Towards Scalability for Federated Identity Systems for Cloud-Based Environments

André Albino Pereira João Bosco M. Sobral Carla M. Westphall
Universidade Federal de Santa Catarina
Programa de Pós-Graduação em Ciência da Computação
Florianópolis, Brasil, 88040-900,
andreptb@gmail.com, bosco.sobral@ufsc.br, carla.westphall@ufsc.br

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 distributed 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: scalability, federated identity, cloud computing, authentication, access control.

1 Introduction

The provisioning of services for the cloud computing [1] establishes that computational resources can be hired on-demand, improving operational efficiency and reducing costs for organizations that adopt this paradigm [2]. However, on environments involving sensitive data, mechanisms of authentication and access management must evolve so that cloud computing become a trusted platform, allowing full adoption by organizations [3]. Identity and access management systems must support cooperation between organizations, mainly to provide features such as *Single Sign-On* (SSO). Identities used within this context are denominated *federated identities* [4]. *Shibboleth* [5] and *Central Authentication Service* (*CAS*) implements federated identity [6], and [7] presents an infrastructure of federated identity for services on the cloud, based on the aforementioned technologies. As cited by [8], regardless if hired resources on the cloud have low (*Amazon EC2*) or high (*AppEngine*) abstraction level, technologies in hardware and software must all focus on horizontal scalability support rather than a high performance central node. Specifically for solutions based on emphShibboleth and *CAS*, horizontal scalability can be achieved by clustering identity and service providers. Since the authentication process on these solutions relies on *HTTP* session mechanism, the data stored on the server needs to be shared between all nodes of the cluster. In [9] is recommended the use of a distributed memory platform to address this requirement. This technique however can considerably increase the solution complexity, as well the cost for the necessary infrastructure. The objective of this paper is to provide an alternative for authentication and access session management, based on an adaptation of the *sticky-session* concept [10]. The *sticky-session mechanism*, supported by most load balancers, aims to redirect user subsequent requests to the same node the first request was made, therefore "sticking" the user to that node while a session context exists. For this to work, load balancers demand that the value of *HTTP Cookie* used to establish session with the user appends at the end of the *String* an unique value that identifies the node which created the session context. For Java systems, application servers such as *Apache Tomcat* and *JBossAS* uses a *HTTP Cookie* named *JSESSIONID* for this end. The identifier suffix in these application servers are called *jvmRoute* identifier. This alternative, provided by [10], aims to minimize the costs and complexity to deploy a federated identity management system for the cloud, improving horizontal scalability support for the components of this system. This paper is organized with the following sections: Section 2 introduces key aspects related to this work, such as cloud computing models and identity

management as a service; Section 3 describes the related work; Section 4 lists the technologies used on a federated identity management platform, as well as it's horizontal scalability support; Section 5 presents a new approach to improve scalability on the components used on a federated identity management for cloud computing; Section 6 provides test results and comparative analysis with other existent alternatives; Finally, Section 7 presents the conclusions and future work.

2 Basic Concepts

In this section, key aspects regarding the contributions provided by this work are fundamented.

2.1 Cloud Computing Infrastructure for Web Applications

Cloud computing is a model in which computational resources, such as servers, application, among others, are hired on-demand. These resources should be provided quickly, without much hassle from the service provider. It is indispensable that technologies used on those environments are horizontally scalable [8], which is accomplished by integrating multiple software or hardware entities to work as one logical unit. For servers, adding new nodes will improve the performance of this logic unit, using mechanisms such as clusterization and load balancing [8]. On the other hand, vertical scaling aims to increase processing power of a single entity, like adding memory to a server.

For Web application cloud environment, such as *Amazon's EC2 IaaS (Platform as a Service)* approach, on demand horizontal scalability is provided with *Amazon's ELB (Elastic Load Balancing)* technology. Other vendors provide similar services (*Microsoft AAB* and *Google Cloud Auto Scaling Orchestrator*). The idea is to automatically distribute incoming application traffic, initializing additional or shutting down hardware instances according to this traffic, as depicted on Figure 1.

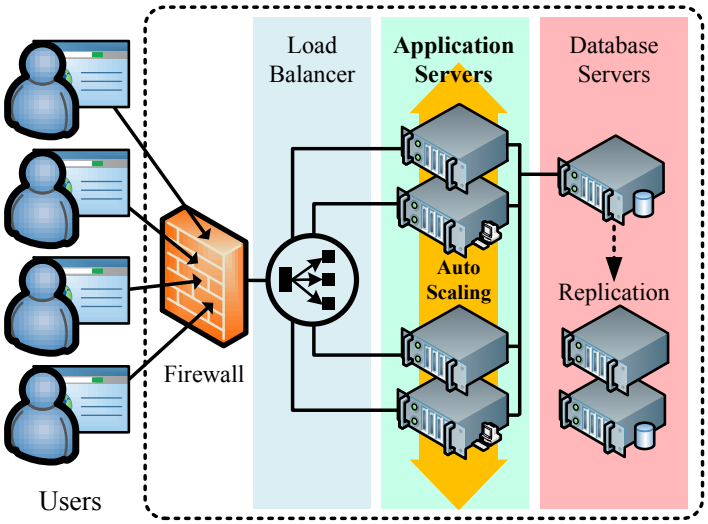


Figure 1: Web application cloud environment with elastic load balancing

However, as stated by [8], software elements must be compliant with auto scaling approaches, and most authentication solutions, such as *Shibboleth*, requires adaptations, and as will be discussed on the following sections.

2.2 Identity Management as a Service

A digital identity is a representation of an entity (or group of entities) in the form of one or more elements of information (attributes) that allows the recognition of an entity in a specific context [4]. A identity management system aggregates a collection of tools to manage individual identities in a digital environment [4] [11]. A feature largely utilized on these systems includes *SSO*, so the user does not need to authenticate every time to access different applications. The responsibilities of an identity management system can categorized in the following items:

- Authentication: Asserts that the user is who he claims to be, using mechanisms such as password, biometry, digital certificate, among others.

- Authorization: Access control in different levels, features and operations within a computational system.
- Federation: When a group of organizations share identity information between it's users in a trusted way [12].

Considering that organizations can provide services of different segments on the cloud, it is recommended to promote the separation of identity management tasks in a single service [3]. This model presents two components:

- Identity Provider *IP*: Service responsible to authenticate users and provision credentials information to registered services that requires it.
- Service Provider *SP*: Provides the features that the user consumes. If it's access is restricted, the *IP* must be queried to collect credentials.

For organizations that develop *IP* it means less preoccupation with identity technology, allowing more investment on the service management and less on security infrastructure [3]. Also, this approach is imperative for *SSO* mechanisms in a federated identity environment.

3 Related Work

The related work of this paper is separated in two groups. The first aims to consolidate identity management and access control in a cloud environment, providing theoretical models and practical implementation for this purpose. The second addresses the session management between user and services on the cloud, a mechanism largely adopted on identity management technologies.

3.1 Access control in a cloud environment related work group

In [13], an authorization model is established, providing means to secure resources on a *SaaS* (*Software-as-a-Service*) cloud. At each request, a new authorization process is performed. This approach is recommended for *REST Web Services* based solutions, since *RESTful* approaches should not make use of *HTTP session* [14]. In [3], the challenges to secure services on the cloud are discussed, and the notion of identity management as a service is presented, as well as the requirements to build a solution for this segment. In [7], a federated identity management architecture for the cloud is proposed. A scenario is structured in a way that services are deployed on virtual machines provided by *Amazon EC2*, and authentication providers in a separated third party environment. The proposal is based on *Shibboleth* and *CAS*, however the horizontal scalability support of the solution is not addressed.

3.2 Session management in cloud services related work group

In [10], a study addressing the scalability of services deployed on an IaaS (Infrastructure-as-a-Service) cloud is made, in which the use of *HTTP session* mechanism is required. The work presents a scenario with clustered services, and a load balancer forwards user requests while applying *sticky-session*. To aggregate robustness to the solution, the authors provides monitoring and failsafe mechanisms in case a cluster node becomes unavailable, situation that, if not treated, can result *HTTP session* data lost on the server and consequently user's work. In [15], a viability analysis is made regarding the deployment of large scale services on the cloud, comparing *sticky-session* and *distributed memory* approaches. The author concludes that, considering the test results, the *distributed memory* is the best approach. However, the analyzed infrastructure do not consider the existence of federated identity management mechanisms. In [9], *CAS* scalability issue is addressed by specifying a designing cluster authentication model, providing a load balancer to manage multiple *CAS Server* instances. The author does not address the scalability issues on client side, specifically *Single-Sign-Out* support. The work on the paper is preliminary and a follow-up work, possibly an implementation, is to be expected. In [16], an authentication server controller is implemented, managing *CAS Server* cluster nodes and balancing requests accordingly. As with [17], the scalability issues on client side are not addressed. Comparing the work exposed on [17] and [16] with this paper, resolving *CAS Server* scalability issues is a common goal, but this paper also addresses the scalability problems regarding client applications that participates in a *CAS* authentication infrastructure. Additionally, no modifications were applied on *CAS* authentication protocol, ensuring compatibility when the solution is embedded in the proposed *Shibboleth* architecture for the cloud [7].

4 Identity Management Technologies

A trusted model for federated identity management and adherent to [3] recommendation was proposed by [7], and was based on two open-source technologies:

- *JASIG Central Authentication Service* (*CAS*): Provides components to act as an authentication service, known as *CAS Server* [6]. The solution implements a *HTTP* based protocol for *SSO*, and provides client tools, in various platforms, so systems can participate in the authentication infrastructure.
- *Shibboleth*: Implements an architecture for contexts involving federated identity management. Provides mechanisms to safely transfer user credentials, as well as been highly adherent to *W3C* specifications, such as *Security Assertion Markup Language* (*SAML*) [5].

For the scenario implemented by [7], *CAS* act as *SSO* authentication mechanism in the *Shibboleth* architecture.

4.1 *CAS* Authentication Flow

Considering that *CAS* is an important component in the federated identity management infrastructure, the *SSO* authentication flow is detailed in this section. For services accessed by the user's browser, *CAS* implements a protocol which make use of *HTTP* mechanisms such as *Cookies* and *Redirects*. The operation flow established on this protocol is exposed on Figure 2, and is composed by eight steps:

1. The user access a service's resource using a web browser.
2. Since the service demands credentials, it responds *HTTP 302 (Redirect)* so that the browser takes the user to the *CAS Server*, to proceed with the authentication.
3. If no previous authentication occurred, the user will be asked to inform credentials (user and password, digital certificate, etc). If is already authenticated on *CAS Server*, the flow skips to the next step, without asking the user to inform his credentials again.
4. If the authentication process succeeds, the *CAS Server* will generate an unique identifier (a *ticket*). This identifier is appended to the redirect *URL* the user accessed previously (the service resource). *CAS Server* keeps the *ticket* in memory for a limited time, so can be processed in step 7.
5. The browser proceeds with the new redirect, this time with the *ticket* appended on the *URL*.
6. The existence of a *ticket* on the *URL*'s query string means that the user is authenticated on *CAS Server*, so the service itself must request the *CAS Server* to validate the authentication context.
7. Upon receiving the service's request, *CAS Server* compares the *ticket* extracted from the *URL* with the *ticket* kept in memory, responding a *SAML* message with the user's credentials.
8. After processing *CAS Server's* response, the service commits the authentication process and returns to the user the content requested in step 1.

Additionally, *CAS* offers *Single-Sign-Out* [18], allowing the user to logout from all participant services with a single request. When a logout request is received from the user, the *CAS Server* sends a *SAML* message to each service accessed by the user so the authentication state can be synchronized (Figure 3).

4.2 Horizontal Scalability Support

To horizontally scale systems based on *CAS* components, clusterization must be enabled, both for the *IP* as for the services providers. A load balancer (*LB*) approach must be deployed so requests can be coordinated to the nodes. There are several solutions for load balancing tasks, such as *Apache Web Server*, *Microsoft IIS*, *Amazon EC2 ELB*, among others [19]. The user authentication establish a *HTTP session* with the server, so that the authentication process do not repeat at each access. Since the data of this session are kept in the memory of the node the user accessed, if in the next access the *LB* forwards the request to another node, no session information will be found, mistakenly demanding a new authentication process. This is a recurring problem in *stateful* approaches, such as *CAS* and *Shibboleth*.

A possible solution is to ensure the *LB* forwards subsequent requests to the same node from the first request [10]. This mechanism is known as *sticky-session*, and uses *HTTP Cookies* created to mantain state between system and user.

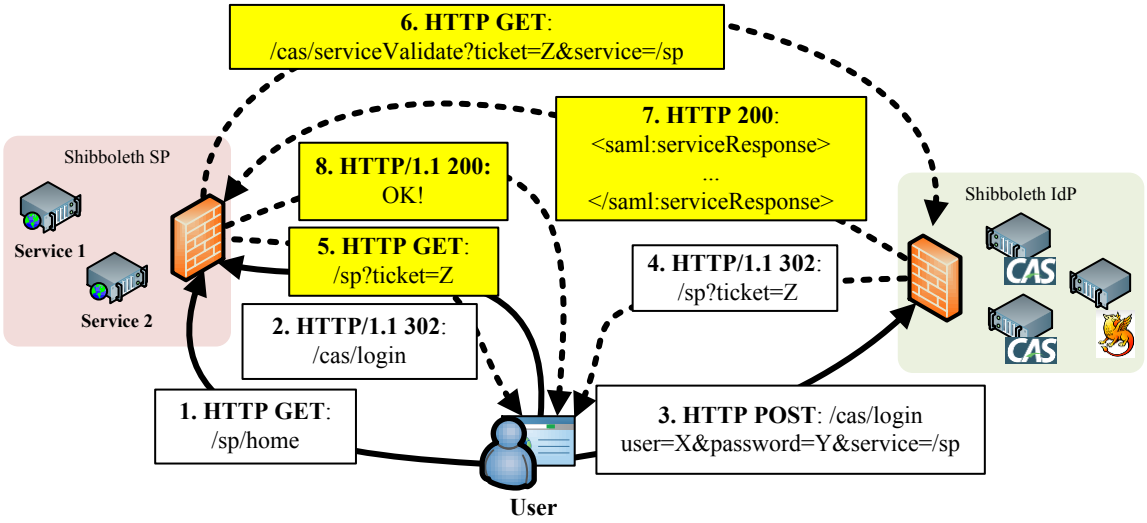


Figure 2: *CAS* authentication flow

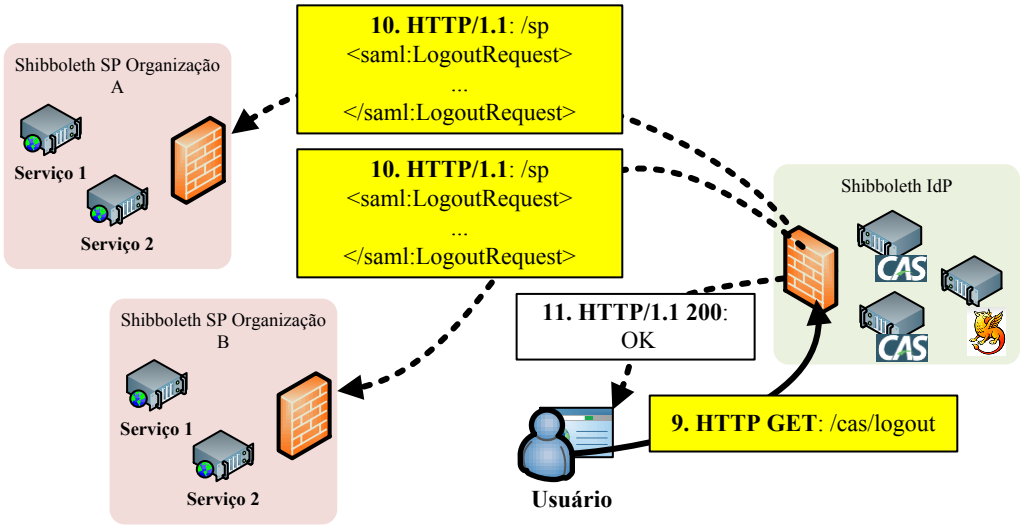


Figure 3: *CAS* Single-Sign-Out operation flow

However, as exposed by [9], the communication step between the service and *CAS Server* also requires the *CAS Server's* node is the same the user accessed, since *ticket* is kept in the memory of that specific node. In this step, the *LB* do not receive the user's *HTTP Cookies*, since the request was made by the service. The *sticky-session* mechanism will not work, and the authentication process may fail. In a cloud computing environment which adopts identity management as a service, a scalable infrastructure will inevitably be compromised by this issue (Figure 4).

The *Single-Sign-Out* is also compromised, since when *CAS* process the *LogoutRequest SAML* message to the services, the *LB* for the same reason may forward requests to *cluster* nodes of services that the user wasn't authenticated (Figure 5).

4.3 Distributed memory with Terracotta

An alternative to address the aforementioned issues is to consider the use of *Terracotta* [20]. This approach was recommended by [15], [9] and [21]. *Terracotta* provides memory information sharing between all nodes, making user's session information available to any cluster's node. This infrastructure uses three *Terracotta* components [20]:

- *Terracotta Server*: A dedicated service to maintain the information to be shared. Can also be clustered in a *master/slave* fashion, providing better performance and fail-safe features.
- *Web Sessions*: Provides *APIs* so application servers can use *Terracotta Server* to share and access memory information.

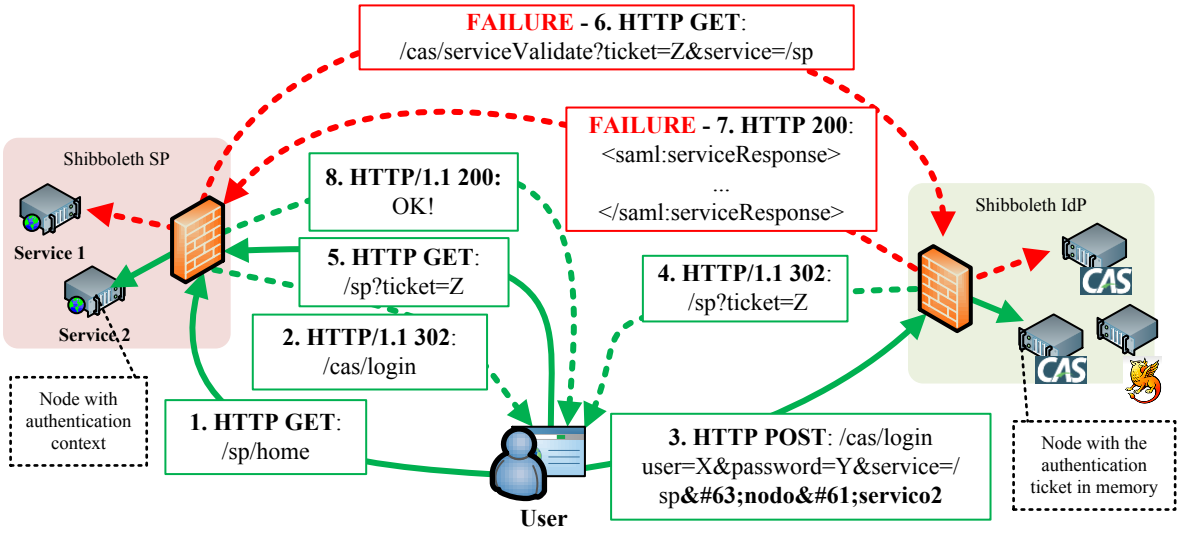


Figure 4: Failure simulation on the authentication process of horizontally scaled services on the cloud

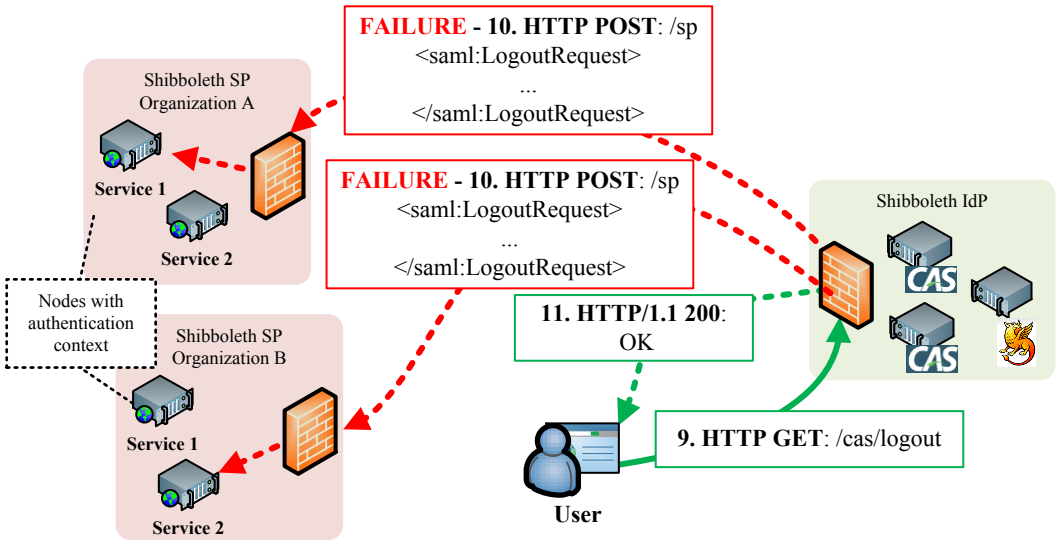


Figure 5: Failure simulation on the *Single-Sign-Out* process of horizontally scaled services on the cloud

- *EhCache*: *Java APIs* to provide cache features and easy mechanisms to use *Terracotta Server* to share information. It is used on *CAS Server* to make *tickets* produced on the auhtentication flow to be available to all cluster nodes.

5 Proposal of an Adapted Sticky-Session Approach to Provide Scalability for Services on the Cloud Using Federated Identities

The increase of complexity to deploy the resources used in a infrastructure with *Terracotta* can elevate costs to the organization. For *Shibboleth* and *CAS* based solutions, the shared information using *Terracotta* presents concerns at a security level, since the content of this information will be exposed on *Terracotta Server* nodes. This section proposes a new alternative without requiring distributed memory mechanisms. The proposal adapts *CAS* components so the authentication flow become compatible with *sticky-session* mechanism. The section 5.1 details how this proposal works, while 5.2 presents test scenario, built to vaildate and compare this proposal with the *Terracotta's* one.

5.1 The Proposal Details

The application that manages the *Cookie* life cycle must append the node identifier every time a session is created, allowing the *LB* to work properly. In *Java* based systems this can be configured in most application servers available to the platform. As detailed previously, the *HTTP Cookies* dependency that the *LB* have

to forward requests properly at first makes the use of *sticky-session* unfeasible with *CAS* solution. However, this can be addressed through the following adaptations:

- *CAS Server*: The *ticket's* value generated on step 6 of Figure 2 must append the *jvmRoute* identifier from *JSESSIONID*. *CAS Server* was modified to support this configuration (Figure 6).
- *LB* configuration: The *sticky-session* rule based on *JSESSIONID Cookie* must be replicated to also consider the ticket's *jvmRoute* value, which was send by the service (as described on step 6 of Figure 2). The *Apache Web Server* was modified to support this configuration.

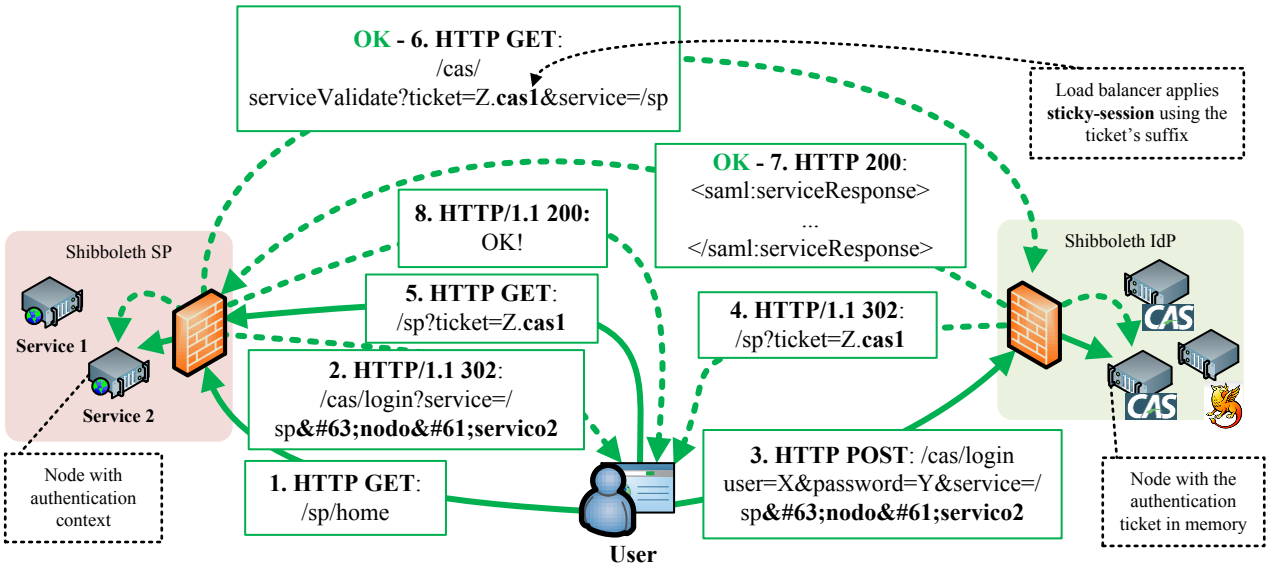


Figure 6: *CAS Server* generates the *ticket* with the node identifier

The *Single-Sign-Out* can also be used with *sticky-session*, addressing the issue described on Section 4.2. The following adaptations must be done:

- The addition of an identifier on the *URL's* parameter named *service* (Figure 6): The *service* parameter informed by the service (step 6 of Figure 2 contains the *URL* that *CAS Server* will use to send *logout* requests. This *URL* must be changed to contain the *jvmRoute*, allowing the *LB* to forward the *logout* request correctly. The the client component provided by *CAS* was modified to support this configuration.
- Service provider's *LB* configuration (Figure 7):

5.2 Test Scenario

In order to validate the adapted *sticky-session* proposal and compare with the *Terracotta* alternative, a scenario was built using virtual machines. Two groups or virtual machines are deployed in different networks, simulating resources in the cloud from different providers. The first group (*VM1* and *VM2*) contains the service's cluster, while the second (*VM3* and *VM4*) the *CAS Servers'* cluster, as shown on Figure 8.

The virtual machines were distributed in a single host server, with the following specifications: *Phenom II X6 1055T (2.8Ghz)* processor, *8192MB DDR3* memory and a *SAMSUNG HD154UI 1.5Tb* hard disk. *VM2* and *VM3* were configured to work 2048MB memory, while *VM1* and *VM4* were configured with 1024MB. *CPU* and *hard disk storage* were equally distributed between all *VMs* (*2x Core 2.8Ghz for CPU* and *10Gb for hard disk storage*). Another computer with similar specifications of the host server was used to simulate users' requests, using *Apache JMeter*. The equipments were connected by a *LAN* network of *1 Gbit/s* bandwith. So the adapted *sticky-session* can be validated in this scenario, when the modifications detailed on section 5 are applied, the *Terracotta* platform is disabled for both virtual machine groups. In both alternatives the expected behavior of the authentication process must be equivalent. For each configuration model (*sticky-session* and *Terracotta*), a stress test procedure was executed using *Apache JMeter*. This test procedure simulates a group of simultaneous users, each triggering an authentication process (detailed in section 4.1), followed by a *logout* if the previous authentication completed successfully. For each test iteration, metrics such as server computational load and *throughput* are collected, providing data for a benchmark analysis of the two solutions.

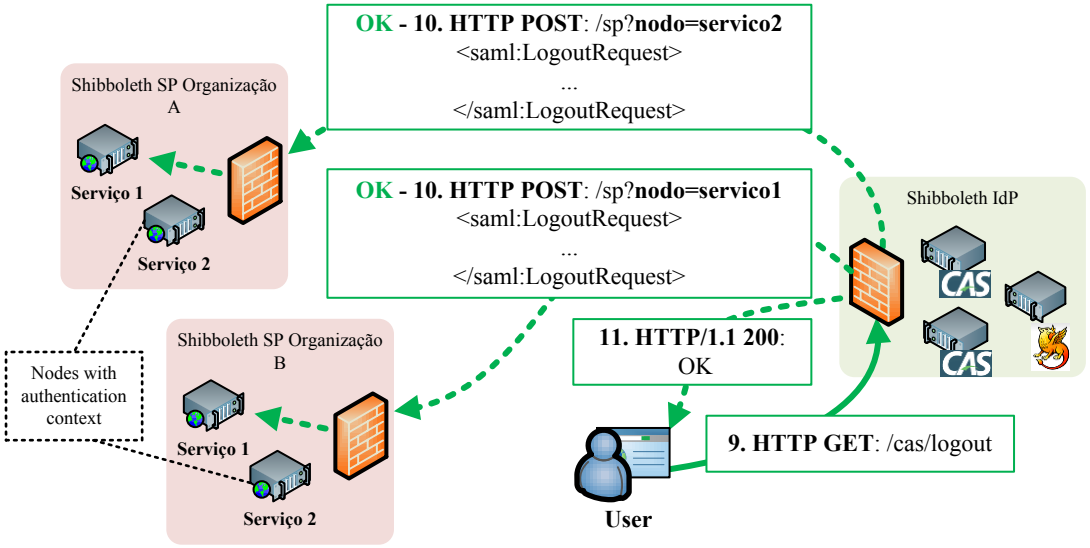


Figure 7: Service cluster's *LB* configuration to consider the new parameter

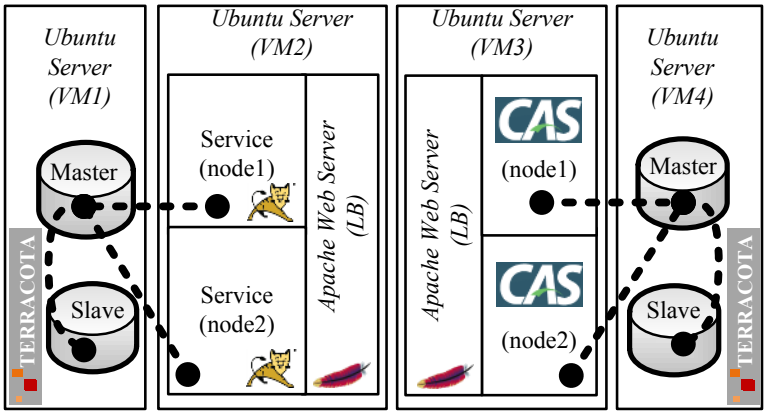


Figure 8: Scenario to simulate the alternatives available to provide horizontal scalability for *CAS* infrastructures

6 Test Results Comparative Analysis

This section presents a comparative analysis of the two proposals that addresses horizontal scalability issues for a federated identity management architecture for the cloud. Considering the designed scenario, the test procedure was executed in three different configurations: The first with 50, the second with 100 and the last with 150 simultaneous users. The request number for each configuration results in, respectively, 20000, 40000 and 60000 requests. The first aspect compared was the *throughput* (requests per second) obtained for each alternative. Both approaches had adequate performance, however in the adapted *sticky-session* approach, the *throughput* was superior in Figure 9.

The *Terracotta* approach presented failures in a few test iterations, with an average of *0,33%* failures per test iteration. All occurrences are associated with the *ticket* validation process between the service and *CAS Server*. Further analysis shown that in specific moments the *VM1* memory reached 100% usage, making *Terracotta Server* use the hard disk to store information, which caused higher latency of data transfer between *VM1* and *VM2*, to a point that the authentication process couldn't be completed. The second aspect analyzed computational resource usage for each approach. While the tests procedure were executed, at each second the *CPU* and memory usage were collected.

Comparing the average usage of computational resources (Figures 10 through 11), it is clear that the *Terracotta* demands more resources. Moreover, this infrastructure required two additional virtual machines (*VM1* and *VM4*), to host dedicated *Terracotta Servers*.

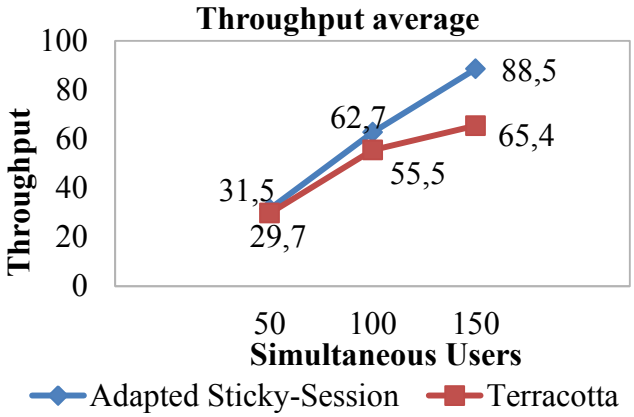
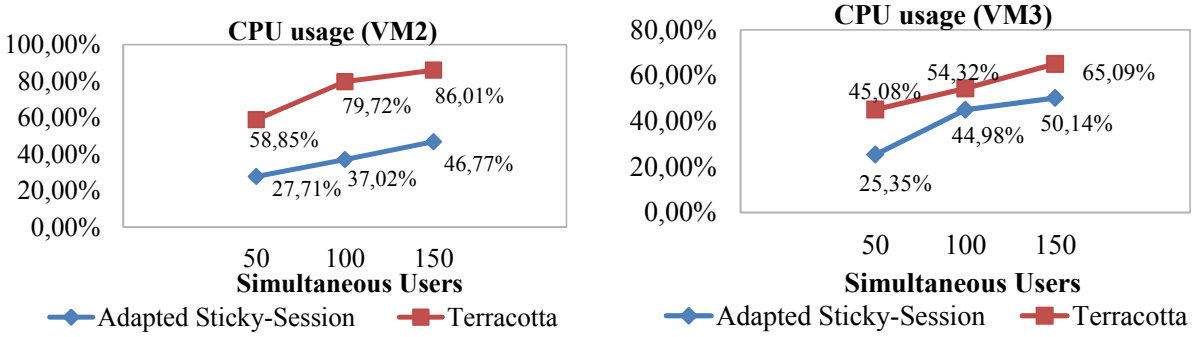


Figure 9: Comparative graphic showing average *throughput*



(a) *VM2* CPU usage

(b) *VM3* CPU usage

Figure 10: Comparative graphic showing *VM2* and *VM3's* CPU usage

7 Conclusions and Future Work

This work aimed to mitigate the technical difficulties to deploy horizontal scalable services in a federated identity management platform for the cloud. We proposed the use of an adaptation of the *sticky-session* mechanism, providing an alternative to a distributed memory approach [9], reducing costs of service deployment, depending of the business nature. However, this proposal have limitations inherited from the *sticky-session* mechanism, such as dynamic scalability and session management, as described by [10]. Considering that services in the cloud requires horizontal scalability [8], this work contributed towards this goal, aggregating this support on the platform's components proposed by [7]. As first future work, the infrastructure proposed by [7], coupled with the *sticky-session* mechanism proposed by this paper, could be deployed in a real cloud computing environment, so statistic data can be collected and analyzed, as well as compared with the *Terracotta* approach. Another future work can address the dynamic scalability limitations and session management, by applying monitoring and migration mechanisms [10] in this infrastructure.

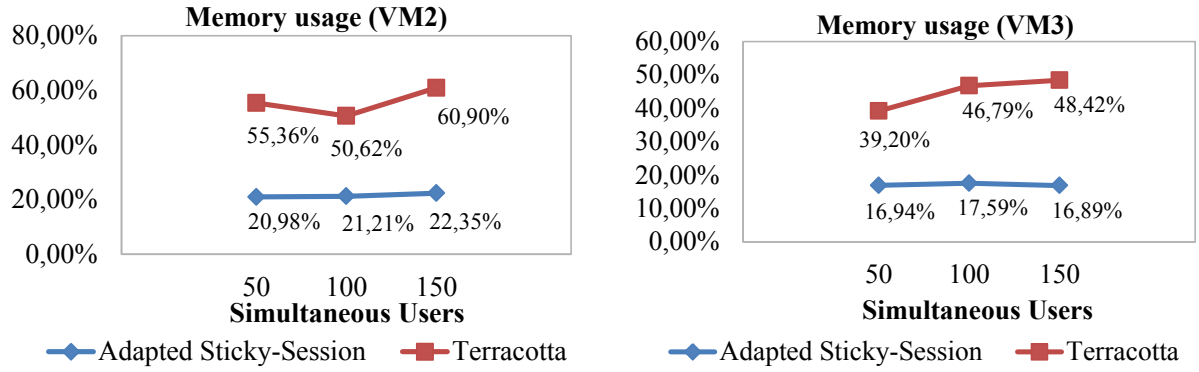
Acknowledgment

This document was extracted from the master's thesis of the first author, titled "**Escalabilidade para Sistemas de Indentidade Federada para Ambientes baseados em Computação em Nuvem**", as a student of the Graduate Computer Science Program at Federal University of Santa Catarina on February 2014.

References

[1] P. Mell and T. Grance, "The nist definition of cloud computing," *NIST special publication*, vol. 800, p. 145, 2011.

[2] M. Zhou, R. Zhang, D. Zeng, and W. Qian, "Services in the cloud computing era: A survey," in *Universal Communication Symposium (IUCS), 2010 4th International*. IEEE, 2010, pp. 40–46.



(a) *VM2* memory usage

(b) *VM3* memory usage

Figure 11: Comparative graphic showing *VM2* and *VM3's* memory usage

[3] E. Olden, "Architecting a cloud-scale identity fabric," *Computer*, vol. 44, no. 3, pp. 52–59, 2011.

[4] D. Chadwick, "Federated identity management," *Foundations of Security Analysis and Design V*, pp. 96–120, 2009.

[5] S. Cantor, S. Carmod, M. Erdos, K. Hazelton, W. Hoehn, R. Morgan, T. Scavo, and D. Wasley, "Shibboleth architecture," *Protocols and Profiles*, vol. 10, 2005.

[6] J. Community. (2012) Jasig Central Authentication Service (CAS). http://www.jasig.org/cas. Acessado em: November 20, 2015.

[7] M. Leandro, T. Nascimento, D. dos Santos, C. t. r. b. Westphall, and C. Westphall, "Multi-tenancy authorization system with federated identity for cloud-based environments using shibboleth," in *ICN 2012, The Eleventh International Conference on Networks*, 2012, pp. 88–93.

[8] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[9] S. Battaglia and B. Savage. (2012) Jasig CAS Documentation: Clustering CAS. https://wiki.jasig.org/display/CASUM/Clustering+CAS. Acessado em: November 20, 2015.

[10] M. Stecca, L. Bazzucco, and M. Maresca, "Sticky session support in auto scaling iaas systems," in *Services (SERVICES), 2011 IEEE World Congress on.* IEEE, 2011, pp. 232–239.

[11] D. R. D. Santos, T. J. Nascimento, C. M. Westphall, M. A. P. Leandro, and C. B. Westphall, "Privacy–preserving identity federations in the cloud: a proof of concept," *International Journal of Security and Networks*, vol. 9, no. 1, pp. 1–11, 2014.

[12] D. R. d. Santos, C. M. Westphall, and C. B. Westphall, "Risk-based dynamic access control for a highly scalable cloud federation," in *SECURWARE 2013, The Seventh International Conference on Emerging Security Information, Systems and Technologies*, 2013, pp. 8–13.

[13] J. M. A. Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray, "Toward a multi-tenancy authorization system for cloud services," *Security & Privacy, IEEE*, vol. 8, no. 6, pp. 48–55, 2010.

[14] L. Richardson and S. Ruby, *RESTful web services.* O'Reilly Media, 2008.

[15] M. Nanda, A. Khanapurkar, and P. Sahoo, "High availability and scalable application clustering solution for a large-scale oltp application," in *India Conference (INDICON), 2011 Annual IEEE.* IEEE, 2011, pp. 1–5.

[16] F. Huang, C.-x. Wang, and J. Long, "Design and implementation of single sign on system with cluster cas for public service platform of science and technology evaluation," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on.* IEEE, 2011, pp. 732–737.

[17] S. Liu and Q. Wen, "Distributed cluster authentication model based on cas," in *Broadband Network and Multimedia Technology (IC-BNMT), 2011 4th IEEE International Conference on.* IEEE, 2011, pp. 46–50.

[18] S. Battaglia and B. Savage, "Jasig cas documentation : Single sign out," 2012, https://wiki.jasig.org/display/CASUM/Single+Sign+Out. Acessado em: November 20, 2015.

[19] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," in *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on.* IEEE, 2010, pp. 551–556.

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

[21] C. La Joie and S. Cantor. (2012) Shibboleth Documentation: IdpCluster and NativeSPClustering. https://wiki.shibboleth.net/confluence/display/SHIB2/Productionalization. Acessado em: November 20, 2015.