Aspectos de Replicação em Serviços Web *Stateful* através do Axis2

Igor Nogueira Santos[1], **Daniela Barreiro Claro**[1]

[1]Laboratório de Sistemas Distribuídos – LASID
Departamento de Ciência da Computação – Universidade Federal da Bahia(UFBA)
Salvador – BA – Brasil

igor@dcc.ufba.br, dclaro@ufa.br

Abstract. *Web services are being widely used in applications which require high reliability. Several specifications have been created in order to standardize the use of reliable mechanisms on Web services. Availability is one of the means to achieve high reliability. Several propositions have been made willing to replicate web services (WS) so as to improve its availability. WS replication, considering they are autonomous and heterogeneous, is even harder when there is state maintenance because web services are developed by different organizations in different ways. This paper evaluates correlated work and introduces a passive replication mechanism with state maintenance on Axis2.*

Resumo. *Os serviços Web estão cada vez mais sendo incorporados e utilizados em aplicações que demandam alta confiabilidade. Diversas especificações têm sido propostas com o intuito de padronizar a utilização dos aspectos de confiabilidade nos serviços Web. Uma das maneiras de garantir alta confiabilidade é através do aumento da disponibilidade dos serviços Web. Várias propostas têm sido feitas com o intuito de replicar serviços Web (WS) para garantir esta disponibilidade. A replicação destes serviços, considerando que eles são autônomos e heterogêneos, é particularmente difícil quando há manutenção de estado entre as requisições, já que eles são implementados de diferentes formas por diferentes organizações. Nesse contexto, este trabalho avalia trabalhos correlacionados e propõe um mecanismo de replicação passiva com garantia de estado para serviços Web no Axis2.*

1. Introdução

A crescente utilização de sistemas computacionais nas mais diversas atividades traz a necessidade de tolerância a eventuais falhas, sobretudo nos sistemas ditos críticos, visto que os mesmos são cada vez mais sofisticados, e os usuários precisam depositar grande confiança em seu funcionamento. Com a disseminação do uso de redes, sobretudo a Internet, uma nova dimensão foi adicionada ao problema, já que atualmente é expressivo o número de aplicações que operam em ambiente distribuído e precisam ser confiáveis. Dentre elas, destacam-se os serviços web, que são aplicações padronizadas, autônomas e heterogêneas operantes na Internet.

Nos últimos anos, várias especificações foram desenvolvidas com o intuito de prover maior confiabilidade aos serviços web. Segurança (WS-Security[Lawrence and Kaler 2004]) e envio confiável de mensagens (WS-ReliableMessaging [Iwasa et al. 2004]) são exemplos de especificações definidas especificamente para serviços Web (WS). Alta disponibilidade (*availability*), apesar de ser um dos principais elementos da confiabilidade de um sistema [Avizienis et al. 2004], é um dos aspectos ainda não padronizados para os WS. A principal forma de aumentar a disponibilidade de um sistema distribuído é replicá-lo em diferentes servidores, de forma que na falha de um deles, outro assuma seu lugar. Para o caso dos WS, essa abordagem é dificultada pela heterogeneidade de plataformas sobre as quais os serviços web são publicados, o que dificulta a determinação de um protocolo de replicação. Além disso, se a manutenção de estado for considerada (serviços *stateful*), modificar a implementação do serviço para que os mecanismos de sincronia de estado sejam implantados torna a replicação não transparente para o usuário.

A fim de contornar essa limitação, vários *middleware* de replicação têm sido propostos pela comunidade acadêmica, tais como: FT-SOAP [Chen et al. 2007], WS-Replication [Jiménez-Peris et al. 2006], Conectores de Tolerância a Falhas [Fabre and Salatge 2007] e a replicação híbrida adotada em [Froihofer et al. 2007]. Este trabalho, além de analisar os trabalhos correlacionados, apresenta a implementação de um mecanismo de replicação passiva para serviços *stateful* desenvolvido dentro da ferramenta Axis2.

O restante deste artigo está dividido da seguinte forma: a seção 2 define e apresenta as principais características dos serviços web. A seção 3 descreve as principais técnicas de replicação. A seção 4 ilustra a aplicação de protocolos de replicação em serviços web. A seção 5 detalha a implementação realizada. A seção 6 exibe os resultados encontrados nos experimentos. A seção 7 apresenta conclusões e considerações finais.

2. Características de um Serviço web

O conceito de serviço web vem ganhando força como um novo paradigma de desenvolvimento e utilização de software, deslocando o foco anteriormente voltado a objetos para uma abordagem baseada em documentos. SOA (*Service Oriented Architecture*) sintetiza essa nova arquitetura orientada a serviços ao propor a transformação dos artefatos pré-existentes em um conjunto de serviços interligados e acessíveis via rede.

A arquitetura SOA (figura 1) é composta por três elementos básicos: o Provedor do Serviço, o Solicitante do Serviço e o Repositório do Serviço. O provedor registra o

Figura 1. Arquitetura SOA

serviço em um repositório público e os solicitantes pesquisam por serviços que forneçam as operações que necessitam nesse repositório. Se algum serviço for encontrado, o repositório fornece o endereço do provedor para que o solicitante invoque o serviço de acordo com um contrato, que especifica as funcionalidades providas. Esse desacoplamento entre provedores e solicitantes é justificável por dar maior flexibilidade à arquitetura. Provedores e solicitantes podem ser construídos sobre infra-estruturas divergentes e, ainda assim, se comunicarem, porque a construção de serviços é baseada em tecnologias bem estabelecidas, o que possibilita a troca de informações independente das plataformas que servidores e consumidores utilizam internamente.

A invocação de serviços é realizada através de mensagens e se baseia nas seguintes especificações: WSDL (*Web Services Description Language*), que é linguagem de descrição de um serviço; SOAP (*Simple Object Access Protocol*), que define o formato das mensagens de um serviço e UDDI (*Universal Description, Discovery, and Integration*) que padroniza a especificação dos repositórios de serviços.

A confiança no funcionamento de um sistema (*Dependability*) é uma medida geral da qualidade provida pelo serviço, pois engloba outros aspectos como Integridade, Manutenabilidade e Disponibilidade [Avizienis et al. 2004]. Esses requisitos ganham mais importância à medida que as interações entre serviços web tornam-se mais automatizadas. Uma falha no funcionamento de um serviço pode não somente afetá-lo individualmente, como também a uma composição de serviços interligados. Dessa forma, técnicas de tolerância a falhas são cada vez mais utilizadas por tornar possível a continuidade do serviço Web na presença de falhas por meio da redundância de infra-estrutura.

3. Tolerância a Falhas

Várias abordagens foram desenvolvidas para tratar a possível presença de falhas nos sistemas. Dentre elas, destaca-se a Tolerância a Falhas. Um sistema tolerante a falhas reage de maneira bem definida à ocorrência de problemas, de forma que o mesmo continue provendo o serviço, ainda que na presença de falhas.

Existem vários mecanismos que podem ser empregados para tolerar falhas, cada um deles tem características peculiares que os tornam mais ou menos adequados a determinados contextos. Dessa forma, antes de escolher e implantar algum desses mecanismos, é essencial saber quais os comportamentos errados que um sistema pode apresentar e quais deles serão controlados pelas técnicas de tratamentos de defeitos escolhidas através da construção de classes de falha.

Sistemas distribuídos dependem, basicamente, de serviços de processamento e comunicação [Cristian 1991]. Técnicas de Tolerância a Falhas quando aplicadas nesse

contexto implementam redundância no hardware, replicando servidores em diferentes localidades e, por conseguinte, os programas que operam sobre eles. Porém, a utilização de replicação impõe a necessidade de manutenção de consistência de estado entre as réplicas, de forma que, na falha de um servidor, o serviço continue a ser provido por outra réplica a partir de um estado coerente com as atividades do sistema até o momento em que a falha ocorreu. A fim de garantir a sincronia de estado entre os membros de um grupo, técnicas de replicação aplicam um protocolo.

Os protocolos de replicação podem ser separados em dois grupos que se diferenciam pelas propriedades que respeitam: protocolos de replicação com processamento moderado (*parsimonious processing*) e protocolos de processamento redundante (*redundant processing*)[Défago and Schiper 2001].

3.1. Processamento Moderado

O principal protocolo que aplica o processamento moderado é a replicação passiva. Nesse protocolo, todas as requisições dos clientes são processadas por uma única réplica: a réplica primária. As outras réplicas, chamadas *backup*, recebem somente atualizações de estado do servidor primário.

Na falha da réplica primária, outra deve assumir seu lugar. Porém, durante o intervalo de tempo específico no qual o novo membro primário estiver sendo eleito, as novas requisições dos clientes serão perdidas, já que não existirá um servidor primário para processá-las. Nesse sentido, a replicação passiva clássica não é totalmente transparente ao usuário, de forma que o mesmo poderá necessitar re-enviar a mensagem, caso nenhuma resposta seja retornada. Outras abordagens de processamento moderado estão sendo utilizadas para contornar limitação de maneira eficiente, dentre elas, destacam-se a *coodination-cohort* e a replicação semi-passiva [Défago and Schiper 2001].

3.2. Processamento Redundante

No processamento redundante, todas as requisições são processadas por todas as réplicas de forma que seja garantido um tempo constante de resposta, ainda que na presença de falhas. Os principal protocolo desse grupo é a replicação ativa.

Na replicação ativa, todos os gerenciadores de réplicas agem como máquinas de estados que desempenham as mesmas atividades e que estão organizadas em grupos. Uma máquina de estados é formada por variáveis que encapsulam as informações de seu estado e de comandos que modificam esse estado ou produzem uma resposta [Schneider 1990]. Nesse modelo, cada comando consiste em um programa determinístico cuja execução é atômica em relação a outros comandos, de forma que a execução de uma máquina é equivalente a efetuar operações em uma ordem estrita.

A utilização desse modelo como protocolo de replicação só é possível se cada réplica começar no mesmo estado inicial e executar a mesma série de comandos na mesma ordem, de forma que cada máquina produzirá os mesmos resultados para entradas iguais. Assim, a replicação ativa requer que o processamento de mensagens seja determinístico, o que impede, por exemplo, a utilização desse mecanismo em aplicações *multithreading*.

Com a crescente utilização de serviços web em sistemas que demandam alta confiabilidade, a aplicação das técnicas de tolerância a falhas nesses componentes é cada

vez mais comum. A próxima seção discutirá como os protocolos de replicação têm sido aplicados em conjunto aos serviços web.

4. Tolerância a Falhas em Serviços Web

O requisito de disponibilidade, um dos principais componentes da confiabilidade de um sistema, é particularmente difícil de atingir no ambiente heterogêneo em que serviços web habitualmente operam. A tarefa de construir modelos de falha, por exemplo, é dificultada pela natureza da publicação dos serviços, já que um arquivo WSDL, obrigatoriamente, especifica apenas informações básicas necessárias à invocação de um serviço, de modo que informações sobre aspectos não-funcionais, tais como disponibilidade e desempenho, não são publicadas. Dessa forma, determinar quais as possíveis falhas que um determinado serviço pode vir a apresentar pode ser impossível sem as informações necessárias sobre cada serviço que o compõe. De fato, a disponibilidade de um serviço pode ser até menor que qualquer um dos componentes que lhe dão forma [Moser et al. 2007].

Apesar da evidente necessidade, nenhuma especificação oficial de confiabilidade no que tange à disponibilidade de serviços ainda foi desenvolvida. Isso ocorre, em grande parte, devido à dificuldade de estender as técnicas de replicação além das fronteiras de uma única organização, já que as tecnologias que elas usam no desenvolvimento de seus serviços são, possivelmente, não equivalentes. Nos últimos anos, vários *middleware* de replicação em web services foram desenvolvidos pela comunidade acadêmica com o propósito de contornar essa limitação. Dentre eles FT-SOAP [Chen et al. 2007], WS-Replication [Jiménez-Peris et al. 2006], Conectores de Tolerância a Falhas [Fabre and Salatge 2007] e a replicação híbrida adotada em [Froihofer et al. 2007] serão analisados nas próximas seções.

4.1. FT-SOAP

FT-SOAP é um *middleware* de replicação passiva. Os componentes básicos dessa especificação são: o **Gerenciamento de Falhas** que é utilizado para realizar o monitoramento de estado das réplicas, o **Mecanismo de Log e Recuperação**, responsável por fazer o log das invocações, de forma que elas não sejam perdidas na falha do gerenciador primário, e o **Gerenciador de Replicação**, que tem a função de monitorar e constituir os grupos de réplicas.

O uso de componentes centralizados adiciona novos pontos de falhas ao sistema, de forma que até os componentes do próprio *middleware* devem ser replicados.

O mecanismo de replicação desenvolvido não segue esta linha. Em sua implementação, cada réplica funciona como um mecanismo de replicação independente, evitando a limitação encontrada na proposta FT-SOAP.

4.2. WS-Replication

WS-Replication implementa replicação ativa no contexto dos serviços web. O protocolo de replicação aplicado é baseado no componente WS-Multicast, que utiliza tecnologias básicas dos serviços web (SOAP e WSDL) para promover sincronia entre as réplicas. Ao utilizar o WS-Multicast em sua estrutura, WS-Replication mantêm na estrutura de replicação a independência de plataformas inerente à arquitetura SOA, escalando o protocolo a operar diretamente sobre os serviços na Internet. Além disso, como nenhuma

modificação é imposta à implementação dos serviços, a replicação é totalmente transparente aos usuários.

O mecanismo de replicação implementado, de maneira semelhante ao WS-Replication também é totalmente transparente aos usuários.

4.3. Conectores

Este *middleware* parte do princípio de que criar serviços web confiáveis é difícil porque eles geralmente dependem de outros serviços que não são confiáveis, de forma que é necessário encontrar meios externos ao serviço para prover mecanismos adicionais de tolerância a falhas. A idéia básica dessa infra-estrutura é realizar a comunicação com os serviços web através de conectores que implementam características de replicação.

A noção de conector é baseada no conceito de ADLs (*Architecture Description Language*), que permite customizar interações entre componentes. Um conector específico de tolerância a falhas é utilizado para interceptar invocações a um serviço Web e realizar ações características de mecanismos de replicação, funcionando como um componente intermediário entre um serviço web construído de forma não-confiável e um cliente que representa uma aplicação SOA crítica.

A presença de conectores encapsulando cada réplica de um serviço torna esse *middleware* bastante flexível, de maneira que é possível configurá-lo para aplicar replicação passiva ou ativa. Na replicação passiva, porém, para que haja manutenção de sincronia de estado entre as réplicas, os serviços devem implementar funções de manipulação de estado e publicá-las conjuntamente às funcionalidades propriamente ditas.

Diferentemente da proposta deste artigo, no mecanismo desenvolvido e acoplado à plataforma Axis, nenhuma alteração é imposta à codificação dos serviços replicados.

4.4. Modelo Híbrido de Replicação

Esse *middleware* de replicação considera serviços publicados sobre infra-estruturas homogêneas, de forma que é possível adotar medidas de replicação comumente utilizadas em objetos distribuídos. O modelo de replicação adotado tem o seu funcionamento básico baseado nos protocolos de replicação passiva e ativa.

Da passiva, ele herda a característica de todas as mensagens serem enviadas somente ao gerenciador primário. Da ativa, ele implementa o processamento redundante.

O mecanismo de interceptação deste último *middleware* é implementado na *engine* do Axis, de forma que o protocolo de replicação não precisa ser implementado nos serviços em si. O mecanismo, porém, só é aplicado sobre serviços publicados através dessa ferramenta. Apesar desta limitação, a plataforma Axis é uma das principais formas de publicação de serviços Web da atualidade, o que facilita a utilização deste mecanismo.

Os mecanismos de interceptação e replicação deste *middleware* serviram de base para o desenvolvimento da proposta do presente trabalho. O principal diferencial foi a incorporação da replicação passiva clássica para serviços e com isso uma diminuição no processamento de cada réplica. A implementação desta replicação passiva será detalhada na próxima seção.

5. Implementação

A fim de testar a aplicação de mecanismos de tolerância a falhas em serviços web, foi desenvolvido um protótipo de replicação passiva em serviços *stateful* levando em consideração a manutenção de consistência de estado entre as réplicas. Os serviços testados foram desenvolvidos em Axis2 [Jayasinghe 2008], e as garantias de comunicação em grupo entre as réplicas foram implementadas através da ferramenta (*toolkit*) JGroups [Ban 2008].

JGroups é um *toolkit* Java para comunicação *multicast* confiável. Sua arquitetura consiste de três partes: uma API Canal (*Channel*) que provê as funcionalidades básicas de acesso e criação de grupos de réplicas; Blocos de Implementação (*Building Blocks*) que provêm uma abstração mais refinada de utilização do canal; e Pilha de Protocolos (*Protocol Stack*) que é formada pelos componentes que implementam as garantias de confiabilidade e ordenação da entrega de mensagens.

Já o Axis2 é um *engine* de implementação da especificação SOAP. Seu funcionamento consiste, basicamente, em enviar e receber mensagens XML. Para tanto, a arquitetura do Axis2 mantém dois fluxos que executam essas atividades. Esses fluxos são implementados pelo mecanismo Axis (*Axis Engine*) através de dois métodos: enviar (*send*) e receber (*receive*). Os dois fluxos são chamados, respectivamente, Fluxo de Entrada (*InFlow*) e Fluxo de Saída(*OutFlow*).

Nesse modelo, cada fluxo é divido em fases (*phases*) e cada fase é formada de *handlers* que agem como interceptadores processando partes da mensagem e provendo qualidade de serviço. Quando uma mensagem SOAP está sendo processada, os handlers registrados nas fases são executados. Dessa forma, para adicionar funcionalidades ao processamento, é necessário registrar novos *handlers* nas fases de execução pré-existentes na arquitetura, ou em novas fases criadas pelo usuário.

Para efeitos de replicação, foi adicionada uma nova fase ao fluxo de entrada intitulada ”FaseReplicacaoPassiva”. O funcionamento do protótipo compreende os seguintes passos no fluxo de entrada:

1. A invocação do cliente chega à interface de transporte da réplica primária e é encaminhada às fases posteriores do fluxo de entrada. Durante as fases, cada handler recebe um objeto do tipo ”contexto de mensagem”(*MessageContext*) e adiciona ou modifica informações no mesmo. Basicamente, um contexto de uma mensagem contém informações sobre a configuração da *engine* Axis, da sessão e o envelope SOAP em si;
2. Ao atingir a fase de replicação, o contexto é serializado e não somente a invocação, o que difere da solução proposta em (FROIHOFER et al., 2007). Essa modificação é feita porque a fase de replicação foi inserida imediatamente antes da fase de processamento, de forma que o contexto ao qual o *handler* implementado tem acesso está completo e pronto para dar início ao processamento da requisição, evitando o reprocessamento da invocação em cada réplica;
3. Serializado o contexto, a réplica primária processa a requisição e a mensagem de atualização é enviada a todas as réplicas backup, exceto à própria réplica primária, que já possui a informação construída. O envio para as outras réplicas é feito utilizando o bloco de implementação *RPCDispatcher* que permite a invocação remota de métodos;

4. No recebimento do contexto, as réplicas executam o algoritmo 1.

Algoritmo 1 Recebimento do contexto pelas réplicas *backup*

```
1. String nomeServico = contexto.get(0);
2. String contextoSerializado = contexto.get(1);
3. historicoContexto.put(nomeServico, contextoSerializado);
```

A mensagem de atualização para a réplica contém dois campos, sendo o primeiro deles o nome do serviço e o segundo o contexto serializado. As linhas 1 e 2 demonstram esse aspecto. Na linha 3, o contexto serializado é armazenado em um objeto *hashmap* mantido em cada réplica. É possível notar que, a cada requisição a um determinado serviço, seu histórico é sobrescrito com o novo contexto. Isto é pertinente porque as informações de sessão são mantidas no contexto do Axis2, de forma que o mesmo acumula todas as modificações feitas aos estados da sessão. Para suportar persistência de estado em dispositivos secundários como arquivos ou bancos de dados, o histórico pode ser facilmente estendido de forma a armazenar uma lista de contextos para cada serviço.

Dessa forma, em cada réplica, antes da execução do fluxo acima, o histórico é verificado à procura de entradas pelo nome do serviço. Como a réplica primária nunca recebe as mensagens de atualização que envia, necessariamente, o seu histórico sempre estará vazio. Na ocorrência de falhas, quando o novo primário é eleito e recebe uma invocação, ele busca no histórico algum contexto associado ao nome do serviço. Caso encontre, esse contexto é reconstruído e processado antes que a nova requisição do usuário seja atendida.

Caso o gerenciador primário falhe, a eleição do novo líder é feita através da nova lista de membros criada pelo JGroups. Como essa lista é mantida pelo grupo, ainda que mais de um servidor falhe ao mesmo tempo, os membros restantes atualizarão e difundirão a lista que contém o novo membro primário. Essa característica, somada ao histórico armazenado em cada réplica, faz que cada membro do grupo de replicação funcione como um mecanismo de replicação em si, dispensando a criação de componentes centralizados e a conseqüente adição de novos pontos de falha ao modelo.

O tempo de recuperação de estado é proporcional ao número de requisições que devem ser processadas antes da execução da nova invocação. Se o contexto for atualizado a cada invocação, então esse tempo é igual a uma única reconstrução de contexto. Caso o histórico seja modificado a fim de manter todo o conjunto de requisições, então esse valor pode ser bem maior, pois cada contexto presente no histórico deve ser reconstruído e processado antes da nova invocação.

A próxima seção apresentará os principais resultados obtidos nos testes realizados.

6. Experimentos

Com o intuito de avaliar o funcionamento do protótipo desenvolvido foram realizados testes de desempenho para determinar o *overhead* da utilização do mecanismo no que tange ao trabalho implementado e a utilização da comunicação em grupo. Também foram validadas a manutenção e consistência de estado entre as réplicas.

Os testes foram realizados de forma a considerar somente o tempo de execução dos protótipos sem influência da latência de redes. Para tanto, foram criadas quatro instâncias

independentes do servidor Tomcat, versão 6.0.18, em um PC Pentium D 2.80 GHz, 1.5 GB de RAM, *Microsoft Windows XP Professional* 2002 *Service Pack 2*. Cada instância Tomcat publica uma instância Axis2 1.4.1. Já a versão do JGroups toolkit é a 2.6.4 e a pilha padrão de protocolos foi utilizada.

Para a realização dos testes foi desenvolvido um serviço web simples, que mantém um vetor de strings na sessão. As operações básicas oferecidas pelo serviço estão listadas a seguir:

1. Criar sessão: Método responsável por criar o vetor na sessão;
2. Adicionar elemento: Adiciona um elemento à sessão;
3. Listar número de elementos: Retorna a quantidade de itens na sessão;
4. Listar elementos: Retorna os elementos na sessão em uma string.

Nos experimentos, cada requisição adicionou um elemento ao vetor da forma ”elemento i”, onde i cresce de acordo com o número da requisição, variando, dessa forma, de ”elemento 0”a ”elemento 2999”.

Esse serviço foi publicado em cada instância Axis2 instalada no computador. Para o teste de desempenho, foi feita uma série de 3000 requisições à réplica primária e calculados a média e o desvio padrão para os seguintes cenários:

1. Um serviço, sem qualquer adição de replicação;
2. Replicação com 1, 2, 3 e 4 réplicas.

Para avaliar o *overhead* do mecanismo em si, foram aplicados testes no protótipo na presença de uma única réplica, de forma a evitar o *overhead* da comunicação em grupo.A média de execução encontrada para o cenário sem replicação foi de 46,1713 $\pm$ 3,4321. Para a replicação passiva foi de 50,9367 $\pm$ 5,7397, impondo um *overhead* de 10.3%.

O aumento no tempo de execução imposto pelo mecanismo é justificável porque a adição de uma fase representa um aumento na execução geral de mensagens, pois cada fase adicionada ao fluxo representa mais um passo de execução na *engine* do Axis2. Outro aspecto importante é a serialização do contexto. À medida que as invocações são feitas e mais informações são adicionadas à sessão, o contexto aumenta de tamanho de forma que a sua serialização representa um dos principais aspectos de consumo de tempo no modelo proposto e explica o desvio padrão relativamente alto, já que as últimas requisições da série demandaram um maior esforço de serialização em relação às requisições iniciais.

Para o teste realizado com várias réplicas, o resultado obtido está ilustrado no gráfico da figura 2. Com a adição de réplicas, o mecanismo de comunicação em grupo é efetivamente utilizado, de forma que o tempo de execução de cada mensagem, além dos fatores considerados anteriormente, é acrescido pelo envio das mensagens de atualização de estado para cada réplica, além do mecanismo de monitoração de estado dos membros do grupo, que cresce conjuntamente ao número de réplicas. Nesse modelo, o *overhead* máximo foi atingido na presença de 4 réplicas, ou seja, 1 réplica primária e 3 réplicas *backup*. O tempo médio para essa configuração foi de 60,0057 $\pm$ 5,95 impondo um aumento máximo de 28,3% em relação à execução sem replicação, e de 18,9% em relação ao esquema passivo com 1 réplica somente.

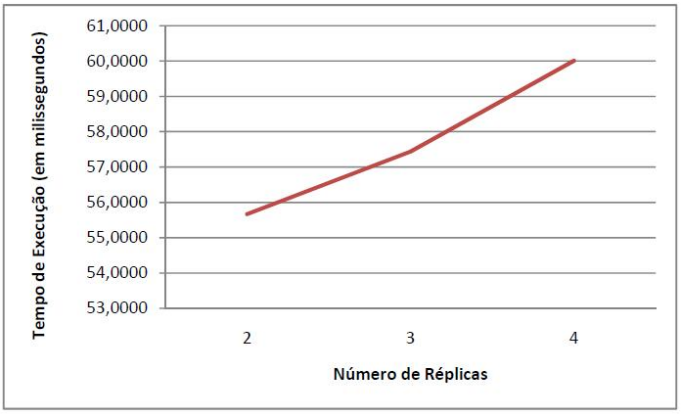

Figura 2. Comportamento da replicação passiva na presença de réplicas.

Para o teste de manutenção de estado, a análise foi feita fazendo a réplica primária falhar continuamente e aplicando a operação de listar o número de itens no novo primário eleito. Para o teste de consistência, a listagem de elementos foi invocada em cada novo primário e armazenada para posterior comparação.

O teste de manutenção de estado funcionou dentro do esperado, de forma que cada réplica apresentou 3000 itens em seus vetores após a falha do servidor primário.

O teste de consistência também foi bem sucedido. Para comparar os estados, após as invocações, a operação de listagem de conteúdo foi invocada em cada réplica e seus resultados armazenados para que pudessem ser comparados através da API Java.

Os testes demonstraram que o maior *overhead* da replicação foi a comunicação em grupo. A pilha de protocolo utilizada nos testes foi a pilha padrão do JGroups, de forma que é necessário estudá-la melhor e customizá-la a fim de apresentar um desempenho mais satisfatório. Além disso, testes mais aprofundados devem ser realizados a fim de avaliar o desempenho do mecanismo em ambientes reais a fim de avaliar a viabilidade de sua utilização nesses contextos.

Como todas as modificações foram feitas na arquitetura Axis2, nenhuma mudança foi necessária diretamente no serviço web, de forma que, nesse aspecto, este trabalho garante que serviços Web autônomos e heterogêneos sejam replicados de uma maneira transparente.

7. Conclusão

A despeito da inexistência de especificação formal, técnicas de tolerância a falhas vêm sendo cada vez mais aplicadas em serviços web (WS) dada a tendência dos mesmos publicarem funcionalidades cada vez mais críticas. A disponibilidade dos serviços, um dos componentes principais da confiabilidade de um sistema, tem sido fruto de pesquisa na comunidade acadêmica que, nos últimos anos, tem proposto diversas propostas de replicação para os WS.

O mecanismo de replicação desenvolvido mostrou-se funcional, aumentando a disponibilidade dos serviços e mantendo a consistência de estado entre os servidores sem a necessidade de modificar a implementação dos serviços para que a sincronia de estado

fosse alcançada.

Dessa forma, conclui-se que a replicação em serviços é praticável inclusive no que se refere à manutenção e consistência de estado entre as réplicas, evidenciando que o principal obstáculo encontrado para a construção de uma especificação oficial de replicação para serviços é a heterogeneidade de plataformas em que os mesmos são publicados, além do fato deles serem providos por diferentes instituições com diferentes políticas de manutenção de software.

Referências

Avizienis, A., Landwher, C., and Laprie, J.-C. (2004). Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing.

Ban, B. (2008). Reliable multicasting with the jgroups toolkit. http://www.jgroups.org/manual/html/index.html. Último acesso em: 26 de novembro de 2008.

Chen, C., Fang, C.-L., and Liang, D. (2007). Ft-soap: A fault-tolerant web service. Journal of Systems Architecture: the EUROMICRO Journal.

Cristian, F. (1991). Understanding fault tolerant distributed systems. Communication of ACM.

Défago, X. and Schiper, A. (2001). Specification of replication techniques, semi-passive replication, and lazy consensus. Academic Press.

Fabre, J.-C. and Salatge, N. (2007). Fault tolerance connectors for unreliable web services. 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks.

Froihofer, L., Goeschka, K., Osrael, J., and Weghofer, M. (2007). Axis2-based replication middleware for web services. IEEE International Conference on Web Services (ICWS 2007).

Iwasa, K., Durand, J., Rutt, T., Peel, M., Kunisetty, S., and Bunting, D. (2004). Web services reliable messaging tc ws-reliability 1.1. OASIS Open 2003-2004.

Jayasinghe, D. (2008). Quickstart apache axis2. Birmingham: Packt Publishing.

Jiménez-Peris, R., Patino-Martinez, M., Pérez-Sorrosal, F., and Salas, J. (2006). Ws-replication: A framework for highly available web services. WWW 2006 - International World Wide Web Conference, Edinburgh,Scotland.

Lawrence, K. and Kaler, C. (2004). Web services security: Soap message security 1.1. http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-pr-SOAPMessageSecurity-01.htm.Último acesso em: 03 de março de 2009.

Moser, L. E., Smith, P., and Zhao, W. (2007). Building dependable and secure web services. JOURNAL OF SOFTWARE, VOL. 2, NO. 1.

Schneider, F. B. (1990). Replication management using the state-machine approach. ACM Press/Addison-Wesley Publishing Co.