UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATISTÍSTICA

André Albino Pereira

ESCALABILIDADE DE SERVIÇOS EM NUVEM COM GERÊNCIA DE IDENTIDADE FEDERADA

Florianópolis

2014

André Albino Pereira

ESCALABILIDADE DE SERVIÇOS EM NUVEM COM GERÊNCIA DE IDENTIDADE FEDERADA

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do Grau de Mestre em Ciência da Computação. Orientador: Prof. Dr. João Bosco Mangueira Sobral

Florianópolis

2014

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina

A ficha catalográfica é confeccionada pela Biblioteca Central.

Tamanho: 7cm x 12 cm

Fonte: Times New Roman 9,5

Maiores informações em:
http://www.bu.ufsc.br/design/Catalogacao.html

André Albino Pereira

ESCALABILIDADE DE SERVIÇOS EM NUVEM COM GERÊNCIA DE IDENTIDADE FEDERADA

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Ciência da Computação", e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Florianópolis, 24 de fevereiro 2014.

Prof. Dr. Ronaldo dos Santos Mello
Coordenador do Curso

Prof. Dr. João Bosco Mangueira Sobral
Orientador

Banca Examinadora:

Prof. Dr. Altair Santin
Pontifícia Universidade Católica do Paraná

Prof. Dr. Frank Augusto Siqueira
Universidade Federal de Santa Catarina

Prof. Dr. Mário Dantas
Universidade Federal de Santa Catarina

Dedico este trabalho primeiramente a Deus, pela saúde, fé e perseverança que tem me proporcionado. A minha esposa, Ariana Marafon, pelo apoio e carinho em todos os momentos. Aos meus pais, Marcos Pereira e Geuza Maria Albino Pereira, pela educação e zelo que me permitiram chegar aonde estou.

AGRADECIMENTOS

Agradecimento especial para meu orientador João Bosco Mangueira Sobral, pela compreensão durante os momentos difíceis e reconhecimento nos momentos de recompensa. Por fim, agradeço à Softplan, empresa que contribuiu na construção da minha vida profissional e propiciou a realização deste trabalho.

RESUMO

Com o amadurecimento de abordagens de autorização multi–inquilino e gerenciamento de identidade federada para computação em nuvem, a provisão de serviços utilizando esse paradigma permite maximizar a eficiência para organizações em que o controle de acesso é imprescindível. No entanto, no que tange o suporte à escalabilidade, principalmente horizontal, algumas características dessas abordagens baseadas em protocolos de autenticação central apresentam problemas. Este trabalho visa mitigar esses problemas provendo uma solução baseada em uma adaptação do mecanismo *sticky-session* para a arquitetura *Shibboleth* utilizando *JASIG CAS*. Essa alternativa, em comparação com a abordagem recomendada baseada em distribuição de memória, mostrou mais eficiência e redução da complexidade da infraestrutura, exigindo um percentual de menos *58%* de recursos computacionais e aprimorando o *throughput* (requisições por segundo) em *11%*.
Palavras-chave: Controle de Acesso, Autenticação e Gerenciamento de Identidade, Segurança em Computação em Nuvem

ABSTRACT

As multi-tenant authorization and federated identity management systems for cloud computing matures, the provisioning of services using this paradigm allows maximum efficiency on business that requires access control. However, regarding scalability support, mainly horizontal, some characteristics of those approaches based on central authentication protocols are problematic. The objective of this work is to address these issues by providing an adapted sticky-session mechanism for a *Shibboleth* architecture using *JASIG CAS*. This alternative, compared with the recommended shared memory approach, shown improved efficiency and less overall infrastructure complexity, as well as demanding less *58%* of computational resources and improving *throughput* (requests per second) by *11%*.
Keywords: Access control, Authentication and Identity Management, Cloud Computing Security

LISTA DE FIGURAS

Figura 1 Infraestrutura tradicional para disponibilização de aplicações *Web* 30
Figura 2 Fluxo de comunicação com serviços *Web* utilizando *sticky-session* 32
Figura 3 Federação de identidades entre múltiplas organizações . 33
Figura 4 Elementos de uma infraestrutura de gerenciamento de identidade com *Shibboleth* 34
Figura 5 Fluxo de Autenticação em uma Infraestrutura com *CAS* 37
Figura 6 Fluxo da operação de *SSOUT* fornecido em uma infraestrutura com *CAS* 37
Figura 7 Simulação de falha no processo de autenticação em serviços escalados horizontalmente com computação em nuvem 38
Figura 8 Falha no processo de *SSOUT* em um contexto de escalabilidade horizontal na computação em nuvem 39
Figura 9 Serviços utilizando *Terracotta* para distribuição de memória 39
Figura 10 Visão geral da adaptação dos elementos para aplicar *sticky-session* com o sufixo do *ticket* gerado pelo *CAS Server* 44
Figura 11 Representação em diagrama de sequência *UML* da adaptação dos elementos para aplicar *sticky-session* com o sufixo do *ticket* gerado pelo *CAS Server* 44
Figura 12 Visão geral da do processo de *logout* com *sticky-session* adaptado 45
Figura 13 Representação em diagrama de sequência *UML* do processo de *logout* com *sticky-session* adaptado 46
Figura 14 Cenário para simular as alternativas para agregar suporte de escalabilidade horizontal em infraestruturas com *CAS* 46
Figura 15 Arquivo de configuração do *Apache Web Server* 48
Figura 16 Configurações no *CAS Server* para distribuição dos *tickets* gerados no fluxo de autenticação 51
Figura 17 Configurações no *CAS Server* para agregar identificador do nodo no *ticket* do fluxo de autenticação 52
Figura 18 Arquivo de configuração *applicationContext.xml* utilizado no serviço com acesso restrito 53
Figura 19 Fluxo de funcionamento do monitor de *CPU* e memória

desenvolvido em *Java* 54
Figura 20 *Apache* JMeter com *GUI* para configuração e execução de roteiro de testes 54
Figura 21 Gráfico comparativo da média de *throughput* 55
Figura 22 Gráfico comparativo da porcentagem de falhas por operação de cada usuário 55
Figura 23 Gráfico comparativo sobre o uso de CPU da *VM2* entre as abordagens 56
Figura 24 Gráfico comparativo sobre o uso de memória da *VM2* entre as abordagens 57
Figura 25 Gráfico comparativo sobre o uso de CPU da *VM3* entre as abordagens 57
Figura 26 Gráfico comparativo sobre o uso de memória da *VM3* entre as abordagens 58

LISTA DE TABELAS

Tabela 1 Distribuição de recursos de processamento para cada *VM* 47
Tabela 2 Comparativo das abordagens 58

LISTA DE ABREVIATURAS E SIGLAS

SSO Single Sign-On 25
CAS Central Authentication Service 25
SaaS Software-as-a-Service 29
PaaS Platform-as-a-Service 29
DNS Domain Name System 30
PI Provedor de Identidade 32
PS Provedor de Serviço 33
Shib SP Shibboleth Service Provider 34
IIS Microsoft Internet Information Services 34
Shib IdP Shibboleth Identity Provider 35
BC Balanceador de Carga 37
REST Representational State Transfer 40
IaaS Infra-structure-as-a-Service 41
SSOUT Single-Sign-Out 41
DoS Denial of Service 45
VM Virtual Machine 46
SSL/TLS Secure Socket Layer/Transport Layer Security 49
DHT Distributed Hash Table 60
P2P Peer-to-peer 60

SUMÁRIO

1 INTRODUÇÃO 25
1.1 CONTEXTUALIZAÇÃO DO TRABALHO 25
1.2 MOTIVAÇÃO E JUSTIFICATIVA 25
1.3 OBJETIVOS 26
1.3.1 Objetivo Geral 26
1.3.2 Objetivos Específicos 26
1.4 ESTRUTURA DO TRABALHO 27
2 FUNDAMENTAÇÃO TEÓRICA 29
2.1 COMPUTAÇÃO EM NUVEM 29
2.2 ARQUITETURA COM APLICAÇÕES WEB EM CONTEXTO DE COMPUTAÇÃO EM NUVEM 30
2.3 STICKY-SESSION 31
2.4 GERENCIAMENTO DE IDENTIDADE COMO SERVIÇO . 31
2.5 SOLUÇÃO PARA GERENCIAMENTO DE IDENTIDADE FEDERADA COM SHIBBOLETH 33
2.6 FLUXO DE AUTENTICAÇÃO COM *CAS* 35
2.7 SUPORTE À ESCALABILIDADE HORIZONTAL 36
2.8 MEMÓRIA DISTRIBUÍDA COM TERRACOTTA 39
2.9 ANÁLISE DE TRABALHOS CORRELATOS 40
3 PROPOSTA COM STICKY-SESSION ADAPTADO . 43
3.1 DETALHAMENTO DA PROPOSTA 43
3.2 RESULTADOS EXPERIMENTAIS 46
3.2.1 Configuração do Balanceamento de Carga com Apache Web Server 48
3.2.2 Instâncias do *CAS Server* 49
3.2.2.1 Configuração do *CAS Server* para o cenário com *Terracotta* 50
3.2.2.2 Configuração do *CAS Server* para o cenário com *Sticky-session* 50
3.2.3 Instâncias do serviço (recurso a ser acessado) 50
3.2.4 Preparação dos Monitores dos Recursos de *Hardware* 51
3.2.5 Configuração do *JMeter* para execução dos testes . . 52
3.3 ANÁLISE COMPARATIVA DOS RESULTADOS 53
4 CONCLUSÕES E TRABALHOS FUTUROS 59
REFERÊNCIAS 61

1 INTRODUÇÃO

Este trabalho contribui no amadurecimento de uma solução de gerenciamento de identidade federada para computação em nuvem. Inicialmente proposta por (LEANDRO et al., 2012), essa solução fornece uma arquitetura para que múltiplas organizações realizem autenticação e controle de acesso de maneira centralizada.

1.1 CONTEXTUALIZAÇÃO DO TRABALHO

A provisão de serviços utilizando computação em nuvem estabelece que recursos computacionais possam ser contratados conforme a demanda, aprimorando a eficiência operacional e reduzindo custos para a organização que adotar esse paradigma (ZHOU et al., 2010).

No entanto, em cenários envolvendo dados sensíveis, é necessário que mecanismos de gerenciamento de identidade e acesso evoluam para que a computação em nuvem se torne uma plataforma mais confiável, podendo ser plenamente adotada pelas organizações (OLDEN, 2011).

Os sistemas de gerenciamento de identidade e acesso devem suportar a cooperação entre organizações, principalmente para fornecer recursos como *SSO* (*Single Sign-On*). As identidades utilizadas nesse contexto são denominadas *identidades federadas* (CHADWICK, 2009).

Tecnologias como *Shibboleth* (CANTOR et al., 2005) e *CAS* (*Central Authentication Service*) (COMMUNITY, 2012) implementam identidades federadas, e (LEANDRO et al., 2012) apresenta a concepção de uma infraestrutura de identidade federada para serviços na nuvem.

1.2 MOTIVAÇÃO E JUSTIFICATIVA

Conforme indicado por (ARMBRUST et al., 2010), independentemente se os recursos em nuvem contratados terem baixo nível de abstração como no *Amazon EC2*, ou um nível mais alto como a *AppEngine*, as tecnologias em hardware e software devem focar na escalabilidade horizontal em detrimento de um nodo central com alto desempenho.

Escalabilidade horizontal nesse contexto implica que a solução deve permitir que seus componentes de software possam ser clusterizados em múltiplos servidores de maneira transparente. Especificamente para soluções com *Shibboleth* e *CAS*, a escalabilidade horizontal pode

ser realizada através da clusterização de seus provedores de identidade e serviço. Como o processo de autenticação nessas soluções precisa estabelecer uma sessão HTTP com o usuário, os dados dessa sessão armazenados no servidor precisam ser distribuídos entre todos os nodos do cluster. Sendo assim, (JOIE; CANTOR, 2012) e (BATTAGLIA; SAVAGE, 2012b) recomendam o uso de uma plataforma de memória distribuída. Contudo, essa técnica tende a aumentar consideravelmente a complexidade da solução e o custo de infraestrutura necessária.

A motivação deste trabalho é de agregar valor na solução proposta por (LEANDRO et al., 2012), garantindo escalabilidade no contexto de computação em nuvem com maior eficiência possivel, ou seja, custo reduzido para implantação e manutenção da infraestrutura.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

O objetivo deste trabalho é de amadurecer uma solução de gerenciamento de identidades federadas, para que seus componentes de software sejam horizontalmente escaláveis em um contexto de computação em núvel com maior eficiência possível. Para concretizar esse objetivo é apresentada uma alternativa para o gerenciamento de sessões de autenticação e acesso, baseada em uma adaptação do conceito de afinidade de sessão, também conhecida como *sticky-session* (STECCA; BAZZUCCO; MARESCA, 2011).

1.3.2 Objetivos Específicos

Além de contribuir com a eficiência no gerenciamento de identidades federas em um contexto de computação em nuvem, identificam-se os seguintes objetivos:

- Avaliar o estado atual das soluções para gerenciamento de identidade e a infraestrutura necessária para um ambiente de computação em nuvem.
- Realizar uma análise comparativa das abordagens para implantação da solução proposta por (LEANDRO et al., 2012).
- Permitir que o estudo possa ser usado como base para organizações definirem a infraestrutura mais adequada para resolver

o problema de controle de acesso em um contexto de computação em nuvem.

1.4 ESTRUTURA DO TRABALHO

Esta dissertação está estruturada da seguinte forma: Capítulo 2 fundamenta os aspectos chave associados a essa pesquisa, como computação em nuvem e gerenciamento de identidade como serviço; Seção 2.5 elenca as tecnologias envolvidas em uma plataforma de gerenciamento de identidade federada e seu suporte à escalabilidade horizontal; Seção 2.9 descreve os trabalhos correlatos; Capítulo 3 detalha uma nova alternativa para agregar escalabilidade aos componentes de uma infraestrutura de gerenciamento de identidades federadas para computação em nuvem; Seção 3.3 mostra os resultados e a análise comparativa da alternativa apresentada com uma existente; Por fim no Capítulo 4 as conclusões e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo são fundamentados os aspectos básicos associados às contribuições promovidas por este trabalho.

2.1 COMPUTAÇÃO EM NUVEM

Computação em nuvem é um modelo que permite o acesso a recursos computacionais sob demanda como servidores, aplicações, entre outros. Esses recursos devem ser provisionados rapidamente com pouco esforço do provedor do serviço (MELL; GRANCE, 2011). Para isso, as tecnologias utilizadas nesses ambientes devem ser horizontalmente escaláveis (ARMBRUST et al., 2010). Escalabilidade horizontal implica na conexão de múltiplas entidades de software ou hardware, como servidores, para que possam trabalhar como uma única unidade lógica. No caso de servidores, a adição de servidores aprimoraria o desempenho dessa unidade lógica, utilizando clusterização e balanceamento de carga. Por outro lado, a escalabilidade vertical corresponde ao aumento do poder de processamento de uma única entidade, como por exemplo adicionando memória em um servidor. Existem três tipos de modelos de serviço que podem ser oferecidos por provedores de computação em nuvem (MELL; GRANCE, 2011):

- *SaaS* (*Software-as-a-Service*): Provisão de aplicações implantadas na nuvem, da qual o contratante não participa da configuração da infraestrutura interna.

- *PaaS* (*Platform-as-a-Service*): Provisão de ferramentas de desenvolvimento para que o contratante possa implantar seus serviços, participando apenas de configurações associadas ao serviço implantado.

- *IaaS* (*Infrastructure-as-a-Service*): Provisão de recursos computacionais, em que o usuário tem participação total nas configurações desses recursos.

Considerando que tanto os contratantes quanto os provedores de recursos em nuvem podem utilizar ou fornecer mecanismos de gerenciamento de identidade federada, as questões abordadas nesse trabalho são aplicáveis em todos os modelos de serviço.

2.2 ARQUITETURA COM APLICAÇÕES WEB EM CONTEXTO DE COMPUTAÇÃO EM NUVEM

Independentemente do nível de abstração fonecido pelo provedor de recursos em nuvem, o modelo de infraestrutura para hospedar aplicações *Web* possui os seguintes componentes (Figura 1):

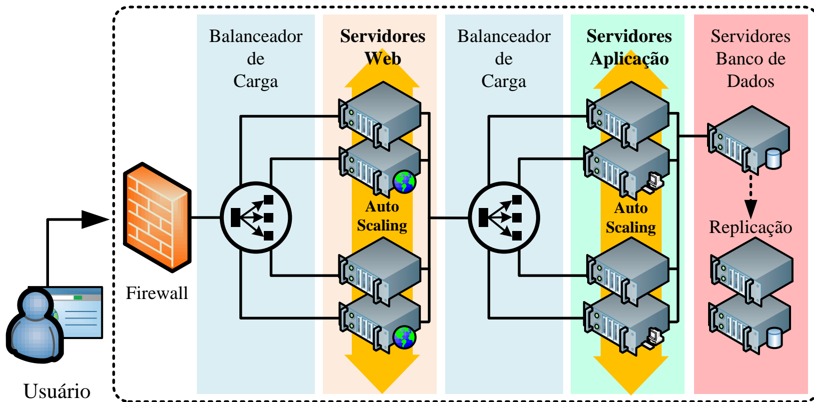

Figura 1 – Infraestrutura tradicional para disponibilização de aplicações *Web*

- Firewall: Área que efetivamente recebe a requisição do usuário e direciona para a infraestrutura do contratante dos recursos em nuvem. Aplica mecanismos como resolução de *DNS*, proteção a ataques como negação de serviço (MIRKOVIC et al., 2004), entre outros.

- Balanceadores de Carga: Tem como responsabilidade direcionar requisições para os servidores que hospedam a aplicação. O balanceador deve garantir que a requisição seja direcionada para o nodo mais disponível, ou seja, o que poderá atender a requisição de maneira mais eficiente. Para isso suporta diferentes algoritmos de balanceamento como *round-robin*, *central management*, *central queue* e *local queue* (SHARMA; SINGH; SHARMA, 2008), além de mecanismos como *sticky-session* (Seção 2.3).

- Servidores *Web* e de Aplicação: Servidores que hospedarão os componentes da aplicação do contratante. Esses servidores são hospedados em infraestruturas que permitem escalabilidade sob

demanda (*Auto Scaling*). Isso significa que a quantidade de recursos aumentará ou diminuirá de acordo com a demanda de carga no sistema.

- Servidores de Banco de Dados: Gerenciam os dados manipulados pela solução, fornecendo instâncias de banco de dados equipados com serviços de *backup* e replicação, de acordo com a necessidade do contratante.

Provedores como *Amazon EC2*, *Windows Azure* e *Google Cloud Platform* fornecem esse tipo de infraestrutura, e considerando a premissa de que os recursos devem ser oferecidos sob demanda (MELL; GRANCE, 2011), a solução deve escalar de acordo com a quantidade de usuários utilizando o sistema.

Essa escalabilidade é garantida atraves do recurso de *Auto Scaling*, que liga e desliga instâncias de servidores *Web* e de *Aplicação* de acordo com a necessidade. Isso significa que a aplicação obrigatoriamente deve ser arquitetada de forma que seu funcionamento não seja afetado pelo mecanismo de *Auto Scaling*.

2.3 STICKY-SESSION

O mecanismo de *sticky-session*, suportado na maioria dos balanceadores, exige que o valor do *Cookie* utilizado para estabelecer a sessão com o usuário contenha um identificador único associado ao nodo que criou o contexto de sessão (Figura 2). Em sistemas *Java*, servidores de aplicação como *Apache Tomcat* e *JBoss* utilizam um *Cookie* denominado *JSESSIONID* para esse fim.

2.4 GERENCIAMENTO DE IDENTIDADE COMO SERVIÇO

Uma identidade digital é uma representação de uma entidade (ou grupo de entidades) na forma de um ou mais elementos de informação (atributos) que permitem reconhecer uma entidade em determinado contexto (CHADWICK, 2009).

Um sistema de gerenciamento de identidade agrega um conjunto de ferramentas para gerenciar identidades individuais em um ambiente digital (CHADWICK, 2009). Uma funcionalidade muito utilizada nesses sistemas inclui o SSO, em que o usuário não precisa se autenticar várias vezes para acessar diversas aplicações (BELAPURKAR et al., 2009).

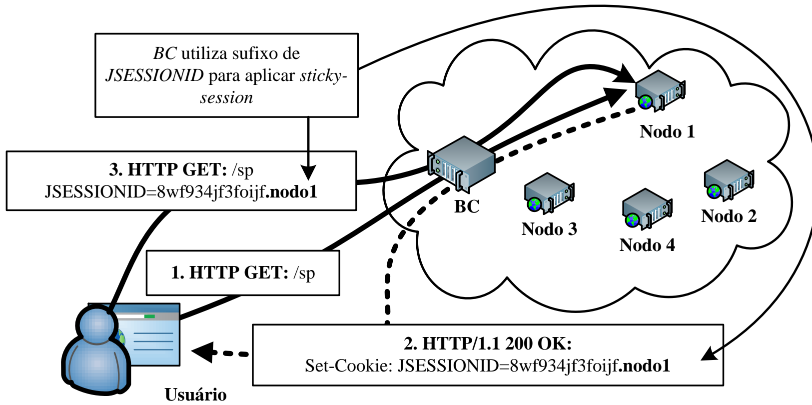

Figura 2 – Fluxo de comunicação com serviços *Web* utilizando *sticky-session*

Podemos categorizar as responsabilidades de um sistema de gerenciamento de identidade da seguinte forma (BELAPURKAR et al., 2009) (BHARGAV-SPANTZEL et al., 2007):

- Autenticação: Garante que o usuário é quem ele diz ser, utilizando mecanismos como senha, biometria, certificado digital, entre outros.

- Autorização: Processo de controle de acesso em diferentes níveis, funcionalidades ou operações dentro de um sistema computacional.

- Federação (Figura 3): Quando um conjunto de organizações compartilha informações de identidade de seus usuários de maneira segura.

Considerando que organizações podem prover serviços de distintos segmentos na nuvem, recomenda-se promover a separação das responsabilidades de gerenciamento de identidade em um serviço específico para esse fim (OLDEN, 2011). A infraestrutura utilizando essa abordagem envolve dois componentes distintos:

- Provedor de Identidade (PI): Sistema que efetivamente autentica o usuário e disponibiliza essa informação para qualquer serviço que precise.

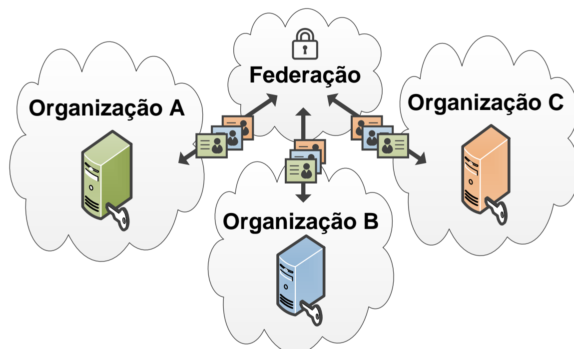

Figura 3 – Federação de identidades entre múltiplas organizações

- Provedor de Serviço (PS): É efetivamente o serviço que o usuário deseja consumir. Caso o acesso seja restrito, o PI deverá ser acionado para prover essas credenciais.

Para as organizações que desenvolvem o PS isso significa menos preocupação com a tecnologia de identidade, investindo mais no gerenciamento do serviço e não da infraestrutura (OLDEN, 2011). Além disso, essa alternativa é imprescindível para viabilizar SSO em contextos com identidades federadas.

Note que a aplicação que gerencia o *Cookie* deverá adicionar esse identificador sempre que a sessão for criada, para que o BC funcione adequadamente. Da mesma forma, em sistemas *Java* esse recurso é configurável na maioria dos servidores de aplicação disponíveis para a plataforma.

2.5 SOLUÇÃO PARA GERENCIAMENTO DE IDENTIDADE FEDERADA COM SHIBBOLETH

Um modelo seguro para gerenciamento de identidades federadas e aderente à recomendação de (OLDEN, 2011) foi concebido por (LEANDRO et al., 2012). Essa infraestrutura foi baseada em duas principais tecnologias *open-source*:

- *Shibboleth*: Implementa uma arquitetura para contextos envolvendo gerenciamento de identidades federadas. Fornece mecanismos seguros no tráfego de credenciais e atributos do usuário, e garante alta compatibilidade por se manter aderente a padrões

W3C, como *Security Assertion Markup Language* (*SAML*) (CANTOR et al., 2005).

- *JASIG CAS*: Solução que fornece um componente desenvolvido em Java para atuar como serviço de autenticação, denominado *CAS Server*. Implementa um protocolo baseado em *HTTP* para autenticação *SSO*, em que sistemas clientes desse serviço podem participar agregando um componente cliente, disponível em diversas plataformas (COMMUNITY, 2012).

No cenário implementado por (LEANDRO et al., 2012), o *CAS* atua como mecanismo de autenticação SSO na arquitetura *Shibboleth*. Essa solução possui os seguintes elementos (Figura 4):

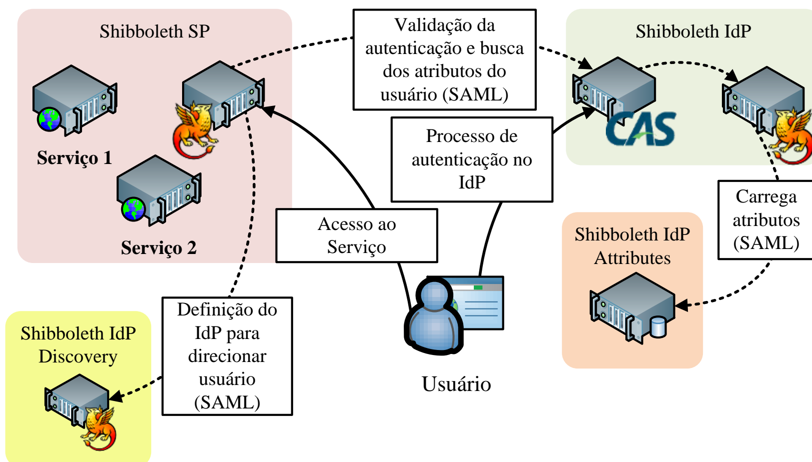

Figura 4 – Elementos de uma infraestrutura de gerenciamento de identidade com *Shibboleth*

- *Shib SP*: Protege os recursos oferecidos pelos serviços que exigem autenticação e autorização. Implementado em *C++*, tem estrutura modular para se integrar com balanceadores de carga e servidores *Web* como *Apache Web Server* e *Microsoft IIS*, atuando como interceptador para aplicar controle de acesso.

- *Shib IdP Discovery*: Diretório que responde ao *Shib SP* onde e como o processo de autenticação deve acontecer, possibilitando ao *Shib SP* que direcione o usuário para iniciar o processo de autenticação.

- *Shib IdP*: Fornece mecanismos para que o usuário informe suas credenciais e se autentique. Além disso, fornece para o *Shib SP* atributos associados ao usuário que está se autenticando, dando insumos para o *Shib SP* verificar se pode liberar acesso a determinado recurso.

- *Shib IdP Attributes*: Repositório, que pode ser externo à infraestrutura do *Shib IdP*, que armazena e fornece os atributos associados a determinado usuário que deseja acesso. Virtualmente qualquer informação pode ser transmitida nesses atributos, qualquer coisa que permita que os elementos envolvidos possam ser assertivos ao validar determinado acesso de um usuário.

Observe que a solução foi concebida de forma que provedores de identidade (*Shib IdP*) de múltiplas organizações podem coexistir para um mesmo provedor de serviço (*Shib SP*) , estabelecendo assim a federação de identidades de maneira confiável.

Por fim, vale salientar que o *CAS* se integra com o *Shibboleth* adotando estratégias de *proxying* de identidades, aplicando assim o fluxo de autenticação estabelecido com *CAS*, mas ainda delegando aos elementos do *Shibboleth* as tarefas descritas na Figura 4.

2.6 FLUXO DE AUTENTICAÇÃO COM *CAS*

Considerando a utilização do *CAS* na infraestrutura de gerenciamento de identidade federada, para que seja possível entender os desafios associados à escalabilidade horizontal é necessário conhecer o fluxo de autenticação SSO com o *CAS*.

Para serviços acessados pelo usuário através de um navegador, o *CAS* implementa um protocolo que faz uso de recursos *HTTP* como *Cookies* e *Redirects*. O fluxo de operação estabelecido nesse protocolo está apresentado na Figura 5, sendo composto por oito etapas:

1. O usuário utiliza o navegador para acessar determinado recurso do serviço.

2. Considerando que o serviço exige credenciais para acesso, o serviço responde um HTTP 302 (Redirect) para que o navegador leve o usuário até o *CAS Server* para que proceda com a autenticação. Note que o *URL* do *redirect* contempla o parâmetro *service*, que contém o *URL* que o *CAS Server* utilizará para redirecionar o

usuário de volta para o serviço, além de futuramente enviar a requisição de *logout*.

3. Caso não tenha se autenticado ainda, uma página é apresentada para o usuário para que ele informe suas credenciais (usuário e senha, certificado digital, etc). Se já estiver autenticado no *CAS Server*, o fluxo segue para a próxima etapa, sem que usuário tenha que informar suas credenciais novamente.

4. Caso o usuário tenha sido identificado com sucesso, o *CAS Server* gerará um identificador único denominado *ticket*. Esse identificador é adicionado como parâmetro no endereço de redirecionamento para o serviço que o usuário havia acessado anteriormente. O *CAS Server* armazena em memória esse *ticket* por um tempo limitado, para ser processado na etapa 7.

5. O navegador faz um novo acesso ao serviço (decorrente do redirecionamento promovido na etapa anterior), dessa vez passando o parâmetro *ticket*.

6. A existência do *ticket* proveniente da requisição do usuário significa para o serviço que o usuário está autenticado no *CAS Server*, e o serviço deve realizar uma requisição para o *CAS Server* repassando esse *ticket*, com o intuito de confirmar se o usuário está realmente autenticado.

7. Ao receber a requisição, o *CAS* valida o *ticket* recebido, respondendo uma mensagem XML com as credenciais do usuário.

8. Após processar a resposta retornada pelo *CAS Server*, o serviço confirma o processo de autenticação e retorna para o usuário o recurso inicialmente solicitado na etapa 1.

Outro mecanismo fornecido pelo *CAS* é o *SSOUT* (BATTAGLIA; SAVAGE, 2012a), em que o usuário realiza uma requisição ao *CAS Server* para sair do sistema. Ao receber essa requisição, o *CAS Server* dispara uma requisição para que todos os serviços acessados pelo usuários também limpem o estado de autenticação (Figura 6).

2.7 SUPORTE À ESCALABILIDADE HORIZONTAL

Para escalar horizontalmente sistemas que utilizam componentes JASIG *CAS*, é necessário viabilizar a clusterização, tanto para o

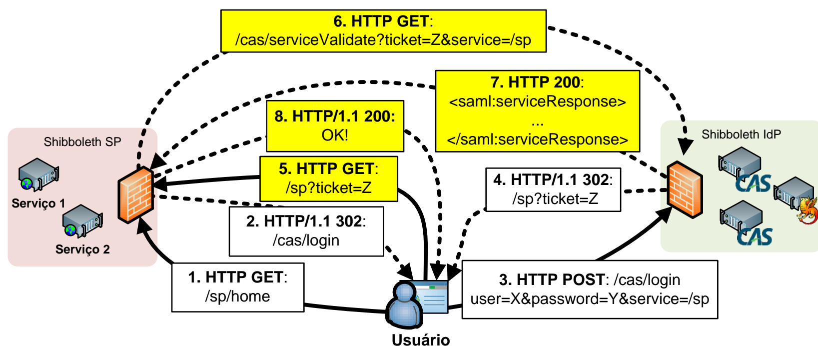

Figura 5 – Fluxo de Autenticação em uma Infraestrutura com *CAS*

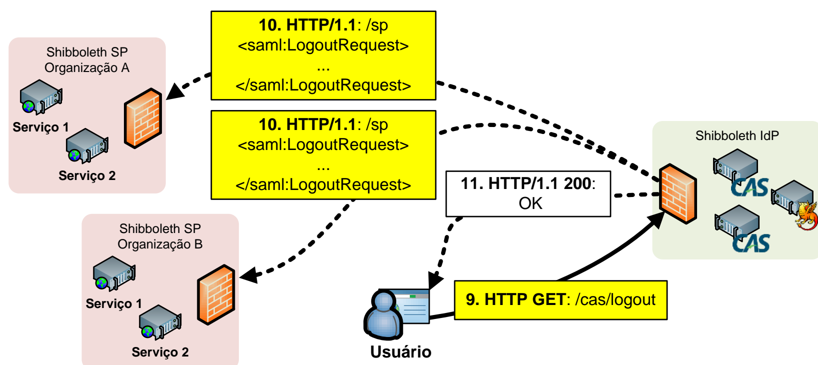

Figura 6 – Fluxo da operação de *SSOUT* fornecido em uma infraestrutura com *CAS*

componente *PI* quanto para os serviços participantes da infraestrutura SSO. O acesso a esses nodos deve ser coordenados por um *BC* (balanceador de carga), como *Apache Web Server*, *Microsoft IIS*, *Amazon EC2 ELB*, entre outros (RANDLES; LAMB; TALEB-BENDIAB, 2010).

A autenticação do usuário com o sistema necessita que se estabeleça uma sessão *HTTP*, para que o processo de autenticação não tenha que ser repetido a cada acesso. Como as informações dessa sessão ficam armazenadas na memória do nodo para o qual o usuário foi direcionado, se no próximo acesso o *BC* direcionar a requisição para outro nodo, nenhuma informação de sessão será encontrada, consequentemente exigindo que o usuário repita o processo de autenticação. Esse é um problema recorrente em sistemas *stateful*, ou seja, sistemas que

precisam manter em memória ações realizadas pelo usuário, como um carrinho de compras em um site de comércio eletrônico (*e-commerce*). Esse é o caso de infraestruturas com *CAS* e *Shibboleth*.

Uma possível solução é garantir que o *BC* direcione requisições subsequentes sempre para o mesmo nodo do primeiro acesso, conforme proposto por (STECCA; BAZZUCCO; MARESCA, 2011). Essa técnica, denominada *sticky-session*, faz uso de *HTTP Cookies* criados na conversação entre o navegador do usuário e o sistema.

Contudo, conforme exposto em (BATTAGLIA; SAVAGE, 2012b), a etapa de comunicação entre o servico e o *CAS Server* também exige que o nodo do *CAS Server* seja o mesmo que o usuário acessou, em decorrência do *ticket* que está armazenado na memória daquele nodo. Nesse momento, o *BC* não recebe os *HTTP Cookies* do usuário, pois quem efetuou a requisição foi o serviço, consequemente podendo direcionar para um nodo que não possui o ticket em memória, falhando assim, o processo de autenticação. Em um contexto de computação em nuvem que adote esse modelo de gerenciamento de identidade como serviço, a necessidade de uma infraestrutura escalável inevitavelmente será afetada por essa falha (Figura 7).

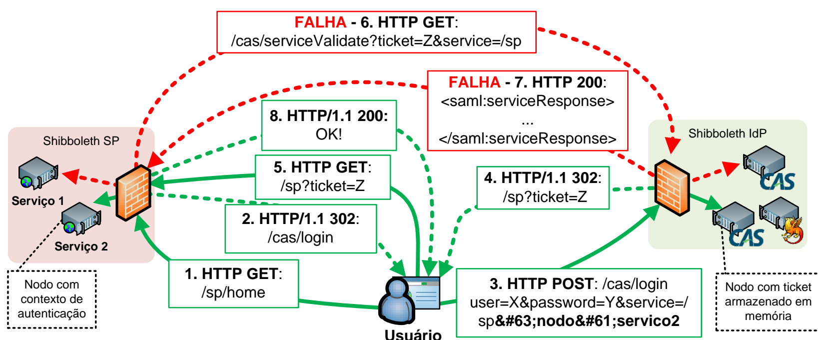

Figura 7 – Simulação de falha no processo de autenticação em serviços escalados horizontalmente com computação em nuvem

O *SSOUT* também é afetado, pois quando o *CAS* processa a requisição *SAML* de *LogoutRequest* para os serviços, o BC pela mesma razão poderá direcionar a requisição para um nodo do *cluster* dos serviços em que o usuário não estava autenticado(Figura 8).

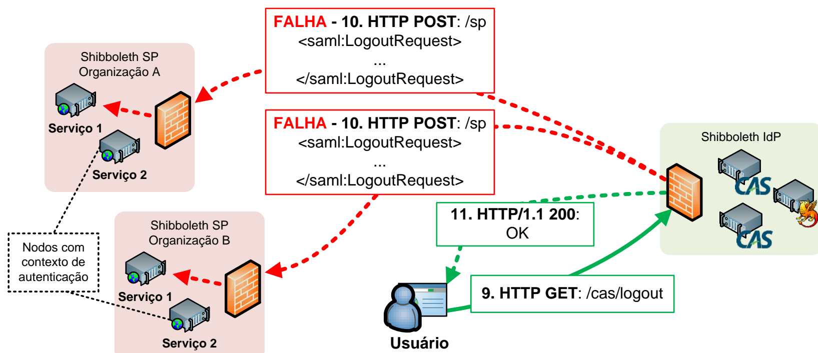

Figura 8 – Falha no processo de *SSOUT* em um contexto de escalabilidade horizontal na computação em nuvem

2.8 MEMÓRIA DISTRIBUÍDA COM TERRACOTTA

Uma alternativa para dirimir essas dificuldades consiste em adotar a plataforma *Terracotta* (TERRACOTTA, 2008). Sendo a abordagem recomendada por (NANDA; KHANAPURKAR; SAHOO, 2011), (BATTAGLIA; SAVAGE, 2012b) e (JOIE; CANTOR, 2012), o *Terracotta* possibilita distribuir dados de memória entre todos os nodos, para que, independentemente para qual nodo o balanceador direcionar à requisição, as informações de sessão com o usuário estejam disponíveis para que a operação de autenticação possa ser estabelecida.

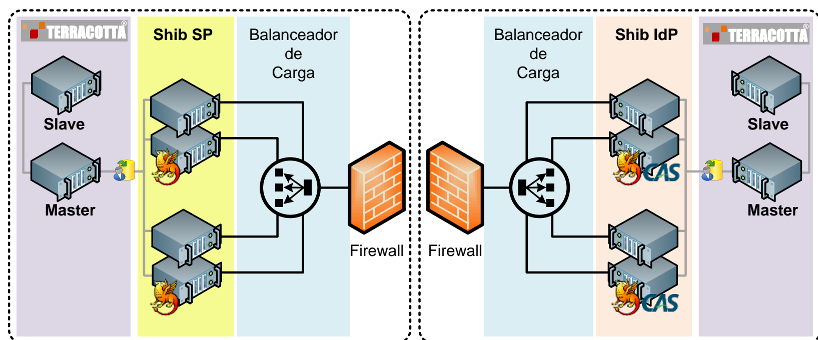

Figura 9 – Serviços utilizando *Terracotta* para distribuição de memória

Conforme pode ser observado na Figura 9, nessa infraestrutura

utilizaremos três componentes do *Terracotta*, dos quais podemos elencar (TERRACOTTA, 2008):

- *Terracotta Server*: Serviço que persiste as informações a serem distribuídas. Também pode ser clusterizado no intuito de fornecer robustez e evitar ponto único de falha, através de um modelo *master/slave*.
- *Web Sessions*: Fornece *APIs* para que servidores de aplicação utilizem o *Terracotta Server* para distribuir informações da sessão do usuário.
- *EhCache*: Disponibiliza *APIs* em *Java* para que aplicações realizem cache ou distribuam qualquer tipo de informação necessária no *Terracotta Server*. Utilizado no *CAS Server*, para distribuir os *tickets* gerados no fluxo de autenticação entre todos os nodos do cluster.

2.9 ANÁLISE DE TRABALHOS CORRELATOS

Os trabalhos correlatos a esta dissertação dividem-se em dois grupos. O primeiro visa consolidar o gerenciamento de identidade e controle de acesso em um contexto de computação em nuvem, apresentando modelos teóricos e até implementações práticas para esse fim. O segundo aborda o gerenciamento da sessão entre usuário e serviços implantados na nuvem, recurso largamente adotado em tecnologias que implementam o gerenciamento de identidade.

Em (CALERO et al., 2010), um modelo de autorização é apresentado visando proteger recursos em nuvem contratados como *SaaS* (*Software-as-a-Service*). A cada requisição, um novo processo de autorização é realizado. Essa alternativa é direcionada para integrações envolvendo *Web Services REST*, já que não devem fazer uso de sessão (RICHARDSON; RUBY, 2008).

Em (OLDEN, 2011), foi realizada uma análise dos desafios ao se consolidar uma infraestrutura de segurança para computação em nuvem, apresentando o conceito de gerenciamento de identidade como serviço e os requisitos necessários para uma potencial solução.

Em (LEANDRO et al., 2012), uma arquitetura de gerenciamento de identidades federadas *multi-tenant* em um ambiente de computação em nuvem é proposto. Um cenário é estruturado de forma que a infraestrutura de serviços é implantada em máquinas virtuais contratadas junto à *Amazon EC2*, e os provedores de autenticação condicionados

em ambientes próprios de entidades terceiras. Esse cenário é implementado utilizando *Shibboleth* e *CAS*, no entanto os autores não analisam a solução sob o ponto de vista da escalabilidade horizontal.

Em (STECCA; BAZZUCCO; MARESCA, 2011), um estudo foi concebido acerca da escalabilidade de serviços implantados em *IaaS*, que precisam estabelecer sessão *HTTP* com o usuário. O trabalho apresenta um cenário com serviços clusterizados, em que um balanceador de carga redireciona as requisições do usuário, aplicando o mecanismo de *sticky-session*. Com isso, duas soluções são propostas, uma delas implementada, com o intuito de fornecer monitoramento e robustez para evitar a indisponibilidade de algum nodo, situação que se não for tratada, faz com que o usuário perca a sessão e consequentemente sua atividade é perdida.

Em (NANDA; KHANAPURKAR; SAHOO, 2011), é apresentada uma análise de viabilidade para se implantar serviços escaláveis, comparando as abordagens *sticky-session* com memória distribuída. No cenário concebido e testado, o melhor resultado foi obtido com uma abordagem baseada em memória distribuída. Contudo, a infraestrutura analisada não contempla a existência de mecanismos de gerenciamento de identidade federada.

Em (LIU; WEN, 2011), a escalabilidade com *CAS* é fornecida através da modelagem de um modelo de autenticação *clusterizado*, provendo um balanceador de carga para gerenciar múltiplas instâncias do servidor *CAS*. Os autores não analisam problemas com escalabilidade no lado do cliente, especificamente suporte ao mecanismo *SSOUT*. O trabalho é preliminar, sendo possível que novos trabalhos contemplando uma implementação possam surgir. Em (HUANG; WANG; LONG, 2011), um servidor de autenticação é implementado, gerenciando nodos de servidores *CAS* e balanceando requisições adequadamente. Assim como em (LIU; WEN, 2011), problemas de escalabilidade no lado do cliente não são endereçados. Comparando o trabalho exposto por (LIU; WEN, 2011) e (HUANG; WANG; LONG, 2011) com essa dissertação, resolver o problema de escalabilidade com o servidor *CAS* é um objetivo comum. Contudo, essa dissertação também mitiga problemas de escalabilidade nas aplicações cliente que participam da infraestrutura de autenticação com *CAS*. Adicionalmente, nenhuma modificação foi aplicada ao protocolo de autenticação com *CAS*, garantindo assim compatibilidade quando a solução é utilizada na arquitetura com *Shibboleth* em nuvem, proposta por (LEANDRO et al., 2012).

3 PROPOSTA COM STICKY-SESSION ADAPTADO

O aumento na complexidade de implantação e os recursos utilizados em uma infraestrutura com *Terracotta* implica em custos mais elevados para a organização.

No contexto de soluções que utilizam *Shibboleth* e *CAS*, as informações distribuídas utilizando *Terracotta* manifestam algumas preocupações no aspecto da segurança, já que o teor dessas informações ficam expostas nos nodos do *Terracotta Server*.

Diante disso, nesta seção é proposta uma nova alternativa, sem utilizar mecanismos de distribuição de memória. A proposta faz adaptações no fluxo de autenticação dos componentes da infraestrutura com o *CAS* para viabilizar o uso do mecanismo de *sticky-session*. A seção 3.1 detalha o funcionamento dessa proposta, enquanto os 3.2 apresenta o cenário de testes, construído para validar e comparar a proposta em relação à abordagem com *Terracotta*.

3.1 DETALHAMENTO DA PROPOSTA

Conforme exposto anteriormente, a dependência de *HTTP Cookies* para que o balanceador direcione corretamente as requisições a princípio impossibilitaria sua utilização em infraestruturas com *CAS*. Contudo, o uso do *sticky-session* é viável, desde que promovendo as seguintes modificações:

- Configuração no *CAS Server*: O *ticket* gerado que será utilizado na etapa 6 do fluxo de autenticação (detalhada na seção 2.6) deverá agregar o sufixo com o mesmo identificador contido no *JSESSIONID*. O *CAS Server* permite que essa configuração seja realizada.

- Configuração no *BC* dos nodos do *CAS Server*: A regra de *sticky-session* baseada no sufixo do valor do *Cookie JSESSIONID* deve ser replicada para considerar também o sufixo do valor do parâmetro *ticket* enviado pelo serviço na mesma etapa citada no item anterior. Balanceadores de carga como *Apache Web Server* e *Microsoft IIS* suportam essa configuração.

Essas adaptações permitem que o *BC* possa direcionar adequadamente não apenas requisições realizadas pelo usuário, mas também

pelo serviço, conforme mostrado na Figura 10 e Figura 11. Assim a falha no processo de autenticação evidenciada na Figura 7 é dirimida.

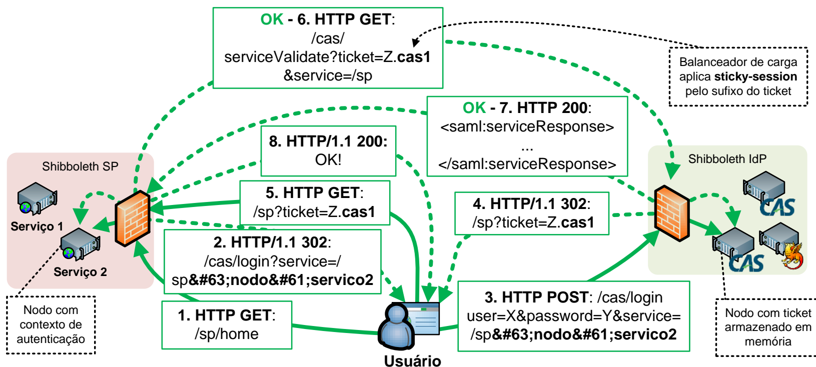

Figura 10 – Visão geral da adaptação dos elementos para aplicar *sticky-session* com o sufixo do *ticket* gerado pelo *CAS Server*

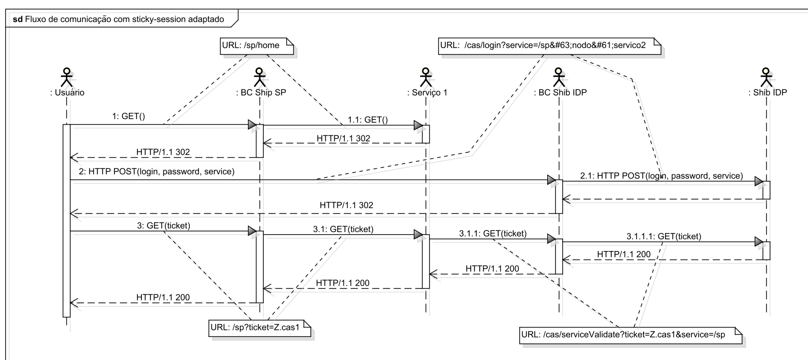

Figura 11 – Representação em diagrama de sequência *UML* da adaptação dos elementos para aplicar *sticky-session* com o sufixo do *ticket* gerado pelo *CAS Server*

Também é possivel utilizar o mecanismo de *SSOUT* com *sticky-session*, eliminando o problema identificado na seçao 2.7. Issó é possivel realizando as seguintes adaptações:

- Adição do identificador do nodo no parâmetro *service* (Figura 10): O parâmetro *service* informado pelo serviço (etapa 2 da seção 2.6)

contém a *URI* que o *CAS Server* utilizará na notificaçao de solicitação de *logout* do usuário. Essa *URI* deverá ser modificada para agregar um novo parâmetro que indique o nodo que estabeleceu a sessão com o usuário, permitindo que o BC direcione a notificação de *logout* corretamente. Essa configuração deverá ser feita no componente disponibilizado pelo *CAS*, que realiza as requisições de responsabilidade do serviço.

- Configuração no *BC* dos nodos do cluster de serviços (Figura 12 e Figura 13): Do mesmo modo das modificações feitas no *BC* do cluster de *CAS Server*, é preciso considerar o valor de um parâmetro recebido para promover o *sticky-session*.

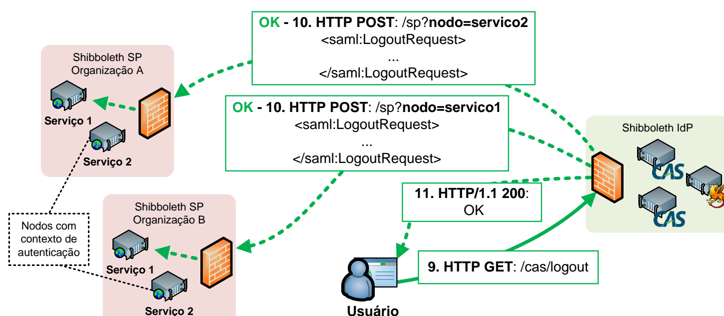

Figura 12 – Visão geral da do processo de *logout* com *sticky-session* adaptado

Considerando que essas modificações exigem que o identificador do nodo que está participando do processo de autenticação trafegue entre os componentes da solução é necessário aplicar mecanismos de criptografia para evitar que esse identificador de nodo seja utilizado para fins de ataque, especialmente *DoS*(*Denial of Service*), em que um atacante pode utilizar o identificador para forçar a seleção de um nodo a ser atacado. A comunicação *SSL* garante essa segurança para comunicações envolvendo o usuário, no entanto em comunicações entre provedor de serviço e identidade se faz necessário o uso de criptografia específica do conteúdo trafegado, recurso já garantido pelas soluções *Shibboleth* e *CAS*.

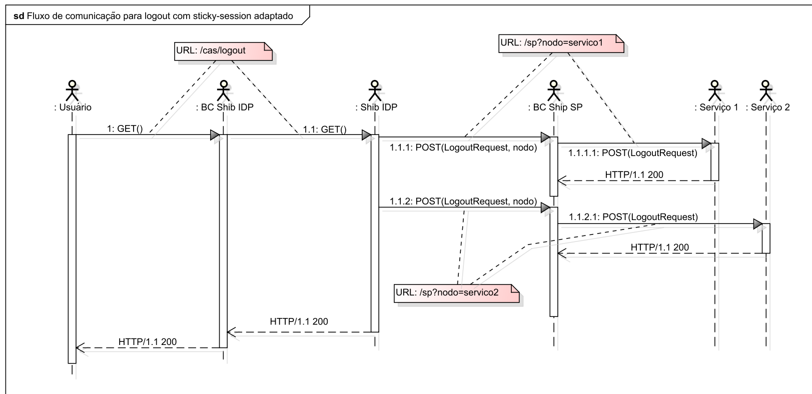

Figura 13 – Representação em diagrama de sequência *UML* do processo de *logout* com *sticky-session* adaptado

3.2 RESULTADOS EXPERIMENTAIS

Com o intuito de validar a alternativa com *sticky-session* adaptado e comparar com a solução utilizando *Terracotta*, um cenário foi construído utilizando máquinas virtuais. Os componentes da infraestrutura foram distribuídos em dois conjuntos de *virtual machines* (*VM*) em redes diferentes, simulando recursos em nuvem de diferentes provedores. O primeiro conjunto (*VM1* e *VM2*) contém o cluster de serviços, enquanto o segundo (*VM3* e *VM4*) hospeda o cluster de *CAS Servers*, conforme mostrado na Figura 14.

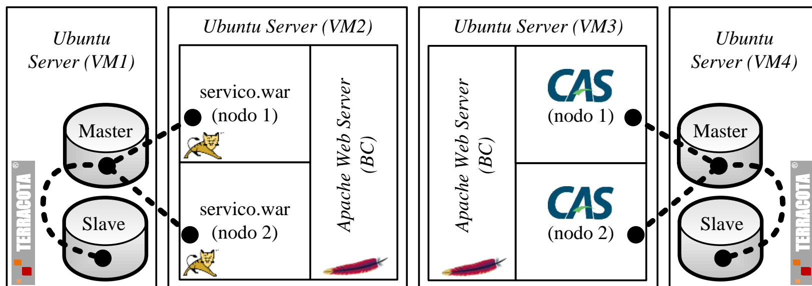

Figura 14 – Cenário para simular as alternativas para agregar suporte de escalabilidade horizontal em infraestruturas com *CAS*

Tabela 1 – Distribuição de recursos de processamento para cada *VM*

Máquina Virtual	Processador	Memória	Disco Rígido
VM1	2x Core 2.8Ghz	1024MB	10Gb
VM2	2x Cores 2.8Ghz	2048MB	10Gb
VM3	2x Cores 2.8Ghz	2048MB	10Gb
VM4	2X Core 2.8Ghz	1024MB	10Gb

Essas máquinas virtuais foram instanciadas em um servidor hospedeiro, com as seguintes especificações: Processador *Phenom II X6 1055T (2.8Ghz)*, *8192MB* de memória *DDR3* e um disco rígido *SAMSUNG HD154UI 1.5Tb*. Os recursos computacionais configurados para cada *VM* estão listados na Tabela 1.

Por fim, um outro equipamento com especificações semelhantes ao do servidor hospedeiro foi utilizado para executar a simulação de requisições com o *Apache JMeter*. Os equipamentos estão conectados através de uma rede com largura de banda de *1 Gb/s*.

Para que a proposta com *sticky-session* possa ser validada nesse cenário, assim que as adaptações detalhadas na seção 3 são aplicadas, a plataforma de memória distribuída com *Terracotta* é desativada em ambos os conjuntos de máquinas virtuais. Em ambas as propostas o comportamento esperado na infraestrutura de autenticação deverá ser o mesmo.

Em cada modelo de configuração (*sticky-session* ou *Terracotta*), um roteiro de testes de carga é executado com a ferramenta *Apache JMeter*. Esse roteiro simula um conjunto de usuários simultâneos realizando cada um uma operação de autenticação e em seguida, uma de *logout*. Dessa execução, métricas como a carga nos servidores e o *throughput*, proporcionam os valores que são coletados, permitindo uma análise comparativa de desempenho das soluções.

Nas próximas subseções será apresentado um guia de configuração de cada componente que compõe esse cenário, além de uma subseção específica para detalhar a configuração e roteiro de testes com a ferramenta *JMeter*.

3.2.1 Configuração do Balanceamento de Carga com Apache Web Server

Componente essencial para simular uma infraestrutura *Web* no contexto de computação em nuvem, uma instância do balanceador de carga *Apache Web Server* foi configurado para direcionar requisições para o cluster de servidores de aplicação *Tomcat*, tanto na *VM2* (que hospeda a aplicação provedora do serviço que o usuário deseja acessar) quanto na *VM3* (que hospeda os nodos do *CAS Server*).

Considerando que o *Apache Web Server* versão *2.4* oferece módulos específicos dependendo do tipo de balanceamento e protocolo desejado, nesse cenário, o balanceamento foi implementado através do módulo *mod_proxy* (ABELE, 2012).

O arquivo de configuração do *Apache Web Server* (*httpd.conf*) foi concebido com 7 características para viabilizar o balanceamento tanto na *VM2* quanto *VM3* (Figura 15):

```
Listen 80        (1)
Listen 443
...
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_ajp_module modules/mod_proxy_ajp.so                        (2)
LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
LoadModule ssl_module modules/mod_ssl.so
LoadModule lbmethod_byrequests_module modules/mod_lbmethod_byrequests.so
...
<VirtualHost *:443>   (3)
  SSLEngine On
  SSLCertificateFile "certificado.crt"
  SSLCertificateKeyFile "chave_privada.key"   (4)
  <Proxy balancer://cluster>
    BalancerMember ajp://authserver:8009 loadfactor=1 route=server1 retry=10   (5)
    BalancerMember ajp://authserver:8109 loadfactor=2 route=server2 retry=10
    ProxySet stickysession=JSESSIONID|ticket   (6)
  </Proxy>
  ProxyPass /context balancer://cluster/context timeout=10 maxattempts=3   (7)
  ProxyPassReverse /context balancer://cluster/context
</VirtualHost>
```

Figura 15 – Arquivo de configuração do *Apache Web Server*

1. Portas: A instrução *Listen* sinaliza ao *Apache Web Server* quais portas deverão receber requisições. Nesse caso foram habilitadas as portas 80 e 433, a primeira utilizada para conexões *HTTP* simples e a segunda para conexões *HTTPS*, imprescíndiveis para

um contexto de autenticação em que credenciais trafegam entre usuário e servidor.

2. Módulos: A instrução *LoadModule* carrega o módulo associado à funcionalidade do *Apache Web Server* que se tem interesse. O suporte para o balanceamento de carga de servidores de aplicação *Apache Tomcat* é garantido pelo módulo *mod_proxy* e derivados (*mod_proxy_ajp* e *mod_proxy_balancer*), enquanto o *SSL/TLS* é implementado pelo módulo *mod_ssl*. Por fim, o algoritmo de balanceamento é implementado por *mod_lbmethod_byrequests*.

3. *VirtualHost*: Domínio que receberá todas as configurações de mapeamento de requisições, para determinado endereço (nesse caso configurado com ***, que significa qualquer endereço) e porta (*443* que corresponde ao *SSL/TLS*.

4. Configuraçoes *SSL/TLS*: Sinaliza que o *SSL/TLS* deverá ser utilizado nesse *VirtualHost* e indica os arquivos (par de chaves *PKCS12*) que processarão a criptografia assimétrica exigida pelo protocolo.

5. *Balancer*: Indica os nodos (servidores) que receberão as requisiçõess de acordo com as configurações de balanceamento (parâmetros *loadfactor*, *retry*, entre outros).

6. *Sticky-session*: Essa instrução configura o mecanismo de *sticky-session*. Nesse cenário estabelece que o sufixo do valor dos *HTTP Cookies* denominados *JSESSIONID* ou *ticket* contém o identificador do nodo que deverá receber a requisição. Note que essa instrução é desabilitada ao simular o cenário com memória distribuída, já que qualquer nodo poderá atender a requisição.

7. *ProxyPass*: Sinaliza qual aplicação receberá esse balanceamento. Na *VM2* o valor *"/context"* (em negrito) é substituído por *"/servico"*, enquanto na *VM3* o valor deverá ser substituído por *"/cas"*.

3.2.2 Instâncias do *CAS Server*

Para a aplicação *CAS Server* duas versões foram preparadas, uma para a configuração com memória distribuída (*Terracotta*) e uma para compatibilizar com o modelo em *sticky-session*.

3.2.2.1 Configuração do *CAS Server* para o cenário com *Terracotta*

Conforme mostrado na Figura 9, também é necessário habilitar o *Terracotta Web Sessions* na instância do *CAS Server*.

Contudo, o mecanismo que resolve o problema de clusterização da solução com *Shibboleth* e *CAS* consiste no *Terracotta EHCACHE*, que tem como objetivo distribuir os *tickets* emitidos pelo *CAS Server* na etapa 6 do fluxo de autenticação (detalhada na seção 2.6). Essa configuração é realizada no arquivo *ticketRegistry.xml* do *CAS Server*, que é modificado para referenciar o *XML* de configuração do *EHCACHE*, denominado *ehcache-replicated.xml*. Esse arquivo contém o endereço do *Terracotta Server* e características como quais elementos devem ser gerenciados pelo *Terracotta*, como *timeout* e tolerância a falhas da infraestrutura (Figura 16).

3.2.2.2 Configuração do *CAS Server* para o cenário com *Sticky-session*

A compatibilização com o mecanismo de *sticky-session* exige que o *ticket* do fluxo de autenticação gerado pelo *CAS Server* contenha o identificador do nodo que gerou o ticket (Figura 10). Essa configuração é realizada no arquivo *uniqueIdGenerators.xml* (Figura 17).

3.2.3 Instâncias do serviço (recurso a ser acessado)

Com o intuito de simular o acesso a um serviço com acesso restrito, uma aplicação de teste em *Java* foi desenvolvida para esse fim. Da mesma forma que o *CAS Server*, o servidor de aplicação *Apache Tomcat* foi utilizado. A abordagem utilizada nessa implementação envolveu duas *APIs*:

- *JASIG CAS Java Client* (KIRST, 2012): Fornece recursos para que a aplicação *Web* em *Java* participe do protocolo de autenticação com *CAS*.

- *Spring Security* (WINCH, 2012): Implementa recursos de controle de acesso, além de fornecer integração com o fluxo de autenticação provido por *JASIG CAS Java Client*.

Com o *Spring Security*, configurações como o endereço do *CAS Server* para o qual o usuário será direcionado, endereço para validação

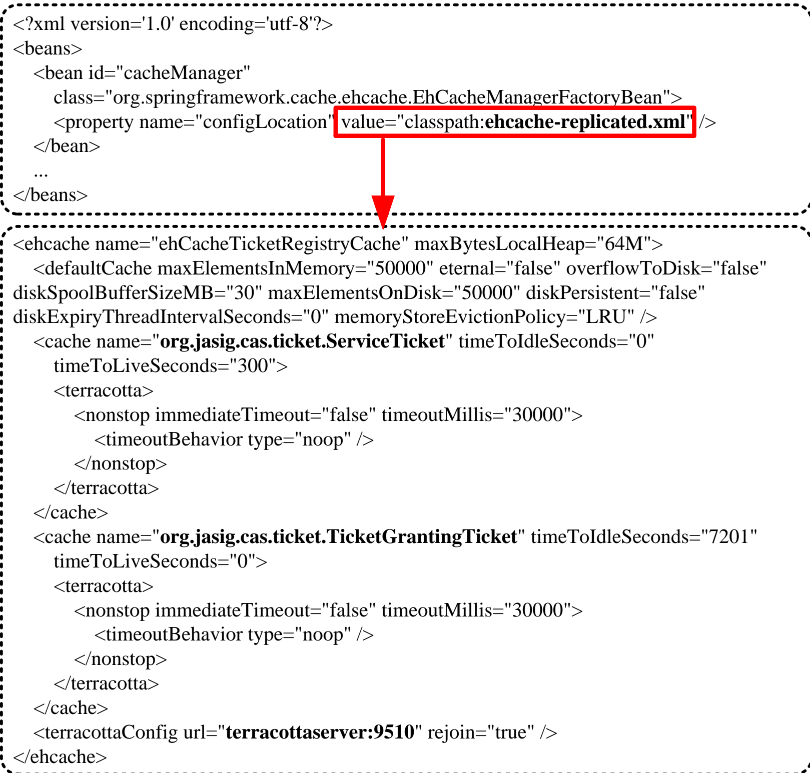

Figura 16 – Configurações no *CAS Server* para distribuição dos *tickets* gerados no fluxo de autenticação

do *ticket*, *URLs* que devem ser protegidas e quais políticas de acesso devem ser aplicadas ficam no arquivo *applicationContext.xml* (Figura 18).

3.2.4 Preparação dos Monitores dos Recursos de *Hardware*

Para contabilizar os recursos de hardware utilizados durante a execução do roteiro de teste foi desenvolvido um aplicativo baseado no *framework Hyperic Sigar* (MORGAN, 2010).

Desenvolvido em *Java*, o aplicativo faz a coleta de uso do processador e memória pelos componentes de *software* envolvidos na solução. Assim que o aplicativo é iniciado, é feita uma coleta a cada segundo, e

```
<?xml version='1.0' encoding='utf-8'?>
<beans>
   <bean id="ticketGrantingTicketUniqueIdGenerator"
      class="org.jasig.cas.util.DefaultUniqueTicketIdGenerator">
      <constructor-arg index="0" type="int" value="50" />
      <constructor-arg index="1" value="nodo1" />
   </bean>
   <bean id="serviceTicketUniqueIdGenerator"
      class="org.jasig.cas.util.DefaultUniqueTicketIdGenerator">
      <constructor-arg index="0" type="int" value="20" />
      <constructor-arg index="1" value="nodo1" />
   </bean>
   ...
   <bean id="proxy20TicketUniqueIdGenerator"
      class="org.jasig.cas.util.DefaultUniqueTicketIdGenerator">
      <constructor-arg index="0" type="int" value="20" />
      <constructor-arg index="1" value="nodo1" />
   </bean>
   ...
</beans>
```

Figura 17 – Configurações no *CAS Server* para agregar identificador do nodo no *ticket* do fluxo de autenticação

as informações são registradas em um arquivo *csv*, permitindo a criação de gráficos para uma posterior análise (Figura 19).

3.2.5 Configuração do *JMeter* para execução dos testes

Apache JMeter oferece recursos para realizar testes de carga em aplicações *Web*, fornecendo uma interface gráfica que permite encadear ações que o usuário realizaria, possibilitando execução de requisições conforme o desejado. Cada iteração do teste com o *Apache JMeter* corresponde às seguintes ações (Figura 20):

1. *acesso_casclient_que_redireciona_para_cas*: Simula a requisição de um usuário que deseja acessar um serviço que possui restrição de acesso.

2. *acesso_cas_submetendo_login*: Considerando que o serviço redirecionará o usuário para o serviço de autenticação (etapa 2 da seção 2.6), simula a submissão das credenciais (login e senha) do usuário. Por fim verifica se o fluxo após submissão das credenciais continuou normalmente, validando se o *CAS Server* redirecionou o usuário novamente para o serviço, dessa vez com a informação

```
<?xml version='1.0' encoding='utf-8'?>
<beans>
  ...
  <security:http entry-point-ref="casEntryPoint" >
    <security:custom-filter position="CAS_FILTER" ref="casFilter" />
    <security:intercept-url pattern="/**" access="ROLE_USER" />
  </security:http>
  ...
  <bean id="casEntryPoint"
    class="org.springframework.security.cas.web.CasAuthenticationEntryPoint">
    <property name="loginUrl" value="https://authserver:8443/cas/login" />
    <property name="serviceProperties" ref="serviceProperties" />
  </bean>
  ...
  <bean id="casAuthenticationProvider"
    class="org.springframework.security.cas.authentication.CasAuthenticationProvider">
    <property name="userDetailsService" ref="userService" />
    <property name="serviceProperties" ref="serviceProperties" />
    <property name="ticketValidator">
      <bean class="org.jasig.cas.client.validation.Cas20ServiceTicketValidator">
        <constructor-arg index="0" value="https://authserver:8443/cas" />
      </bean>
    </property>
    <property name="key" value="authserver" />
  </bean>
  ...
</beans>
```

Figura 18 – Arquivo de configuração *applicationContext.xml* utilizado no serviço com acesso restrito

do *ticket*.

3. *validar_logout_se_login_ok*: Caso a autenticação tenha sido realizada com sucesso, tenta novamente acessar o serviço restrito. Ao conseguir esse acesso, inicia imediatamente um processo de *logout* no *CAS Server*. Por fim, faz um novo acesso ao serviço, que dessa vez deverá ser negado após operação de *logout*.

3.3 ANÁLISE COMPARATIVA DOS RESULTADOS

Nessa seção é apresentada uma análise comparativa das duas propostas de adaptação da infraestrutura de gerenciamento de identidade federada para que possa ser escalada horizontalmente.

Considerando o cenário concebido, o roteiro de testes foi executado com 3 configurações diferentes: a primeira com 50, a segunda

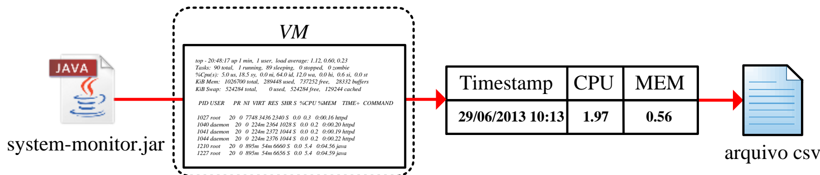

Figura 19 – Fluxo de funcionamento do monitor de *CPU* e memória desenvolvido em *Java*

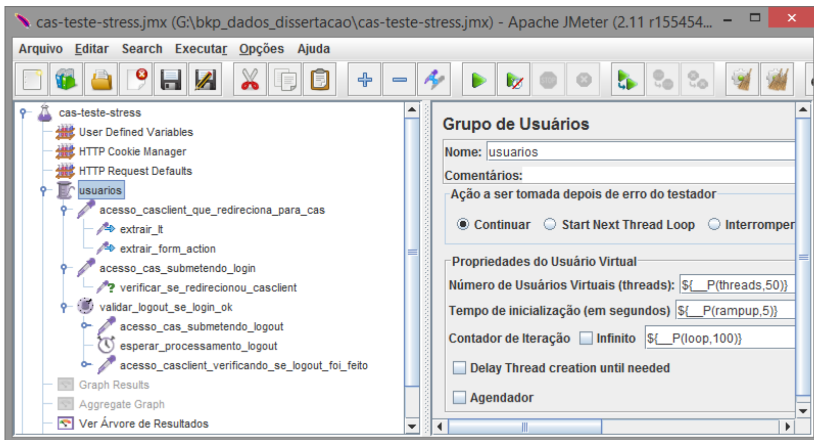

Figura 20 – *Apache* JMeter com *GUI* para configuração e execução de roteiro de testes

com 100 e uma última com 150 usuários simultâneos. O número de requisições para os serviços realizados em cada configuração totalizam, respectivamente, 20000, 40000 e 60000. Cada iteração de requisições de um usuário trafegou em média *228,1 KBytes* por segundo.

O primeiro critério comparado foi o *throughput* (requisições por segundo) obtido para cada uma das infraestruturas. Ambas as abordagens tiveram desempenho adequado, contudo a com *sticky-session* adaptado mostrou *throughput* superior (Figura 21).

Percebe-se também que a abordagem com *Terracotta* apresentou falhas em algumas operações de autenticação (Figura 22). Todas as ocorrências registradas estão associadas ao processo de validação do *ticket* pelo serviço ao *CAS Server*. Ao investigar melhor a questão detectou-se que em momentos de pico de uso de memória pela *VM1*, o *Terracotta* fez uso do disco rigído para o armazenamento das in-

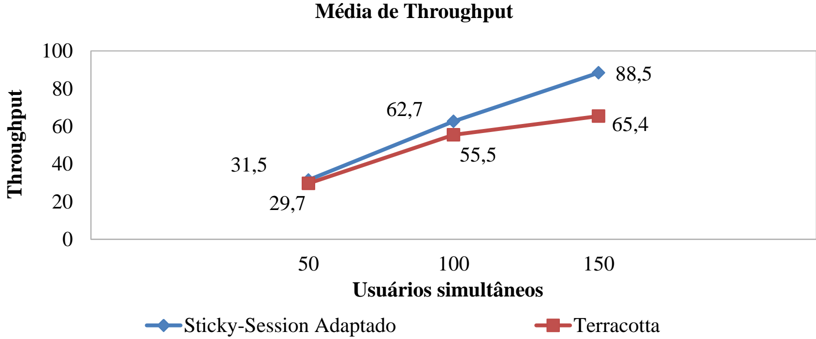

Figura 21 – Gráfico comparativo da média de *throughput*

formações, dado a limitação de memória no cenário, elevando a latência no tráfego desses dados aos nodos do *CAS Server*.

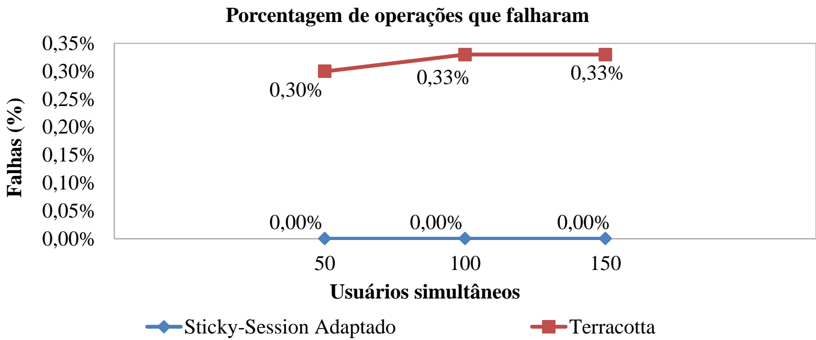

Figura 22 – Gráfico comparativo da porcentagem de falhas por operação de cada usuário

O segundo critério confronta o uso de recursos computacionais para cada abordagem, em que, enquanto o roteiro de testes era executado, uma coleta da porcentagem de uso de *CPU* e memória foi realizado a cada segundo.

Comparando a média de uso de recursos computacionais (Figuras 23, 24, 25 e 26), para o mesmo roteiro de testes, percebe-se que a alternativa com *Terracotta* exige mais recursos, cerca de 58% a mais para cada servidor que faz uso de um ambiente de memória distribuída remoto. Além disso, essa infraestrutura exigiu mais duas máquinas virtuais (*VM1* e *VM4*), para hospedar os *Terracotta Servers* dedicados.

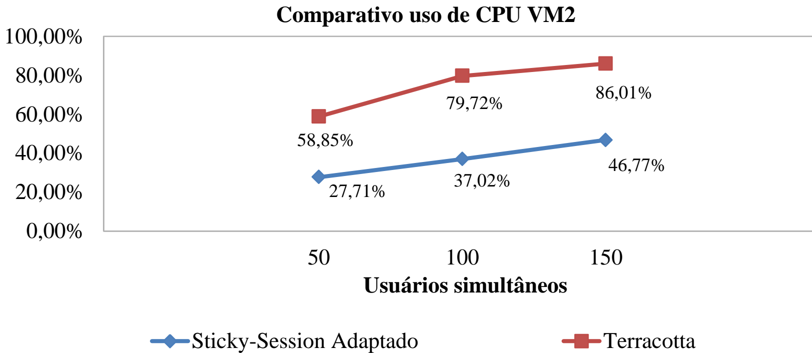

Figura 23 – Gráfico comparativo sobre o uso de CPU da *VM2* entre as abordagens

Por fim, se tratando do *throughput* (requisições por segundo), a abordagem com *sticky-session* adaptado mostrou que consegue 11% mais desempenho.

Apesar do mecanismo de *sticky-session* teoricamente exigir mais processamento e uso de memória pelo balanceador de carga, o *overhead* gerado pelo gerenciamento de sincronização das informações no ambiente de memória distribuída foi maior.

A partir das informações coletadas e da percepção das vantagens e desvantagens de cada abordagem durante o processo de desenvolvimento, estabelecemos na Tabela 2 um quadro comparativo geral dos mecanismos que permitem agregar suporte à escalabilidade horizontal em uma infraestrutura de gerenciamento de identidade federado para computação em nuvem.

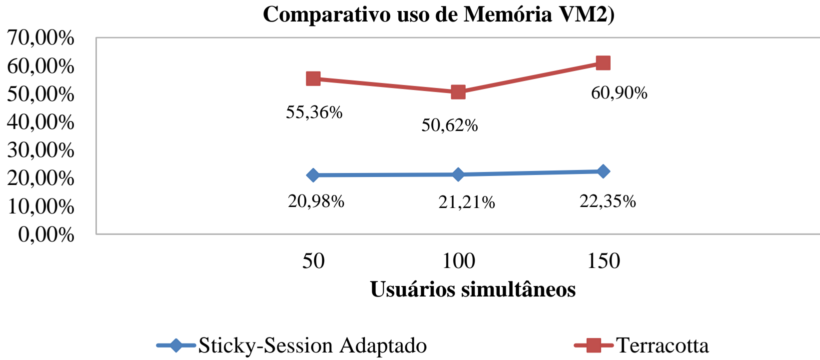

Figura 24 – Gráfico comparativo sobre o uso de memória da *VM2* entre as abordagens

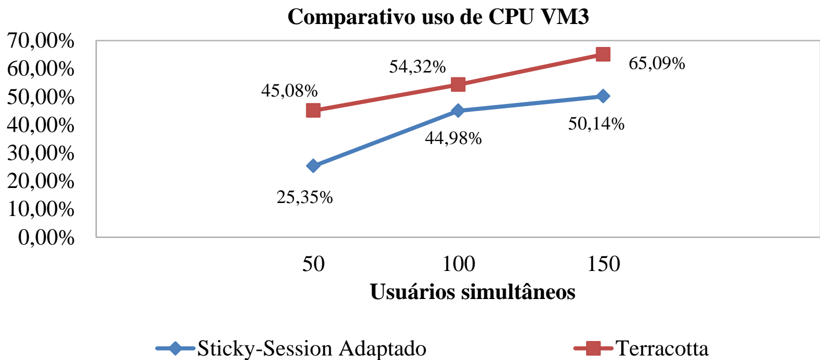

Figura 25 – Gráfico comparativo sobre o uso de CPU da *VM3* entre as abordagens

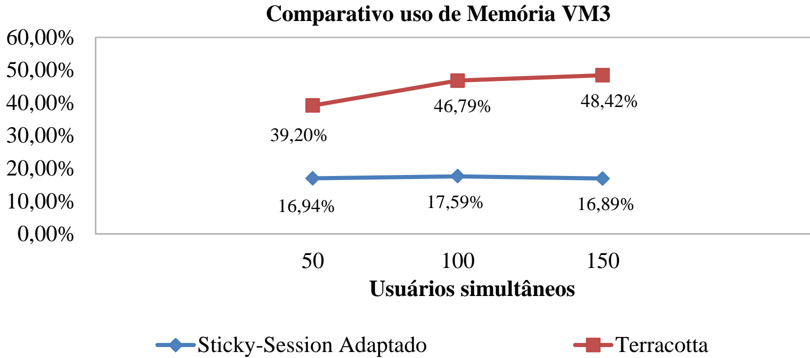

Figura 26 – Gráfico comparativo sobre o uso de memória da *VM3* entre as abordagens

Tabela 2 – Comparativo das abordagens

Criterio	***Terracotta***	***Sticky-session* adaptado**
Desempenho	Adequado	Melhor
Recursos computacionais necessários	Maior	Menor
Complexidade de implantação	Servidores dedicados e configurações para a comunicação entre os nodos clientes	Configurações no balanceador de carga
Suporte a escalabilidade dinâmica	Sim	Possível, com mecanismos de monitoramento e migração de sessão (STECCA; BAZZUCCO; MARESCA, 2011)

4 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apontou os obstáculos encontrados para viabilizar a escalabilidade horizontal em uma plataforma de gerenciamento de identidade federada para computação em nuvem. Uma proposta envolvendo a adaptação do mecanismo de *sticky-session* foi apresentada, com o intuito de fornecer uma alternativa ao mecanismo de memória distribuída, proporcionando a implantação de serviços com menores custos, dependendo da natureza do negócio.

Comparando o uso de recursos computacionais, percebe-se que a alternativa com *Terracotta* exigiu mais recursos, cerca de 58% a mais para cada servidor que faz uso de um ambiente de memória distribuída. Além disso, essa infraestrutura exigiu recursos adicionais para hospedar servidores *Terracotta* dedicados. Com relação ao *throughput* (requisições por segundo), a abordagem proposta, com *sticky-session* adaptado, superou em 11% a alternativa com *Terracotta*.

No entanto, mesmo com o mecanismo de *sticky-session* adaptado, pode-se verificar duas limitações inerentes a este mecanismo. A primeira está associada à escalabilidade horizontal dinâmica, em que a disponibilidade de recursos computacionais em nuvem deve aumentar e diminuir conforme a quantidade de acessos. Isso significa ligar ou desligar uma instância que hospeda a aplicação. Como dados de memória não são distribuídos, se um nodo for desligado os dados de memória contidos nele serão perdidos, fazendo com que usuários associados a esse nodo percam o trabalho que estavam fazendo. A segunda ocorre quando determinado servidor falha. Da mesma forma, como os dados contidos na memória do servidor não são distribuídos, a sessão estabelecida com usuários conectados a esse servidor é destruída. Esses problemas podem ser dirimidos com mecanismos de migração de sessão e tolerância a falhas propostas por (STECCA; BAZZUCCO; MARESCA, 2011).

Uma infraestrutura para gerenciamento de identidades federadas para um contexto de computação em nuvem foi proposta por (LEANDRO et al., 2012). Contudo, o estudo não contemplou o suporte à escalabilidade horizontal dos componentes dessa infraestrutura, aspecto imprescindível para serviços na nuvem (ARMBRUST et al., 2010). A proposta aqui apresentada, com *sticky-session* adaptado, contribui neste sentido, propondo as alternativas para agregar suporte a esses componentes.

Como primeiro trabalho futuro, a infraestrutura proposta por

(LEANDRO et al., 2012) aliada ao mecanismo de *sticky-session* adaptado proposto nesse trabalho, pode ser implantada em um ambiente real de computação em nuvem, coletando dados estatísticos reais e inclusive comparando com a mesma infraestrutura com *Terracotta*.

Um segundo trabalho futuro consiste em minimizar as limitações de escalabilidade dinâmica e gerenciamento de sessão, aplicando mecanismos de monitoramento e migração de sessão nessa infraestrutura, como apontado em (STECCA; BAZZUCCO; MARESCA, 2011).

Por fim, um terceiro trabalho futuro poderia adicionar à pesquisa outra alternativa para mitigar os problemas de escalabilidade horizontal da solução, como *DHT*, largamente utilizada em ambientes *P2P*.

REFERÊNCIAS

ABELE, N. K. E. *Apache Module mod_proxy*. 2012. http://httpd.apache.org/docs/2.4/mod/mod_proxy.html. Acessado em: .

ARMBRUST, M. et al. A view of cloud computing. *Communications of the ACM*, ACM, v. 53, n. 4, p. 50–58, 2010.

BATTAGLIA, S.; SAVAGE, B. *Jasig CAS Documentation : Single Sign Out*. 2012. https://wiki.jasig.org/display/CASUM/Single+Sign+Out. Acessado em: .

BATTAGLIA, S.; SAVAGE, B. *Jasig CAS Documentation: Clustering CAS*. 2012. https://wiki.jasig.org/display/CASUM/Clustering+CAS. Acessado em: .

BELAPURKAR, A. et al. *Distributed systems security: issues, processes and solutions*. [S.l.]: Wiley, 2009.

BHARGAV-SPANTZEL, A. et al. User centricity: a taxonomy and open issues. *Journal of Computer Security*, IOS Press, v. 15, n. 5, p. 493–527, 2007.

CALERO, J. M. A. et al. Toward a multi-tenancy authorization system for cloud services. *Security & Privacy, IEEE*, IEEE, v. 8, n. 6, p. 48–55, 2010.

CANTOR, S. et al. Shibboleth architecture. *Protocols and Profiles*, v. 10, 2005.

CHADWICK, D. Federated identity management. *Foundations of Security Analysis and Design V*, Springer, p. 96–120, 2009.

COMMUNITY, J. *Jasig Central Authentication Service (CAS)*. 2012. http://www.jasig.org/cas. Acessado em: .

HUANG, F.; WANG, C.-x.; LONG, J. Design and implementation of single sign on system with cluster cas for public service platform of science and technology evaluation. In: IEEE. *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*. [S.l.], 2011. p. 732–737.

JOIE, C. L.; CANTOR, S. *Shibboleth Documentation: IdpCluster and NativeSPClustering*. 2012. https://wiki.shibboleth.net/confluence/display/SHIB2/Productionalization. Acessado em: .

KIRST, M. M. M. *JA-SIG Java Client Simple WebApp Sample*. 2012. https://wiki.jasig.org/display/CASC/JA-SIG+Java+Client+Simple+WebApp+Sample. Acessado em: .

LEANDRO, M. et al. Multi-tenancy authorization system with federated identity for cloud-based environments using shibboleth. In: *ICN 2012, The Eleventh International Conference on Networks*. [S.l.: s.n.], 2012. p. 88–93.

LIU, S.; WEN, Q. Distributed cluster authentication model based on cas. In: IEEE. *Broadband Network and Multimedia Technology (IC-BNMT), 2011 4th IEEE International Conference on*. [S.l.], 2011. p. 46–50.

MELL, P.; GRANCE, T. The nist definition of cloud computing. *NIST special publication*, v. 800, p. 145, 2011.

MIRKOVIC, J. et al. *Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)*. [S.l.]: Prentice Hall PTR, 2004.

MORGAN, D. M. R. *SIGAR - System Information Gatherer And Reporter*. 2010. note=https://www.oasis-open.org/committees/security/docs. Acessado em: ,. Acessado em: .

NANDA, M.; KHANAPURKAR, A.; SAHOO, P. High availability and scalable application clustering solution for a large-scale oltp application. In: IEEE. *India Conference (INDICON), 2011 Annual IEEE*. [S.l.], 2011. p. 1–5.

OLDEN, E. Architecting a cloud-scale identity fabric. *Computer*, IEEE, v. 44, n. 3, p. 52–59, 2011.

RANDLES, M.; LAMB, D.; TALEB-BENDIAB, A. A comparative study into distributed load balancing algorithms for cloud computing. In: IEEE. *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*. [S.l.], 2010. p. 551–556.

RICHARDSON, L.; RUBY, S. *RESTful web services*. [S.l.]: O'Reilly Media, 2008.

SHARMA, S.; SINGH, S.; SHARMA, M. Performance analysis of load balancing algorithms. *World Academy of Science, Engineering and Technology*, v. 38, p. 269–272, 2008.

STECCA, M.; BAZZUCCO, L.; MARESCA, M. Sticky session support in auto scaling iaas systems. In: IEEE. *Services (SERVICES), 2011 IEEE World Congress on*. [S.l.], 2011. p. 232–239.

TERRACOTTA, I. *The Definitive Guide to Terracotta: Cluster the JVM for Spring, Hibernate and POJO Scalability: Cluster the JVM for Spring, Hibernate and POJO Scalability*. [S.l.]: Apress, 2008.

WINCH, R. *Spring Security CAS Authentication*. 2012. http://docs.spring.io/spring-security/site/docs/3.0.x/reference/cas.html. Acessado em: .

ZHOU, M. et al. Services in the cloud computing era: A survey. In: IEEE. *Universal Communication Symposium (IUCS), 2010 4th International*. [S.l.], 2010. p. 40–46.