. Opcionalmente, pode-se inserir no corpo desta mensagem de erro um parâmetro chamado *RetryAfter*, que permite a definição de um intervalo mínimo, em milissegundos, após o qual a requisição poderá ser retransmitida.

A definição destas duas mensagens de erro relevantes à TF pelo WS-Addressing, juntamente com o que foi exposto na Seção 1, corroboram a escolha do WS-Addressing como sendo o padrão mais apropriado para ser estendido.

2.2. Tratamento de erros no WS-Addressing 1.0

O WS-Addressing define um elemento que pode ser utilizado para tratar falhas, o *FaultTo*. Este elemento permite que o emissor de uma mensagem especifique um terceiro *endpoint* ao qual o recebedor da mensagem deve enviar possíveis mensagens de erro. Entretanto, o uso deste elemento como base para a adição de técnicas de TF é limitado.

A Fig. 1 ilustra uma interação de requisição e resposta típica em que o *endpoint* C acessa um serviço fornecido pelo *endpoint* S. Primeiramente, C obtém o documento

[4]Os padrões geralmente são implementados como módulos dos *frameworks* de Serviços Web.

WSDL que define a interface de S. Este pode ser obtido de um terceiro *endpoint* W (mensagens 1 e 2) ou diretamente de S. Em posse da interface de S, C gera um trecho de código chamado *stub*[5]. A principal funcionalidade dos stubs é facilitar a vida do programador, permitindo que estruturas da própria linguagem de programação sejam utilizadas para interagir com os serviços ao invés de manipular diretamente as mensagens em XML. A invocação de um método, por exemplo, torna-se uma invocação de serviço, feita pelo *framework* de Serviços Web.

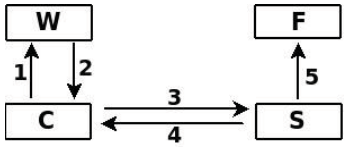

Figura 1. Possíveis interações entre nós utilizando Serviços Web.

A partir de então, C realiza requisições (mensagem 3) e S envia as respectivas respostas (mensagem 4) enquanto não há erros. Em caso de erro, S pode enviar uma mensagem de erro para C ou para um quarto *endpoint* F (mensagem 5) especificado pelo elemento *FaultTo* definido nos cabeçalhos da mensagem 3.

Ao se utilizar o elemento *FaultTo* como único meio de tratamento de falhas em uma solução baseada em Serviços Web, duas possibidades são descartadas: a de aumentar a dependabilidade do sistema por meio da replicação de S utilizando apenas padrões estabelecidos e a de realizar o *failover* de maneira transparente ao cliente. Como apenas uma mensagem de erro é enviada a F, algum tipo de protocolo de recuperação de falhas *ad-hoc* deve ser estabelecido entre F, C, S e, possivelmente, as réplicas de S.

Outro problema presente no esquema de tratamento de falhas atual definido pelo WS-Addressing é o fato de a geração de mensagens do tipo *Destination Unreachable* ser opcional. Se deseja-se que o mecanismo de tolerância a falhas funcione de maneira independente do protocolo de transporte, a geração desta mensagem deve ser *obrigatória*.

2.3. Novos elementos inseridos pela extensão

Para permitir a definição de réplicas de um serviço, complementando o elemento *FaultTo*, e permitindo a construção de soluções mais confiáveis, propomos a adição dos elementos *Replicas*, *Replica* e *Metadata* ao WS-Addressing. Estes elementos são definidos neste artigo como se já fossem parte do *XML Schema* do WS-Addressing 1.0 [W3C 2006c], conforme definido na Fig. 2.

Um elemento do tipo *Replica* é responsável pela definição do endereço de uma réplica. Este elemento é do tipo *EndpointReferenceType* já definido pelo WS-Addressing [W3C 2006c], utilizado para especificar o endereço de um *endpoint*.

Todos os elementos do tipo *Replica* em um cabeçalho devem ter um, e apenas um, elemento pai do tipo *Replicas*, que tem como funções encapsular a lista de réplicas e definir o estilo de replicação (ver Seção 2.4). Opcionalmente, seguindo a premissa de

[5]A maioria dos frameworks modernos para Serviços Web fornecem ferramentas para geração de *stubs* que recebem como entrada a interface WSDL de um serviço. Uma vez que o *stub* é gerado (ou implementado), todas as invocações realizadas por meio dele invocarão o *endpoint* definido na interface do serviço.

```
<xs:complexType name="ReplicasReferenceType">
   <xs:sequence>
     <xs:element name="Replica" type="tns:EndpointReferenceType" minOccurs="1"
                 maxOccurs="unbounded"/>
     <xs:element ref="tns:Metadata" minOccurs="0"/>
   </xs:sequence>

   <xs:attribute name="style" default="passive">
     <xs:simpleType> <xs:restriction base="xs:string">
         <xs:enumeration value="passive"/>
         <xs:enumeration value="active"/>
       </xs:restriction> </xs:simpleType>
   </xs:attribute>
</xs:complexType>

<xs:element name="Replicas" type="tns:ReplicasReferenceType"/>
```

Figura 2. *XML Schema* dos elementos da extensão.

extensibilidade de Serviços Web, o elemento *Replicas* pode também possuir um elemento filho *Metadata*, já definido pelo WS-Addresing 1.0. Este elemento pode conter qualquer tipo de metadados peculiares a uma aplicação.

Serviços replicados devem inserir um, e apenas um, elemento *Replicas* nos cabeçalhos de todas as mensagens enviadas. Essa convenção permite uma implementação mais direta e simples de se gerenciar. Outras opções seriam enviar a lista de réplicas apenas quando esta fosse atualizada ou fazer com que o cliente enviasse, a cada requisição, a sua lista local das réplicas do serviço. Neste último caso, o servidor enviaria uma nova lista de réplicas sempre que houvesse disparidade entre a versão recebida e a sua versão local. Testar possíveis ganhos de desempenho através destas outras abordagens é uma sugestão para trabalhos futuros.

2.4. Estilos de replicação

A extensão permite a utilização de dois estilos de replicação pelo fornecedor do serviço: passiva ou ativa. O estilo é definido através do atributo *style* do elemento *Replicas*.

Na replicação passiva (*style*=“passive”) apenas uma réplica (primária) está ativa em um determinado instante. Assim, requisições devem ser feitas ao *endpoint* definido pelo primeiro elemento *Replica* filho de *Replicas*. Em caso de falha, a requisição é feita à próxima réplica da lista de réplicas. Neste caso, a réplica contactada pode retornar a resposta da requisição, caso ela seja a nova primária, ou pode retornar uma mensagem de erro do tipo *Endpoint Unavailable* (ver Seção 2.1) com a lista de réplicas atualizada em seus cabeçalhos e com o parâmetro *RetryAfter* definido[6]. No segundo caso, a lista de réplicas no cliente é atualizada e garante-se que, após o tempo definido por *RetryAfter*, as interações poderão ser reestabelecidas.

Por outro lado, caso a replicação ativa (*style*=“active”) esteja sendo utilizada pelo fornecedor do serviço, todas as réplicas estão ativas e devem responder às requisições. Logo, estas devem ser enviadas para todas réplicas. O módulo do WS-Addressing no cliente fica responsável por definir o critério de entrega da resposta à aplicação. Um critério geralmente utilizado é o de esperar apenas pela primeira resposta recebida de

[6]Esta proposta não define um valor específico para *RetryAfter*, delegando essa responsabilidade ao gerenciador de réplicas no lado do servidor.

qualquer uma das réplicas. Outras opções podem vir a ser definidas, por exemplo, através do elemento *Metadata*.

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
            xmlns:wsa="http://www.w3.org/2005/08/addressing">
 <S:Header>
  <wsa:MessageID>http://example.com/1</wsa:MessageID>
  <wsa:Replicas>
   <wsa:Replica>
     <wsa:Address>http://rep1.example.com</wsa:Address>
   </wsa:Replica>
   <wsa:Replica>
     <wsa:Address>http://rep2.example.com</wsa:Address>
   </wsa:Replica>
  </wsa:Replicas>
  <wsa:To>http://client.example.com</wsa:To>
 </S:Header>
 <S:Body> ... </S:Body>
</S:Envelope>
```

Figura 3. Utilizando o elemento *Replicas* em um cabeçalho SOAP.

2.5. Um Exemplo

Considere que um serviço possua duas réplicas localizadas em *http://rep1.example.com* (*Rep1*) e *http://rep2.example.com* (*Rep2*). A Fig. 3 ilustra um exemplo simples de utilização do elemento *Replicas*. Este deve ser inserido nos cabeçalhos das mensagens de resposta, possibilitando a identificação das réplicas do serviço pelo seu consumidor.

Agora considere o cenário de replicação passiva da Fig. 4, em que mensagens são identificadas por uma tupla (endereço, tipo de mensagem, identificador da mensagem) e em que há dois tipos de mensagem: requisição (Req) e resposta (Res). Os endereços nas mensagens correspondem aos elementos *To* e *From* do WS-Addressing nas requisições e nas respostas, respectivamente.

Na Fig. 4 o cliente recebe a resposta de sua primeira requisição e então *Rep1* quebra. Por sorte, o elemento *Replicas* da Fig. 3 estava contido nos cabeçalhos da mensagem de resposta 1. Assim, quando o cliente faz a segunda invocação ao serviço, o *stub* invocará *Rep1*[7], mas o módulo do WS-Addressing (que fica após o *stub* na cadeia de processamento das mensagens) no lado do cliente detecta que *Rep1* quebrou e contacta *Rep2*. Como esta responde à requisição, as requisições subsequentes são redirecionadas automaticamente para *Rep2* pelo módulo do WS-Addressing, mas este altera as respostas para que a aplicação as enxergue como tendo sido geradas por *Rep1*.

3. O Protótipo

Para implementar o protótipo da extensão, escolhemos alterar o módulo responsável pelo processamento do WS-Addressing no *framework* Apache Axis2 1.4.1 em Java [The Apache Software Foundation 2008]. O Axis2 é um *framework* para desenvolvimento de Serviços Web maduro que oferece suporte às últimas versões de alguns padrões WS-*, dentre eles o WS-Addressing 1.0. O protótipo atual suporta apenas a replicação passiva. A realização de reinvocações a partir do recebimento de *Endpoint Unavailable* ainda não está completa e, por isto, não fez parte dos testes apresentados neste artigo. Esta Seção descreve seus componentes e seu funcionamento. Ao final alguns resultados preliminares de desempenho são fornecidos.

[7] O *stub* foi gerado com *Rep1* como *endpoint* de destino.

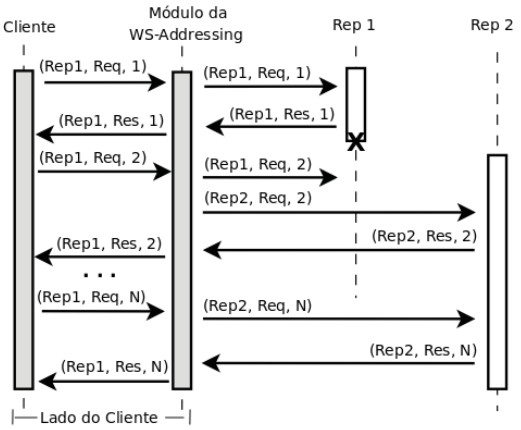

Figura 4. Comportamento da extensão em caso de falha da réplica primária.

3.1. Estrutura do Axis2

A unidade básica para a manipulação de mensagens no Axis2 é chamada *handler*. Cada *handler* implementa um método que recebe como parâmetro, entre outros dados, o envelope SOAP, podendo manipulá-lo como desejar. Os padrões WS-* são implementados no Axis2 através de conjuntos de *handlers*, responsáveis pela interpretação ou inserção dos cabeçalhos específicos de cada padrão. Os *handlers* podem ser agrupados em fases que, por sua vez, constituem os fluxos do Axis, conforme ilustrado pela Fig. 5a.

Um fluxo nada mais é que um conjunto de fases, logo de *handlers*, responsáveis pelo processamento das mensagens antes que elas sejam entregues à aplicação, no caso dos fluxos de entrada, ou antes que elas sejam enviadas pela rede, no caso dos fluxos de saída. O Axis2 fornece quatro abstrações de fluxo de mensagens: o *InFaultFlow*, fluxo de entrada para mensagens de erro; o *InFlow*, fluxo de entrada para mensagens que não sejam de erro; o *OutFaultFlow*, fluxo de saída para mensagens de erro; e o *OutFlow*, fluxo de saída para mensagens que não sejam de erro.

Além dos fluxos, o Axis2 também possui componentes localizados após os fluxos de saída, chamados *Transport Senders*. Um *Transport Sender* é invocado após o processamento da mensagem em um fluxo de saída e é responsável pelo envio da mensagem na rede, de acordo com o protocolo de camada de transporte que implementa[8].

3.2. Organização do protótipo

A implementação atual do protótipo é composta por cinco componentes (ver Fig. 5b). São quatro *handlers* (ver Seção 3.3) e um *Transport Sender* (ver Seção 3.4). Além disso, utilizamos também o banco de dados Apache Derby [The Apache Software Foundation 2009] para armazenar as listas de réplicas de maneira persistente no cliente. Ele é acessado através de uma classe intermediária *AddressingStorageHelper*, que permite a troca de meio de armazenamento sem a necessidade de modificar o restante do protótipo.

3.3. Os *Handlers*

Os quatro *handlers* são ilustrados em caixas cinzas na Fig. 5b. O comportamento de cada um deles é descrito a seguir:

[8]Vários protocolos de camada de transporte são implementados no Axis2. Para cada um deles, há um *Transport Sender* específico.

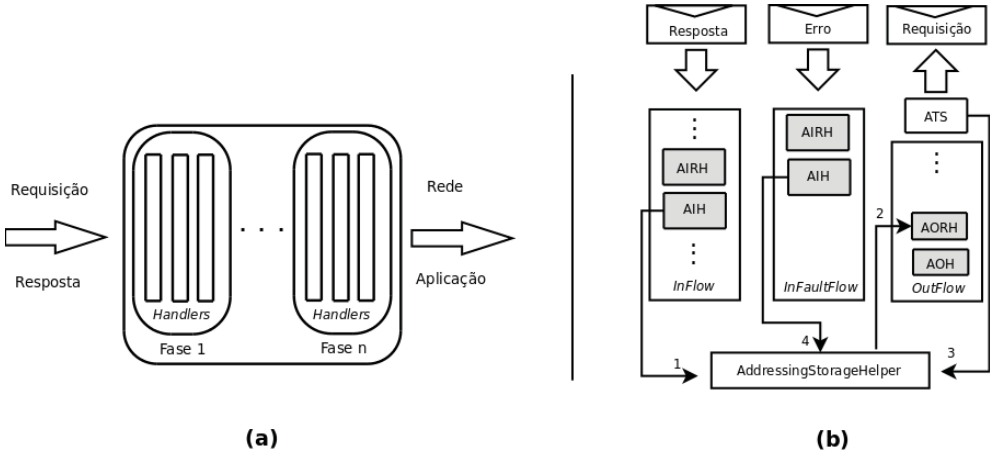

Figura 5. (a) Um fluxo do Axis2 no lado do cliente. (b) Estrutura do protótipo.

- *AddressingInHandler* (AIH): trata-se do *AddressingInHandler* distribuído juntamente com o módulo WS-Addressing do Axis2, responsável por interpretar os cabeçalhos do WS-Addressing das mensagens recebidas. Ele foi modificado para também ser capaz de interpretar o elemento *Replicas* e armazenar a lista de réplicas (mensagens 1 e 4 na Fig. 5b). Sempre que uma nova lista de réplicas é recebida, a lista antiga é descartada. O AIH, portanto, constitui a primeira porta de entrada das listas de réplicas no sistema, baseando-se apenas nos cabeçalhos do WS-Addressing. Contudo, futuramente, a lista pode vir a ser definida em outros locais (ver Trabalhos Futuros na Seção 5).
- *AddressingInReplicationHandler* (AIRH): localizado no *InFlow* e no *InFaultFlow*, modifica os cabeçalhos das mensagens recebidas para que estes correspondam ao *endpoint* contactado originalmente. Ele é necessário pois as respostas recebidas, advindas de um *failover* anterior, possuirão o endereço da nova réplica primária como origem.
- *AddressingOutHandler* (AOH): trata-se do *AddressingOutHandler* distribuído juntamente com o Axis2, responsável pela inserção dos cabeçalhos do WS-Addressing nas mensagens enviadas. A única modificação necessária foi a inserção de um método para inserir também o elemento *Replicas*, caso uma lista de réplicas tenha sido especificada no arquivo de configuração do módulo do WS-Addressing.
- *AddressingOutReplicationHandler* (AORH): responsável por verificar (mensagem 2 na Fig. 5b) se o serviço sendo contactado possui réplicas, ou seja, o elemento *Replicas* foi recebido nos cabeçalhos de uma mensagem anterior. Caso o serviço possua réplicas, o *Transport Sender* atual é substituído pelo *AddressingTransportSender* (ver Seção 3.4).

3.4. O AddressingTransportSender

O *AddressingTransportSender* (ATS), assim como qualquer outro *Transport Sender*, é responsável pelo envio das mensagens na rede. Entretanto, além de enviar a mensagem, ele é também responsável por monitorar a sua chegada no *endpoint* de destino e, em caso de falha, invocar as outras réplicas do serviço. Isto é feito por meio dos seguintes passos:

- Passo 1: Modificar os cabeçalhos do WS-Addressing da mensagem sendo enviada de maneira a refletir uma possível mudança de réplica primária (ver Passo 5).
- Passo 2: Invocar o serviço.
- Passo 3: Se nenhuma falha for detectada, o cliente recebe a resposta normalmente.
- Passo 4: Se a réplica primária estiver inacessível, a próxima primária é extraída da lista de réplicas. Se não houver mais réplicas disponíveis, uma mensagem *Destination Unreachable* é enviada para a aplicação e o processo termina.
- Passo 5: O endereço da nova réplica primária é atualizado no meio de armazenamento (mensagem 3 na Fig. 5b) e o processo de *failover* volta para o Passo 1.

3.5. Considerações em relação à implementação

Uma decisão de implementação importante que deve ser feita se refere ao tempo de vida da lista de réplicas no cliente. Se a lista é armazenada em memória, esta será perdida quando a sessão[9] expirar. Caso isto ocorra, se a réplica primária original quebrar e não for substituída ou não se recuperar, interações futuras não serão possíveis, a não ser que um novo *stub* do serviço seja gerado para corresponder ao endereço de uma nova réplica primária.

Uma solução para o problema de perda da lista de réplicas é utilizar um meio de armazenamento permanente, assim como é feito pelo protótipo. Neste caso, o nível de TF é estendido: se a réplica primária não se recuperar, enquanto as outras réplicas continuarem interagindo com o cliente e, por consequência, continuarem atualizando a lista de réplicas, a obtenção do serviço será possível. O custo desta solução é baixo: basta armazenar, juntamente com a lista de réplicas, o endereço da réplica primária original, que é contactada pelo *stub*.

Por fim, é preciso ressaltar que algumas medidas de segurança também precisam ser implementadas. Em relação ao elemento *Replicas*, as medidas devem ser as mesmas adotadas pelo WS-Addressing 1.0 em relação aos elementos *FaultTo* e *ReplyTo* [W3C 2006a]. Isto significa estabelecer um mecanismo que garanta que o emissor da lista de réplicas (fornecedor do serviço) está habilitado a redirecionar fluxos de mensagens para os *endpoints* especificados pela mesma.

3.6. Considerações em relação ao desempenho

Para medir o desempenho do protótipo foram usadas três máquinas em rede local de 100 Mbps, cada uma com as seguintes características: processadores Intel Core 2 Duo E8300 com 4 GB de RAM rodando a distribuição de linux Ubuntu 8.10 e a JVM 1.6.0_0. Os serviços foram hospedados no servidor de aplicações Apache Tomcat 6.0.18. Inicialmente o banco de dados foi populado com cem *listas* de réplicas.

Primeiramente, é importante notar que apenas a inicialização do banco de dados demora 0,5 segundo, em média. Não é possível fugir dessa sobrecarga, mesmo quando apenas serviços não replicados são acessados, pois o banco de dados é consultado ao menos uma vez para verificar se alguma lista relativa ao serviço já foi armazenada no cliente anteriormente.

[9]Intervalo de tempo em que o processo cliente estiver ativo.

Durante os testes foi invocado um serviço de *echo* (que apenas retorna a string fornecida como parâmetro) e cada invocação possuía um parâmetro de 2 KB. Testes com outros tamanhos de parâmetro foram realizados, mas não foi percebida uma grande influência nos resultados finais. Os experimentos avaliaram a sobrecarga imposta pela extensão em quatro cenários:

1. O serviço foi invocado utilizando o Axis2 não modificado. Este cenário serve de base para as comparações.
2. O serviço foi invocado utilizando a extensão, mas o elemento *Replicas* não foi enviado nos cabeçalhos, simulando a utilização de um serviço não replicado.
3. O serviço foi invocado utilizando a extensão, o elemento *Replicas* foi enviado, mas não há quebra da réplica primária.
4. Similar ao item 3, porém, com quebra da réplica primária. A quebra da réplica primária foi simulada da seguinte maneira: após trinta invocações a *thread* da aplicação cliente foi suspendida por cinco segundos e neste intervalo de tempo o servidor de aplicações (Tomcat) foi desligado. Ao retornar à execução, o cliente identificou a falha[10], porque o Tomcat não estava mais escutando na porta TCP, e realizou o *failover*.

Os tempos de resposta obtidos por meio de requisições consecutivas são ilustrados pela Fig. 6.

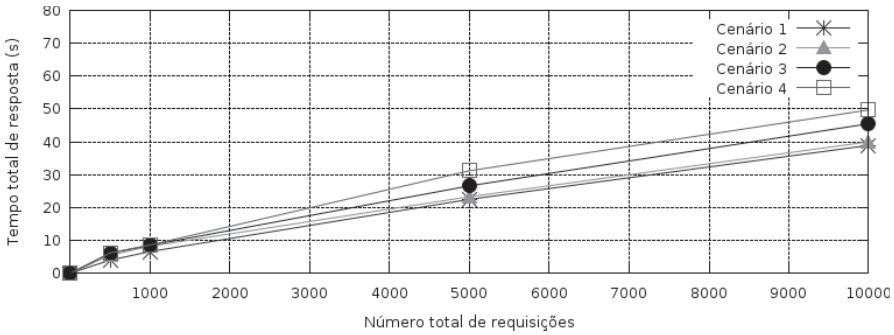

Figura 6. Tempos de resposta obtidos em quatro cenários diferentes.

O segundo cenário elicita a sobrecarga inerente à utilização da extensão, mesmo quando um serviço não replicado estiver sendo acessado. A sobrecarga é praticamente constante e diretamente proporcional ao tempo de inicialização do banco de dados.

O terceiro cenário mensura a sobrecarga gerada pelo acesso a um serviço replicado, mesmo que não ocorram falhas. O tempo para gerar as respostas é maior, e tem como fonte principal de sobrecarga a utilização do *AddressingTransportSender*.

O quarto cenário ilustra a quebra da réplica primária e os subsequentes redirecionamentos das requisições para a nova réplica primária. Os tempos de resposta dependem do momento em que a quebra ocorre, quanto mais cedo a quebra ocorrer, maior será a sobrecarga total, pois maior será o período em que a extensão fará redirecionamentos para a nova réplica primária (ver Fig. 4). Por este motivo, nos testes a réplica primária quebra após atender trinta requisições apenas, evidenciando a sobrecarga da transparência do *failover* ao cliente.

[10] A implementação atual do protótipo também trata falhas de quebra do servidor, com o tempo de *timeout* dependente do protocolo de transporte utilizado.

4. Trabalhos relacionados

A extensão proposta foi inspirada pelo elemento <WSG> do projeto FT-SOAP [Fang et al. 2007]. Entretanto, a utilização desse elemento, conforme proposto em [Fang et al. 2007] apresenta dois pontos negativos:

- Estende apenas a WSDL, logo, a lista de réplicas só pode ser definida na interface do serviço. Desta forma, mudanças futuras no grupo de réplicas não serão visíveis aos clientes que já tenham gerado o *stub* do serviço. Ao estender o WS-Addressing, é possível enviar novas versões do grupo de réplicas conforme mensagens são trocadas entre o fornecedor e o consumidor do serviço.
- Baseia-se em registros que seguem o padrão *Universal Description Discovery and Integration* (UDDI) como mecanismo de propagação de atualizações do grupo de réplicas. Acreditamos que a utilização de um registro UDDI como parte da infraestrutura de TF apresenta alguns problemas:
  - O padrão, por si só, até o presente momento, não é amplamente adotado. Outros padrões com os mesmos propósitos foram criados, como o ebXML, mas ainda sem grande representatividade [Michlmayr et al. 2007].
  - Em várias soluções a utilização de um registro UDDI não é necessária.
  - O registro se torna mais um ponto único de falha na arquitetura.

  Entretanto, ao se estender o WS-Addresing, remove-se a necessidade de utilização de UDDI, sem impor qualquer tipo de restrição a sua utilização.

Além do WS-Addressing, a especificação WS-ReliableMessaging [OASIS 2007] também engloba algumas funcionalidades interessantes que poderiam ser reutilizadas ou estendidas para alcançar os objetivos da extensão. Contudo, essa especificação não sofreu a mesma adoção que o WS-Addressing e tem o seu o escopo restrito a comunicações ponto a ponto. Um estudo sobre a combinação da WS-ReliableMessaging com a extensão proposta neste artigo pode ser um bom ponto inicial para trabalhos futuros.

4.1. Tolerância a falhas em Serviços Web

Há muitas propostas de TF em Serviços Web e a grande parte deles pode se beneficiar deste trabalho. Os FT-SOAP [Fang et al. 2007] e o trabalho de Wah [Wah 2006] utilizam um registro UDDI para atualizar a lista de réplicas. Caso esses trabalhos passem a utilizar o WS-Addressing estendido, seria possível remover o registro UDDI da solução.

O SmartWS [Junior et al. 2006], o WS-Replication [Salas et al. 2006] e o trabalho de Osrael et al. [Osrael et al. 2007] interagem com o cliente através de *proxies*[11], que são responsáveis pelas interações com as réplicas. Essa abordagem é transparente ao cliente, mas os *proxies* constituem pontos únicos de falha. Esta deficiência, todavia, pode ser superada por meio da replicação dos *proxies* ou do endereçamento das réplicas diretamente, utilizando a extensão.

O FTWeb [Santos et al. 2005] e o trabalho de Ye e Shen's [Ye and Shen 2005] utilizam replicação ativa. Assim, é necessário possuir uma lista atualizada de réplicas a cada interação com cliente. Porém, nenhum desses trabalhos detalha o método utilizado para especificar a lista de réplicas para o cliente. Ambos podem se beneficiar da extensão proposta por este trabalho e o mesmo pode ser dito a respeito de propostas baseadas

[11] O trabalho de Osrael et. al utiliza a réplica primária como *proxy*.

no modelo de *n-Versões* [Avizienis 1985], como o FT-GRID [Townend et al. 2005], que possuem a mesma necessidade.

O Thema [Merideth et al. 2005] tolera falhas bizantinas [Lamport et al. 1982] em Serviços Web. Entretanto, ele não pode ser considerado transparente ao cliente, pois este precisa utilizar a biblioteca do *framework* tanto no lado do cliente quanto do servidor.

Alwagait e Ghandeharizadeh [Alwagait and Ghandeharizadeh 2004] propõem uma arquitetura baseada em um servidor tratador de exceções, ao qual eles denominam *registro DeW*. Essa solução não é transparente ao cliente, pois o cliente precisa implementar a lógica responsável pelas interações com o registro DeW.

Por fim, o mais importante a ser notado é o fato de que a extensão pode ser utilizada por quase todos os trabalhos apresentados nesta Seção, conferindo, em alguns casos, até mesmo aumentos no nível de TF do *framework*, caso a lista de réplicas seja armazenada em meio permanente.

5. Conclusões e trabalhos futuros

A arquitetura de Serviços Web tem sofrido expansão significativa nos últimos anos e considerações em relação a sua dependabilidade são de importância fundamental. A proposta deste artigo fornece um maneira simples e genérica de criar uma ponte entre o cliente e a arquitetura de tolerância a falhas no lado do servidor de maneira transparente. É interessante notar que esta proposta também livra os desenvolvedores de técnicas de replicação e balanceamento de carga da tarefa de definir a maneira de referenciar as réplicas e de implementar o código correspondente.

Com os testes (Seção 3.6), o que foi apresentado de fato é a sobrecarga pura gerada pelo uso da extensão. Em um cenário real, com atrasos na rede e tempos de processamento mais longos, tanto no lado do cliente quanto do servidor, a porcentagem de tempo relativa ao processamento da extensão será reduzida.

Considerando as pequenas modificações feitas pela extensão no WS-Addressing, em várias situações os benefícios à dependabilidade dos serviços valem o custo. A possibilidade de enviar novas versões do grupo de réplicas juntamente com os cabeçalhos SOAP permite aumentar o nível de tolerância a falhas nas interações entre os Serviços Web. Além disso, uma grande qualidade da extensão é o fato de ser independente de framework de TF podendo, inclusive, trabalhar em conjunto com vários dos frameworks já propostos (ver Seção 4.1). Caso a extensão seja incorporada ao WS-Addressing ou se uma nova especificação de TF venha a incorporá-la, todas as soluções que estejam em conformidade com o padrão resultante serão capazes de se beneficiar do *failover* transparente ao cliente, incluindo-se neste conjunto os sistemas legados.

Por fim, algumas possibilidades de trabalhos futuros são: permitir a especificação de réplicas também nas interfaces WSDL, para que o cliente não precise realizar ao menos uma interação com o fornecedor do serviço para receber a lista de réplicas inicial; mensurar os impactos da utilização de outros meios de armazenamento no desempenho do protótipo; mensurar os efeitos colaterais que eventualmente possam ser inseridos pela extensão em outros padrões e vice-versa e implementar a replicação ativa no protótipo.

Referências

Alwagait, E. and Ghandeharizadeh, S. (2004). DeW: A Dependable Web Services Framework. In *RIDE '04: Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04)*, pages 111–118, Washington, DC, USA. IEEE Computer Society.

Avizienis, A. (1985). The N-Version Approach to Fault-Tolerant Software. *IEEE Transactions on Software Engineering*, SE-11(12):1491–1501.

Birman, K., van Renesse, R., and Vogels, W. (2004). Adding High Availability and Autonomic Behavior to Web Services. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 17–26, Washington, DC, USA. IEEE Computer Society.

Birman, K. P. (2006). The Untrustworthy Web Services Revolution. *Computer*, 39(2):98.

Budhiraja, N., Marzullo, K., Schneider, F. B., and Toueg, S. (1993). The Primary-Backup Approach. In *Distributed systems (2nd Ed.)*, pages 199–216, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Défago, X. and Schiper, A. (2004). Semi-passive replication and lazy consensus. *Journal of Parallel and Distributed Computing*, 64(12):1380–1398.

Fang, C.-L., Liang, D., Lin, F., and Lin, C.-C. (2007). Fault Tolerant Web Services. *Journal of Systems Architecture*, 53(1):21–38.

Foster, I., Parastatidis, S., Watson, P., and Mckeown, M. (2008). How do I model state?: Let me count the ways. *Communications Of The ACM*, 51(9):34–41.

Hansen, M. (2007). *Soa Using Java(Tm) Web Services*. Prentice Hall PTR, Upper Saddle River.

Huhns, M. N. and Singh, M. P. (2005). Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81.

Junior, J. G. R., Carmo, G. T. S., and Valente, M. T. O. (2006). Invocation of Replicated Web Services Using Smart Proxies. In *WebMedia '06: Proceedings of the 12th Brazilian symposium on Multimedia and the web*, pages 138–147, New York, NY, USA. ACM.

Lamport, L., Shostak, R., and Pease, M. (1982). The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401.

Merideth, M. G., Iyengar, A., Mikalsen, T., Tai, S., Rouvellou, I., and Narasimhan, P. (2005). Thema: Byzantine-Fault-Tolerant Middleware for Web-Service Applications. In *SRDS '05: Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pages 131–142, Washington, DC, USA. IEEE Computer Society.

Michlmayr, A., Rosenberg, F., Platzer, C., Treiber, M., and Dustdar, S. (2007). Towards Recovering the Broken SOA Triangle: A Software Engineering Perspective. In *IW-SOSWE '07: 2nd International Workshop on Service Oriented Software Engineering*, pages 22–28, New York, NY, USA. ACM.

OASIS (2007). Web Services Reliable Messaging (WS-ReliableMessaging). Disponível em: http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-os.html.

Osrael, J., Froihofer, L., Weghofer, M., and Goeschka, K. M. (2007). Axis2-based Replication Middleware for Web Services. *IEEE International Conference on Web Services (ICWS)*, pages 591–598.

Papazoglou, M. P. and Georgakopoulos, D. (2003). Service-oriented Computing. *Communications of the ACM*, 46(10):25–28.

Salas, J., Perez-Sorrosal, F., Patiño-Martínez, M., and Jiménez-Peris, R. (2006). WS-Replication: A Framework For Highly Available Web Services. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 357–366, New York, NY, USA. ACM Press.

Santos, G. T., Lung, L. C., and Montez, C. (2005). FTWeb: A Fault Tolerant Infrastructure for Web Services. In *EDOC '05: Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05)*, pages 95–105, Washington, DC, USA. IEEE Computer Society.

Schneider, F. B. (1990). Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys*, 22(4):299–319.

The Apache Software Foundation (2008). Apache Axis2/Java - Next Generation Web Services. Disponível em: http://ws.apache.org/axis2.

The Apache Software Foundation (2009). Apache Derby. Disponível em: http://db.apache.org/derby.

Townend, P., Groth, P., Looker, N., and Xu, J. (2005). FT-Grid: A Fault-Tolerance System for e-Science. In *Proceedings of the UK e-Science All Hands Meeting 2005*, Nottingham UK.

W3C (2006a). Web Services Addressing 1.0 - Core. Disponível em: http://www.w3.org/TR/ws-addr-core.

W3C (2006b). Web Services Addressing 1.0 - SOAP Binding. Disponível em: http://www.w3.org/TR/ws-addr-soap.

W3C (2006c). WS-Addressing 1.0 Latest XML Schema. Disponível em: http://www.w3.org/2006/03/addressing/ws-addr.xsd.

W3C (2007). Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. Disponível em: http://www.w3.org/TR/wsdl20.

Wah, C. P. (2006). Reliable Web Services by Fault Tolerant Techniques: Methodology, Experiment, Modeling and Evaluation. Technical report, The Chinese University of Hong Kong Shatin, N.T., Hong Kong.

Weerawarana, S., Curbera, F., Leymann, F., Storey, T., and Ferguson, D. F. (2005). *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall, Upper Saddle River.

Ye, X. and Shen, Y. (2005). A Middleware for Replicated Web Services. In *ICWS'05: Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, pages 631–638, Washington, DC, USA. IEEE Computer Society.