CloudGI - Gerenciando Instâncias de um Serviço Replicado em uma Plataforma de Computação em Nuvem OpenStack

Poliana S. Nascimento*
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Campus de Salvador, Rua Emídio dos Santos, 40.301-015 - Salvador, Bahia, Brasil
polianasantos@ifba.edu.br

Allan E. S. Freitas†
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Campus de Salvador, Rua Emídio dos Santos, 40.301-015 - Salvador, Bahia, Brasil
allan@ifba.edu.br

ABSTRACT
Nowadays, cloud computing platforms have been used to provide computing services, allowing on-demand rent of computational resources. However, even cloud computing-based services may fail, and replication mechanisms shoud be used, providing an adequate number of replicas to run the service.

In this work, we present CloudGI , a replica manager for OpenStack cloud computing platform. Such tool allows for deploy a replicated service, instantianting N replicas for tolerate F faults: crash-recovery or bizantine – according to a prescribed service level. CloudGI monitors replicas, indicate faults and act under fault replicas for fixing them. Also, it implements periodic replica rejuvenation, in order to pro-actively minimizing the number of possible compromised replicas.

Keywords
Cloud Computer, DevStack, Distributed Systems, Fault Tolerance, OpenStack

RESUMO
Plataformas de computação em nuvem têm sido amplamente utilizadas para o provimento de serviços computacionais nos dias atuais, permitindo o dimensionamento de recursos sob demanda e a escalabilidade adequada no uso de serviços. Contudo, mesmo serviços providos por meio de um ambiente de Computação em Nuvem estão sujeitos a falhas, e mecanismos usuais de replicação podem ser utilizados, instanciando um conjunto de réplicas adequado para a manutenção do serviço.

*Graduanda em Análise e Desenvolvimento de Sistemas
†Prof. D.Sc. Membro da SBC e ACM. Instituto Federal da Bahia GSORT - Campus Salvador.

Neste artigo, apresentamos a ferramenta CloudGI, um gerenciador de réplicas para o ambiente de computação em nuvem *OpenStack*. Por meio desta ferramenta, pode-se implementar um serviço replicado, instanciando N réplicas para tolerar F falhas – *crash-recovery* ou bizantinas, de acordo com o nível de serviço estabelecido. CloudGI monitora o funcionamento das réplicas, indica falhas e atua sob as réplicas falhas para re-estabelecer o funcionamento. Ainda, implementa o rejuvenescimento periódico de réplicas sujeitas a falhas bizantinas, de modo a minimizar o número de réplicas comprometidas.

Palavras Chaves
Computação nas nuvens, *DevStack*, Sistemas Distribuídos, Tolerância a falhas, OpenStack

1. INTRODUÇÃO
Com o avanço constante das tecnologias da informação, a necessidade do desenvolvimento de ferramentas e técnicas - que auxiliem nos processos de gerenciamento, armazenamento, distribuição e acesso aos dados e aplicações -, vem se tornando cada vez mais necessárias. Diante disto, a computação em nuvens tem sido um poderoso mecanismo para atender a essas questões, por meio de distribuição no processamento e armazenamento de serviços e dados. Os recursos mantidos em nuvem devem ter garantias de disponibilidade, integridade, segurança e confiabilidade. Entretanto, mesmo sistemas computacionais em nuvem estão sujeitos a falhas. Falhas estas, que podem acarretar sérios problemas para as organizações, ocasionando perda de dados, perda de disponibilidade dos serviços, furto de informações, dentre outros.

Muitas técnicas sugiram com o objetivo de mitigar e tolerar estas falhas, tais como *backup* dos dados, espelhamento de discos e redundância de equipamentos. Uma solução que tem acarretado grandes benefícios é a de replicação de componentes em um sistema distribuído, [10].

A solução de replicar máquinas e dados para garantir confiabilidade, integridade, segurança e disponibilidade tem sido muito utilizada. O gerenciamento de réplicas, tanto em máquinas físicas, quanto por meio de máquinas virtuais hospedadas em nuvens computacionais, tem sido um assunto bastante discutido nos dias atuais. Trabalhos como [23],

[24], [25] abordam o gerenciamento de réplicas.

O gerenciamento de réplicas deve considerar o cenário de falhas que deva ser tolerado, tais como: falhas por crash e falhas bizantinas. Falhas estas que comprometem a confiabilidade, segurança e disponibilidade dos serviços.

Diante disto, apresenta-se a ferramenta CloudGI. A aplicação desenvolvida tem por objetivo monitorar, gerenciar e criar instâncias de maquinas em um ambiente de nuvem *OpenStack*, com foco na gerência de réplicas que provêm à tolerância a falha em sistemas distribuídos. A mesma complementa a atuação de middlewares de replicação na tolerância a um dado número de réplicas falhas por crash ou de forma arbitrária (bizantina), conforme um nível de serviço previamente estabelecido (aqui denominado ouro, prata ou bronze).

Para garantir a cobertura dessas falhas, através de comandos de gerenciamento do serviço *OpenStack*, o CloudGI monitora periodicamente todas as instâncias, visando garantir que as mesmas estejam sempre em estado ativo. Caso uma réplica venha apresentar algum problema, a aplicação tomará as devidas providências para reaver a instância.

A aplicação em questão conta com dois módulos: a aplicação web, voltada para o cliente, fornece os serviços de inicialização e visualização das instâncias; o módulo desktop é voltado para monitorar, gerenciar e visualizar as réplicas, garantido que sejam identificados problemas com instâncias para que sejam tomadas as devidas soluções.

O presente trabalho está organizado como exposto abaixo:

- A Seção 2 apresenta conceitos básicos a cerca de computação em nuvens;
- A Seção 3 apresenta conceitos referentes à tolerância a falhas em sistemas distribuídos. Incluindo subseções sobre modelos de sistemas síncronos e assíncronos e falhas por *crash*, *crash recovery* e *bizantinas*;
- A Seção 4 discorre sobre o projeto *OpenStack*, com uma subseção sobre o *shell script devstack* utilizado para instalação do ambiente em nuvem;
- A Seção 5 expõe os trabalhos correlatos;
- A Seção 6 apresenta a ferramenta CloudGI, expondo o caso de uso, a arquitetura, o modelo de base de dados, a implementação e as funcionalidades da mesma;
- A Seção 7 apresenta a avaliação de funcionalidades, expondo o ambiente de experimentação, cenários avaliados e resultados dos testes.
- A Seção 8 apresenta a conclusão e os trabalhos futuros.

2. COMPUTAÇÃO EM NUVENS

Com o crescimento dos serviços de *Internet* e a necessidade de estar conectado a todo o momento e poder acessar seus arquivos e aplicações em qualquer lugar, se tornou necessário um mecanismo que suportasse a grande demanda de dados e acessos a esses serviços de forma segura e eficiente. Diante disto, a computação nas nuvens foi uma tecnologia desenvolvida que possibilita acesso a programas, serviços e arquivos com o uso da *Internet*. Segundo [4], as nuvens são repositórios que contém recursos virtualizados que são fáceis de usar, são acessíveis e reconfiguráveis para atender uma demanda variável e para otimizar recursos. A computação em nuvem foi classificada como uma das melhores tecnologias da década [1]. De acordo com [2], é uma arquitetura emergente no qual os dados e os aplicativos residem no ciberespaço, permitindo aos usuários acessá-los por meio de qualquer dispositivo conectado à *web*. Computação em nuvem fornece uma infraestrutura de computação em rede que reduz os custos associados à gestão de recursos de *hardware e software*. Nuvens possibilitam a terceirização do fornecimento da infraestrutura de computação necessária para hospedar serviços. Segundo [3], é um modelo computacional onde os dados e os softwares residem e podem ser acessados por computadores pessoais e por dispositivos como *smartphones*, PDAs, aparelhos de informática, consoles de jogos e até mesmo carros.

A computação em nuvem é um modelo de infraestrutura comercial emergente que fornece para as empresas a possibilidade de eliminar *hardware in-house* de alto custo, *software* e infraestruturas de rede. Além disso, reduz ou até mesmo elimina o alto custo de recrutamento de profissionais técnicos para apoiar estas infraestruturas e operar as soluções de TI *in-house* [1]. Nos dias atuais, a mesma emerge como o paradigma adequado para hospedagem de infra-estrutura e serviços na *Internet*, em face de prover mecanismos adequados para provisionamento dinâmico de recursos, permitindo às organizações escalabilidade no uso de recursos, conforme a demanda dos seus usuários [34].

Existem três modelos onde as *clouds* são utilizadas: *Infrastructure as a Service (IaaS)*, onde os IPs *(Infrastructure Providers)* gerenciam um grande conjunto de recursos de computação e através da virtualização são capazes de dividir, ceder e dinamicamente redimensionar esses recursos para construir sistemas *ad-hoc* como exigido pelos clientes; *Platform as a Service (PaaS)*, no qual sistemas em nuvens podem oferecer um nível de abstração adicional podendo fornecer uma plataforma de software onde os sistemas podem ser executados; e *Software as a Service* (SaaS) que são serviços de potencial interesse para uma ampla variedade de usuários e que podem ser hospedados em sistemas de nuvem, sendo uma alternativa para executar aplicativos localmente. De acordo com [1] os requisitos para a melhoria dos serviços em nuvem são segurança, privacidade, disponibilidade, auditoria, a flexibilidade, o arquivamento, escalabilidade e qualidade de serviços (QoS) [4].

O uso de IaaS para a computação em nuvem permite às organizações a vantagem de manter máquinas virtuais que combinem diferentes plataformas de execução e serviços, nem sempre disponíveis sob a forma de PaaS ou de SaaS, bem como gerenciar de forma mais suave e flexível a alocação de recursos para cada máquina do que com a alocação de máquinas físicas. Isto pode ser obtido por meio da contratação de recursos em uma nuvem pública, como Amazon EC2 [36], ou ainda, através de uma infraestrutura própria ou compartilhada entre um número reduzido de organizações, em nuvens privadas, as quais podem ser construídas a partir

do uso de plataformas computacionais como *Eucalyptus* [32], *OpenNebula* [31] ou *OpenStack* [35].

3. TOLERÂNCIA A FALHAS EM SISTEMAS DISTRIBUÍDOS

Diante do crescimento dos recursos computacionais e do armazenamento de dados, se tornou necessário o desenvolvimento de técnicas que garantissem a disponibilidade e integridade das informações. Todos os sistemas são suscetíveis a falhas, causando assim, constantes prejuízos e problemas para grandes e pequenas organizações. Diante das vulnerabilidades, técnicas são desenvolvidas para tolerar falhas, garantindo a disponibilidade dos serviços e integridade das informações.

Simples técnicas como *backup* dos dados, redundância de equipamentos e espelhamento de discos, podem solucionar estes problemas, mas, muitas vezes, consomem recursos de hardware e diminuem o desempenho. Diante da popularização das redes, fornecendo diversos serviços, soluções mais robustas se tornaram necessárias para atenuar tais vulnerabilidades [10]. Sistemas Distribuídos tem sido uma solução para amenizar o problema de disponibilidade de serviços, integridade e armazenamento dos dados, entretanto se tornou necessário desenvolvimento de soluções cada vez mais complexas para tolerar falhas nestes ambientes.

Sistemas distribuídos consistem em uma coleção de dispositivos autônomos que se conectam através de redes e *middleware* distribuído. São construídos em cima de redes existentes e sistemas operacionais. Para que se tornem autônomos existe uma associação *master/slave* entre computadores e redes. O *middleware* permite que as máquinas coordenem as atividades e compartilhem os recursos do sistema para aparentar um mecanismo único de computação integrada [12].

Sistemas distribuídos podem ser caracterizados considerando dois modelos: o físico, onde os componentes são a rede de comunicação e os nodos, e o lógico no qual a aplicação é vista como distribuída, considerasse que a rede esta conectada e que os canais entregam mensagens na ordem que foram enviadas, apesar de não existir uma ordenação total e sim parcial das mensagens [10]. Eles devem garantir transparência, comunicação, desempenho, escalabilidade, heterogeneidade, *Openness*, confiabilidade, tolerância a falhas e segurança.

Os sistemas de computação distribuída são caracterizados pela sua estrutura, onde um grande número de dispositivos interage. Diferenciam-se de computadores paralelos, por serem de fraco acoplamento, pois os elementos desses sistemas não têm acesso a uma memória comum. Os mesmos apresentam uma redundância natural, o que facilita a utilização de técnicas de tolerância falhas [10] [13].

Do ponto de vista de tolerância a falhas, os sistemas distribuídos têm uma grande vantagem por serem redundantes. Entretanto, existem diversas dificuldades que permeiam questões de *dependability*, - que indica qualidade de serviço e agrega atributos como segurança (*security*), disponibilidade, confiabilidade, segurança de funcionamento (*safety*), testabilidade, mantenabilidade e desempenho (*performability*) - [10]. Geralmente os problemas em lidar com *dependability* em tais ambientes surgem a partir da distribuição geográfica e instabilidade dos recursos, as altas frequências de atualizações do usuário e as transferências de dados, restrições das aplicações, entre outros [14]. É de essencial importância solucionar problemas referentes de consenso, ordenação e atomicidade na troca de mensagens entre grupos de processos, sincronizar relógios quando necessário, implementar réplicas consistentes de objetos, garantir consistência de dados e processos em sistemas computacionais [10].

3.1 Modelos de Sistemas Distribuídos

Os sistemas distribuídos podem ser classificados em síncronos e assíncronos, a depender se existem ou não a previsibilidade temporal para a comunicação entre processos e execução de passos computacionais para troca de mensagens.

3.1.1 Síncronos

Os sistemas síncronos são determinados pela existência e conhecimento dos limites temporais. Neles, cada processo executa um passo para realizar a computação e todas as mensagens serão entregues. Estes possibilitam que existam uma estimativa de tempo para a execução dos protocolos do sistema e das funções da aplicação distribuída [19] [13]. Portanto, para que os sistemas sejam síncronos limites devem ser definidos: devem existir limites de tempo inferiores e superiores para executar cada processo; mensagens devem ser recebidas em um tempo limite conhecido; e cada processo tem um *clock* onde a taxa de desvio tem limite conhecido [20].

Estes sistemas são eficientes para tolerar falhas, pois a existência de *timeouts* assessora na detecção de falhas [19]. Os processos executam em lock-step, ou seja, avançam em conjunto, e não existe incerteza de sincronismo. Isto facilita a criação de protocolos, mas os torna mais suscetíveis a problemas de sincronização do mundo real [13].

3.1.2 Assíncronos

Diferente dos sistemas síncronos, nos assíncronos não existem hipóteses acerca de intervalos de tempo [20] [19]. Neste modelo, os processos não precisam satisfazer propriedades de *timeliness*. Não existem limites de taxas de execução, com isso um processo pode ser executado com um limite de taxa mais rápida que o outro. Também não tem limites sobre atrasos de mensagens, com isso elas podem ser enviadas de um processo para outro e serem realizados de forma rápida ou devagar [21].

Em sistemas assíncronos não é possível obter consenso na presença de falhas, o que dificulta definir se um processo esta lento ou se é falho [19]. Nestes, uma solicitação enviada não precisa esperar por resposta, diferente dos síncronos. Com isso, não é necessário aguardar a finalização da execução de um processo de iniciar outro.

3.2 Falhas em Sistemas Distribuídos

3.2.1 Falhas por Crash e Crash Recovery

Falhas por *crash* ocorrem quando um processo falha em prosseguir com sua computação, não entregando ou recebendo mensagens e nem executando eventos locais. Em um ambiente síncrono, ou seja com *timeouts* conhecidos, é possível perceber a falha por *crash* e os demais processos de uma

computação distribuída excluirem o processo falho. Para que um sistema tolere esse tipo de falha é necessário pelo menos *f+1* réplicas. Já as falhas por *Crash Recovery* são aquelas que mesmo que um serviço falhe, é possível que o processo retorne a um estado anterior ao da falha e a partir de então continue a computação - isto pode requerer, por exemplo, uma sincronização do estado local deste processo com os demais participantes da computação distribuída [13] [22].

3.2.2 *Falhas Bizantinas*

O modelo de falhas bizantinas foi inspirado no problema dos generais bizantinos. Onde n generais defendendo uma cidade, no qual t deles foram subordinados pelos turcos para atrapalhar a operação. Por estarem separados pelo relevo, somente se comunicam através de mensageiros e os mesmos só poderão vencer se todos atacarem. A questão que norteia esse problema é, qual a quantidade necessária de generais para suportar o mínimo de traidores. O número de *n=f+1* processos utilizado para falhas por crash não se aplica a este contexto, não se aplica a este contexto [15] [13].

O problema é insolúvel para qualquer arranjo de $n \leq 3f$ (n é o número de réplicas e f é número de instâncias que podem falhar), onde n=3 e f=1, como ilustrado na Figura 1 , que apresenta três generais, onde o C é um traidor. A diz para B e C atacar, entretanto C diz para B recuar, que não consegue saber a qual dos comandos atender. Com isso, eles nunca chegarão a um consenso. Diante disto, *Lamport, Shostack e Peace* provaram que o problema tem solução se, e somente se, $n \geq 3f + 1$, ou seja, para um sistema suportar o mínimo de falhas *(f=1)*, serão necessárias no mínimo 4 máquinas [16] [15] [13]. A Figura 1 também mostra a solução proposta por Lamport, Shostack e Peace, onde os generais conseguem entrar em um consenso.

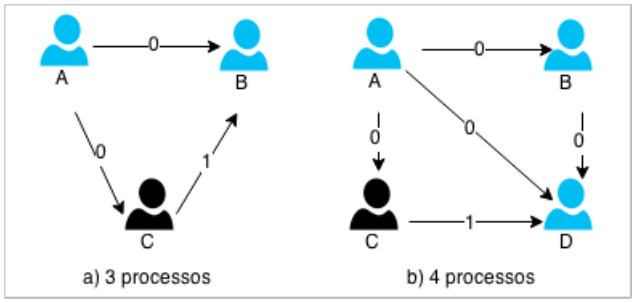

Figure 1: Problema dos generais bizantinos: 3 processos vs. 4 processos

Duas condições são necessários para identificar falhas bizantinas: a primeira relata que os processos corretos devem possuir uma visão coerente das mensagens enviadas por cada processo. Técnicas como redundância da informação e assinaturas digitais não adulteráveis podem atender a este requisito. O segundo discorre que os processos corretos, devem poder verificar se as mensagens enviadas pelos demais, estão consistentes com os requisitos do algoritmo sendo executado para detecção de falhas bizantinas. Esta exigência pode ser atendida incluindo informações às mensagens, com certificados, para autenticar o conteúdo sendo transmitido [18] [17].

Segundo [17] e [18], as falhas bizantinas podem ser diferenciadas em detectáveis e não-detectáveis. As primeiras, se referem aos desvios que são revelados no comportamento externo de um processo, constatados por mensagens enviadas, ou não, por processos em execução, a partir de um algoritmo. Estas, podem ser classificadas em:

- Falhas por omissão: onde o processo falho não envia as mensagens requeridas a ele, ou envia apenas para uma parte dos processos [17];
- Falhas de segurança: que violam propriedades invariantes dos processos [17].

O segundo caso, são as falhas que não são notadas nas mensagens, e se forem identificadas não podem ser atribuídas a um processo [18] [17]. Estas podem ser classificadas em:

- Não-observáveis: quando os processos que não falharam, não percebem a ocorrência da falha [17];
- Não-diagnosticáveis: quando não é possível saber qual o processo que gerou a falha [17].

4. OPENSTACK

O *OpenStack* é um projeto *open source* desenvolvido, inicialmente, pela NASA *(National Aeronautics and Space Administration)* e pela *Rackspace*. É um sistema operacional na nuvem que tem fornecido confiabilidade, robustez e disponibilidade com uma solução IaaS (*InInfrastructure as a Service*) tanto para infraestruturas públicas, quanto privadas. Possui uma arquitetura modular e configurável que possibilita as empresas escolherem uma diversidade de serviços que atendam as necessidades de rede, computação e armazenamento. Uma solução que contempla recursos de hardware adequados, disponíveis tanto para um nível profissional, quanto básico, com poucos ou muitos nós de computadores [5] [6].

O objetivo principal do projeto *OpenStack* é desenvolver um sistema operacional em nuvens altamente escalável. Com elasticidade e escalabilidade horizontal, todos os serviços e componentes seguem uma política *shared-nothing*, que faz com que qualquer um de seus serviços ou componentes possam ser instalados em qualquer local. O *OpenStack* é composto por sete serviços principais, como é ilustrado na Figura 2, que mostra a arquitetura básica do sistema [9].

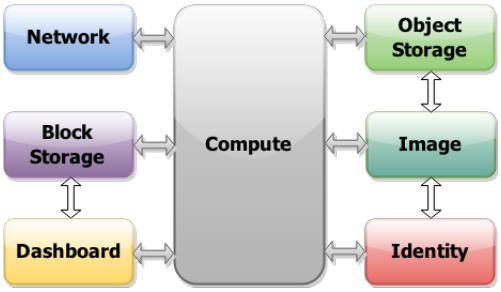

Figure 2: Arquitetura do OpenStack

O sistema da *OpenStack* controla uma larga quantidade de *pools* de armazenamento, computação e recursos de rede em um *datacenter*. Através de um painel, fornece um conjunto de ferramentas aos administradores para gerenciar, controlar e oferecer aos seus usuários uma prestação de serviços através de uma interface web [5]. *OpenStack* gerencia um grupo de recursos virtuais interdependentes, como máquinas virtuais, redes virtuais e volume de dados, e define os vários níveis de granularidade dos recursos virtuais. Fornece vários mecanismos para interação com o mesmo, com APIs (*Application Programming Interface*) em REST e *Python* que fornecem serviços de notificação [7].

A *OpenStack* dispõe de diversos componentes, dentre os principais estão: *Nova* (computação), *Glance* (Repositório VM), *Neutron* (networking), *Swift* e *Cinder* (armazenamento), que serão descritos a seguir. Além destes, em um projeto separado, é disponibilizado uma implementação do *Open Cloud Computing Interface*(OCCI) [6] [7].

- *Nova*: Fornece servidores virtuais sob demanda que interagem com vários tipos de *hypervisors*, como KVM, *Xen, VMware* ou *Hyper-V*. Essencial para uma implementação básica de arquitetura em nuvem, o mesmo fornece serviços para a gestão de recursos na *cloud* por meio de APIs, capazes de organizar instâncias em execução, redes e controle de acesso. Os recursos e funções mais importantes da Nova são: Gestão do ciclo de vida das instâncias; Gestão dos recursos de computação; Redes e Autorização; API baseada em REST; comunicação assíncrona consistente; suporte para *Xen, XenServer / XCP*. Os principais componentes que compõem a mesma são: API *Server (nova-api); Message Queue (rabbit-mq server) Compute Workers (nova-compute); Network Controller (nova-network); Volume Worker (nova-volume); Scheduler (nova-scheduler)* [6] [8].
- *Glance*: Sistema de recuperação e pesquisa de imagens de maquinas virtuais (VM). O *Glance* oferece serviços para recuperar, descobrir e registrar imagens virtuais através de uma API que propicia a consulta de *metadados* de imagens VM, catálogo e gerenciamento de grandes bibliotecas de imagens de servidores [6]. A função principal do *Glance* é fornecer um serviço de imagem, ele pode ser configurado para utilizar as seguintes infraestruturas de armazenamento: *OpenStack Object Store* para armazenar imagens, *S3 storage directly* e *S3 storage* com *Object Store* como intermediário para acessar S3 [8].
- *Neutron*: Serviço de rede da *OpenStack*, que fornece recursos para o gerenciamento de protocolo de configuração dinâmica de host (DHCP), protocolo de *Internet* estático (IP's), ou redes virtuais (VLANs), juntamente com outras políticas avançadas e topologias. Devido a sua arquitetura, permite aos usuários aprimorar *frameworks* para suportar os fornecedores [6].
- *Swift*: Serviço de armazenamento distribuído, com alta disponibilidade. Permite aos usuários armazenar e recuperar arquivos [6]. Semelhante ao serviço *Amazon Web Services - Simple Storage Service*, pode armazenar bilhões de objetos distribuídos entre os nós. É extremamente escalável tanto em termos de tamanho, quanto de capacidade. As principais características e funções do *Swift* são: armazenamento de um grande número de objetos; redundância de dados; trabalha com grandes conjuntos de dados; capacidades de streaming de mídia; armazenamento seguro de objetos; *backup* e arquivamento; escalabilidade e armazenamento de grandes objetos [8].
- *Cinder*: Serviço que fornece armazenamento persistente em blocos para *guest* de máquinas virtuais. Trabalhando com o *Swift* pode fazer *backup* de grandes volumes de VMs. Cinder interage principalmente com Nova, fornecendo volumes de suas instâncias, permitindo através de uma API a manipulação de volumes, tipos de volume e *snapshots* de volume [5] [6].

4.1 DevStack

DevStack é um *shell script* usado para implantar um ambiente completo de desenvolvimento *OpenStack*. É um repositório de código que instala e inicia o *OpenStack development* e um ambiente de testes. O mesmo suporta um grande número de opções de configurações, plataformas alternativas e serviços de apoio. Originalmente foi desenvolvido pela *Rackspace Cloud Builders* e atualmente mantido pela comunidade da *OpenStack* [11] [5].

Os componentes suportados pelo DevStack são: Sistemas Operacionais (*Ubuntu, Fedora, RHEL*); Base de dados (*MySQL e PostgreSQL*); *Queues (Rabbit e Qpid); Web Server (Apache); OpenStack Network (Nova Network e Neutron)*; Serviços (*Identity (Keystone), Object Storage (Swift), Image Storage (Glance), Block Storage (Cinder), Compute (Nova), Network (Nova), Dashboard (Horizon) e Orchestration (Heat)*); Configurações de nós e Exercícios [5]. Dentre uma variedade de scripts que são disponibilizados pelo *DevStack*, os mais utilizados são os descritos abaixo, pois realizam a instalação, a reinicialização, entre outras ações para o sistema [11] [5].

- *stack.sh*: *Script* utilizado inicialmente para instalar o *OpenStack*, permite especificar as configurações do repositório *git*, as imagens ISO que serão utilizadas, as configurações de rede e senhas, entre outros. O mesmo instala e configura vários combinações do *Ceilometer, Cinder, Glance, Heat,Horizon, Keystone, Nova, Neutron e Swift*. É necessário executá-lo toda vez que uma alteração na configuração do *OpenStack* é realizada ou um novo serviço habilitado [11] [5].
- *unstack.sh*: Interrompe todos os serviços inicializados pelo *stack.sh*, exceto *mysql* e *rabbit* que são deixados em execução [11] [5].
- *rejoin-stack.sh*: *Script* que recria a sessão de uma execução anterior. Possibilitando que não haja perda de dados caso a maquina seja reiniciada [11].
- *run_test.sh*: Script que executa uma ferramenta chamada *bash8* que fornece informações sobre todos os espaços em brancos dispersos [11].
- *clean.sh*: *Script* utilizado para apagar todos os arquivos relacionados ao *OpenStack* do sistema [11] [5].

5. TRABALHOS CORRELATOS

Nesta seção, serão apresentados os trabalhos relacionados. Muitos sistemas desenvolvidos com o objetivo de gerenciar réplicas e recursos em ambientes de nuvens podem ser encontrados na literatura.

O trabalho de [23] propõe em um de seus módulos um gerenciador de réplicas em ambientes de nuvem *eucalyptus*. O *Just-in-Time* (JiT) *Cloud* é uma agregação de diversos *JiT Data Centres* que estão dispersos geograficamente, provendo infraestrutura computacional como serviço, por meio da alocação sob demanda de máquinas virtuais [23]. O mesmo é composto por três módulos, dentre estes o *JiT DC Middleware* é o responsável por gerenciar os recursos das máquinas virtuais instanciadas por meio abstração para a *cloud* privada implantada pelo *eucalyptus*. Outro módulo, denominado *MDA* (Módulo de Dependabilidade Autonômica), oferece um serviço de replicação *primary-backup* para máquinas virtuais com mecanismo de gestão autonômica.

A artigo [24] discorre sobre um sistema de criação e gestão de tolerância a falhas em ambientes de cloud computing. Ele apresenta o FTM (*Fault Tolerance Manager*), um *framework*, onde em um de seus componentes, - o *Replication Manager* -, fornece um serviço de tolerância a falhas. Para isto, ele utiliza uma técnica de replicar as aplicações dos usuários, por meio de um mecanismo de replicação *primary-backup*, garantindo que se uma falhar ocorrer, haverá uma copia dos dados. No ambiente de computação em nuvem, a redundância também pode ser aplicada em toda a instância VM em que o aplicativo está hospedado [24].

O trabalho [25] apresenta *Claudia*, um sistema de gerenciamento de uma nuvem federada que implementa uma camada de abstração que pode ficar em cima de diferentes provedores de nuvem. Este sistema provê uma camada de abstração única, que tenta amenizar problemas de *lock-in* e permitir a federação transparente de nuvens para a execução dos serviços, preocupando-se com o ciclo de vida das aplicações. Permitindo que réplicas estejam hospedadas nas nuvens de diferentes provedores, como, por exemplo, em uma nuvem baseada em *OpenStack* e outra baseada em *Eucalyptus*.

6. CLOUDGI - GERENCIANDO INSTÂNCIAS DE UM SERVIÇO REPLICADO EM UMA PLATAFORMA DE COMPUTAÇÃO EM NUVEM OPENSTACK

Esta seção do artigo discorre sobre o funcionamento do Gerenciador de Instâncias de réplicas em Ambientes de Nuvens com Serviço de Tolerância a Falhas, o CloudGI. Um sistema que tem como principal objetivo, criar e monitorar instâncias de réplicas em uma *cloud*, com foco no tipo de serviço e tipo de falha, a ser tolerada, definidos pelo cliente.

A proposta do CloudGI difere das apresentadas, na seção de trabalhos correlatos, por fornecer um suporte ao serviço de replicação ativa. A aplicação permite instanciar máquinas virtuais em uma plataforma de computação em nuvem IaaS correspondentes às réplicas de um serviço replicado.

O CloudGI monitora e gerencia as réplicas, atuando na plataforma de computação em nuvem em caso de falhas, podendo reiniciar instâncias, ou em caso de um estado mais severo, recriá-las. Este suporte em um ambiente de replicação ativa permite suportar o modelo de falhas *crash-recovery*, em que réplicas falhas, ao reiniciarem, podem sincronizar o estado com as demais e prosseguem na computação.

Se o *middleware* ou servidor de aplicação utilizado provê suporte a replicação de falhas bizantinas, implementando um protocolo como o PBFT [30]. Ou seja, se, dentre outras coisas: as réplicas utilizam de chaves criptográficas apropriadas para a comunicação entre si, e o progresso da computação depende da concordância de mais de dois terços das réplicas, CloudGI implementa o mecanismo de rejuvenescimento de réplicas, no qual, periodicamente cada réplica é recriada, assumindo que ao longo do tempo uma réplica pode ser comprometida e esta operação restabelece o código original da réplica. Neste caso, mecanismos adicionais de salvaguarda, como a manutenção das imagens, que instanciam as máquinas virtuais, em dispositivo protegido de escrita, podem ser implementados.

O CloudGI fornece um serviço de criação e monitoramento de réplicas. O mesmo auxilia no suporte a replicação tolerante a falhas por *crash* e falhas bizantinas. De acordo com a falha selecionada, o mesmo fornece três tipos de serviço, que foram classificados em: Ouro, Prata e Bronze. No qual, para cada tipo de falha, existem estes serviços relacionados. A Tabela 1 apresenta o número de réplicas e falhas para cada serviço de acordo com o tipo de falha, que pode ser bizantina e por crash.

Tipo de Serviço	Nº de réplicas Falhas	réplicas por tipo de falha	
		Crash	Bizantina
Bronze	1	2	4
Prata	2	3	7
Ouro	3	4	10

Tabela 1: réplicas e falhas por serviço

Em linhas gerais, no Modo *Crash-Recovery*, temos que $n = f + 1$; e no Modo Bizantino, $n = 3f + 1$. Assumimos como premissa que a infra-estrutura de computação em nuvem é configurada sob um ambiente dedicado e com uma reserva adequada de recursos (e.g. rede de computadores, servidores, sistemas operacionais etc.), no qual é possível determinar com alto grau de probabilidade as latências máximas de comunicação e processamento, ou seja, o conjunto de réplicas, os componentes de software do CloudGI, os sistemas operacionais hospedeiros e a plataforma de computação em nuvem utilizada atuarão como um sistema síncrono. Esta premissa pode ser válida apenas para o conjunto de réplicas, uma vez que os clientes do serviço replicado podem estar dispostos sob a *Internet*. Contudo, tal premissa permite assumir a detecção perfeita de falhas por *crash*.

CloudGI foi desenvolvido para atuar sob a plataforma de computação em nuvem IaaS *OpenStack* [35]. Outras plataformas IaaS de código-aberto que poderiam ter sido utilizadas incluem *OpenNebula* [31], *Eucalyptus* [32] e *Nimbus* [33]. O conjunto de funcionalidades do *OpenStack*, como a compatibilidade com Amazon EC2 e Amazon S3, permitindo a oferta de mesmos serviços da Amazon, mas em uma nuvem privada ou comunitária, bem como uma tendência crescente

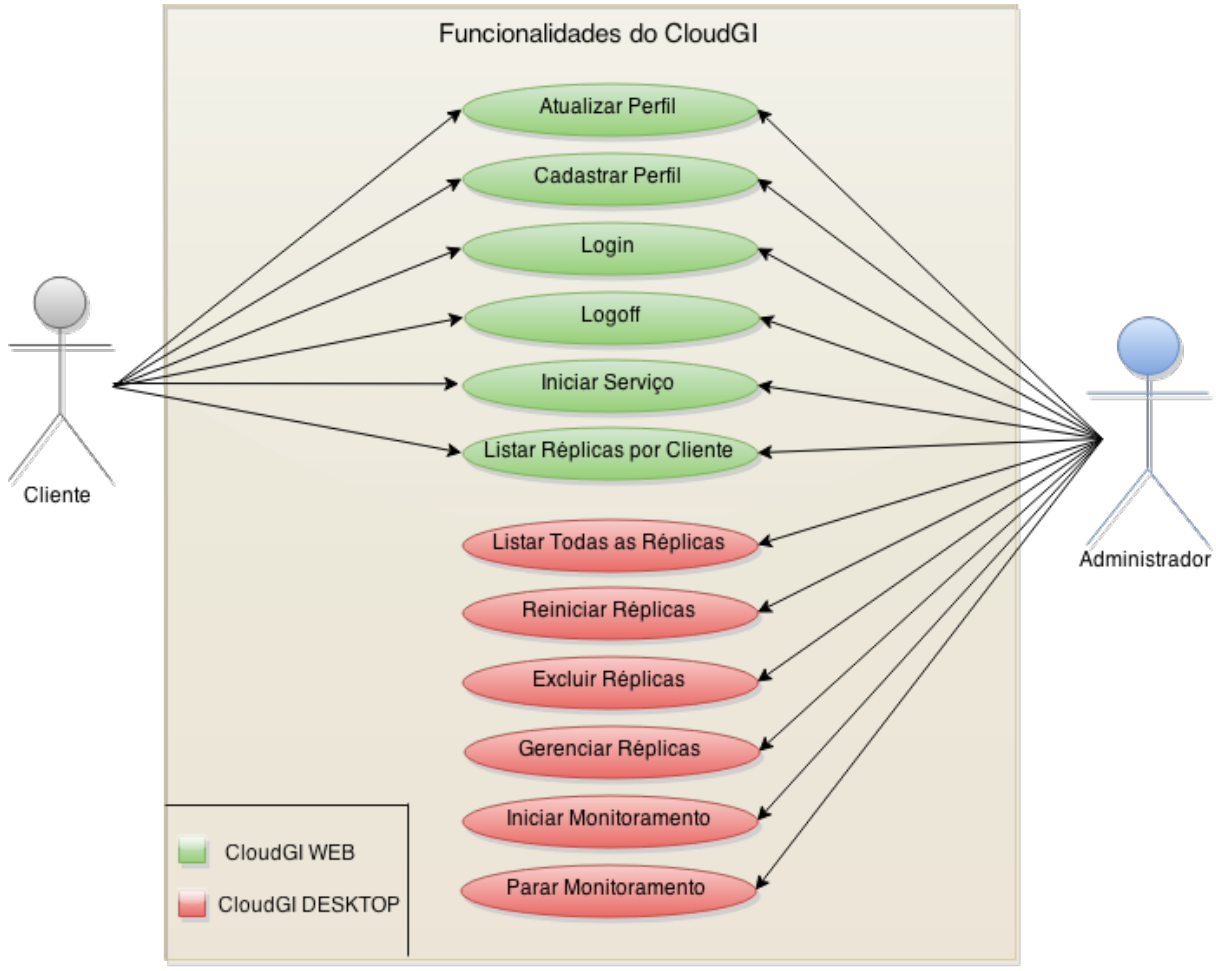

Figure 3: Diagrama de caso de uso para os principais atores do sistema: Cliente e Administrador

de usuários em comparativo a outras plataformas de nuvem IaaS [39, 40] embasou a escolha do *OpenStack*.

O CloudGI foi desenvolvido em dois módulos. Um voltado para o cliente, uma aplicação *web*. Um com foco no gerenciamento e monitoramento das instâncias, um aplicativo para *desktop*.

6.1 Casos de Uso

Esta seção, apresenta os casos de uso do sistema. Discorre os usuários do sistema e quais as funcionalidades que cada perfil tem acesso.

A Figura 3 apresenta o caso de uso para os principais atores do sistema e as funcionalidades que cada um tem acesso. A aplicação será utilizada por três perfis de usuário:

- Utilizador Geral: Toda e qualquer pessoa, que queira visitar a página do sistema para conhecer o projeto, podendo a mesma realizar o cadastro no site, e vir a se tornar um usuário do tipo cliente.
- Utilizador Administrador: Quem administra o sistema, tem acesso a todas as funções do mesmo. O Administrador pode criar instâncias, iniciar e parar o monitoramento das mesmas, excluir e visualizar todas as réplicas presentes na nuvem, cadastra usuários, atualizar seu perfil.
- Utilizador Cliente: Usuário para qual o sistema é destinado, quem contrata o serviço de gerenciamento e tem acesso a funções importantes do sistema. O cliente pode iniciar um serviço, que possibilita a criação de instâncias na nuvem, obter uma lista de todas as suas réplicas e atualizar o seu perfil. Para isto, o mesmo necessita está autenticado no sistema.

6.2 Arquitetura

Uma visão geral do sistema proposto neste artigos, está ilustrada na Figura 4. A mesma apresenta a *cloud* que é gerenciada pelos serviços da *OpenStack*, que por sua vez, é executada em uma Servidor *Ubuntu*. Um cliente acessa o CloudGI, online, e solicita os serviços por ele disponibilizados. O gerenciador monitora o estado das maquinas, e realiza as ações necessárias para o correto funcionamento, através do acesso ao servidor *Ubuntu* com comandos de gerenciamento do *OpenStack*.

O padrão arquitetural utilizado no CloudGI foi o MVC (*Mo-*

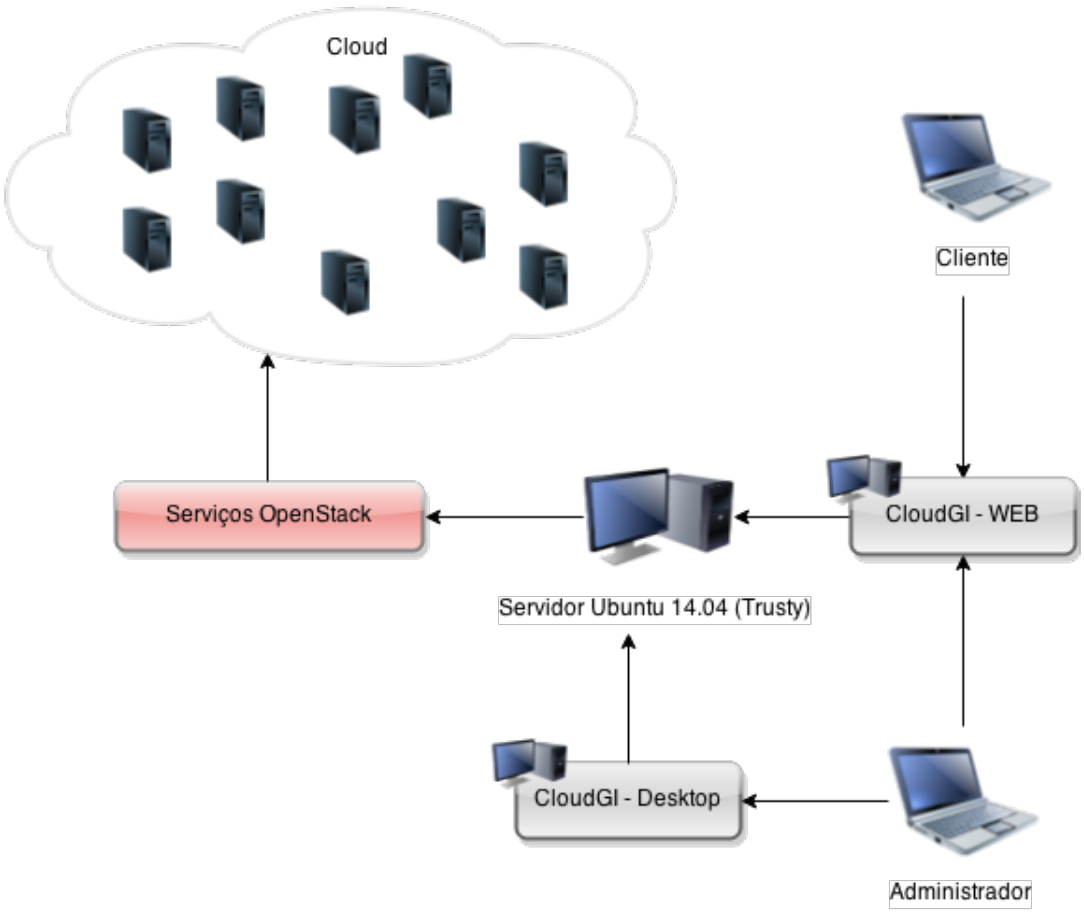

Figure 4: Visão Geral do CloudGI

del - View - Controller)/Modelo - Visão - Controle. Este padrão promove a separação em camadas de diferentes níveis de abstração. Isso garante o fácil acoplamento e manutenção dos aplicativos [29].

O *Model* é a camada lógica da aplicação, responsável pela validação lógica, pelo processamento, integração e cálculos. O *View* é a camada que tem por objetivo realizar a interação com o usuário, geralmente são elementos de interface. O *Controller* é responsável por processar e responder os eventos que recebe de requisições, principalmente de usuários, e interagir com a camada de modelo [29].

O CloudGI utilizou o padrão de Projeto DAO (*Data Access Object*). O mesmo tem por objetivo abstrair o mecanismo de persistência utilizado numa aplicação, permitindo a organização lógica de acesso aos dados e encapsulamento de características especificas [38]. A Figura 5 apresenta o diagrama de classes, a interação entre o *Controller* e o DAO.

6.2.1 Requisitos Funcionais

São definidos como requisitos funcionais para atender as necessidades propostos pelo sistema CloudGI:

1. Definir um serviço a ser replicado e o acordo de prestabilidade do mesmo: O sistema deve permitir ao cliente escolher o tipo de serviço (ouro, prata, bronze) de acordo com suas demandas;
2. *Login/Logout* no sistema: O sistema deve permitir que o usuário se autentique no sistema por meio de um *login* e uma senha de acesso. Também deve permitir que o usuário se desconecte do sistema;
3. Cadastrar perfil: O sistema deve fornecer ao usuário a opção de cadastrar-se no sistema;
4. Criar instâncias de réplicas: O sistema deve ser capaz de iniciar as réplicas solicitadas pelo cliente de acordo com o serviço escolhido;
5. Listar todas as instâncias por cliente: A aplicação deve capaz de listar todas as réplicas que pertencem ao cliente;
6. Listar todas as instâncias: A aplicação deve ser capaz de listar todas as réplicas que se encontram na nuvem da *OpenStack* para o administrador;
7. Deletar instâncias de réplicas: O sistema deve fornecer a opção de exclusão de instância da nuvem;
8. Reiniciar instâncias: O sistema deve ser capaz de reiniciar instâncias, caso necessário;

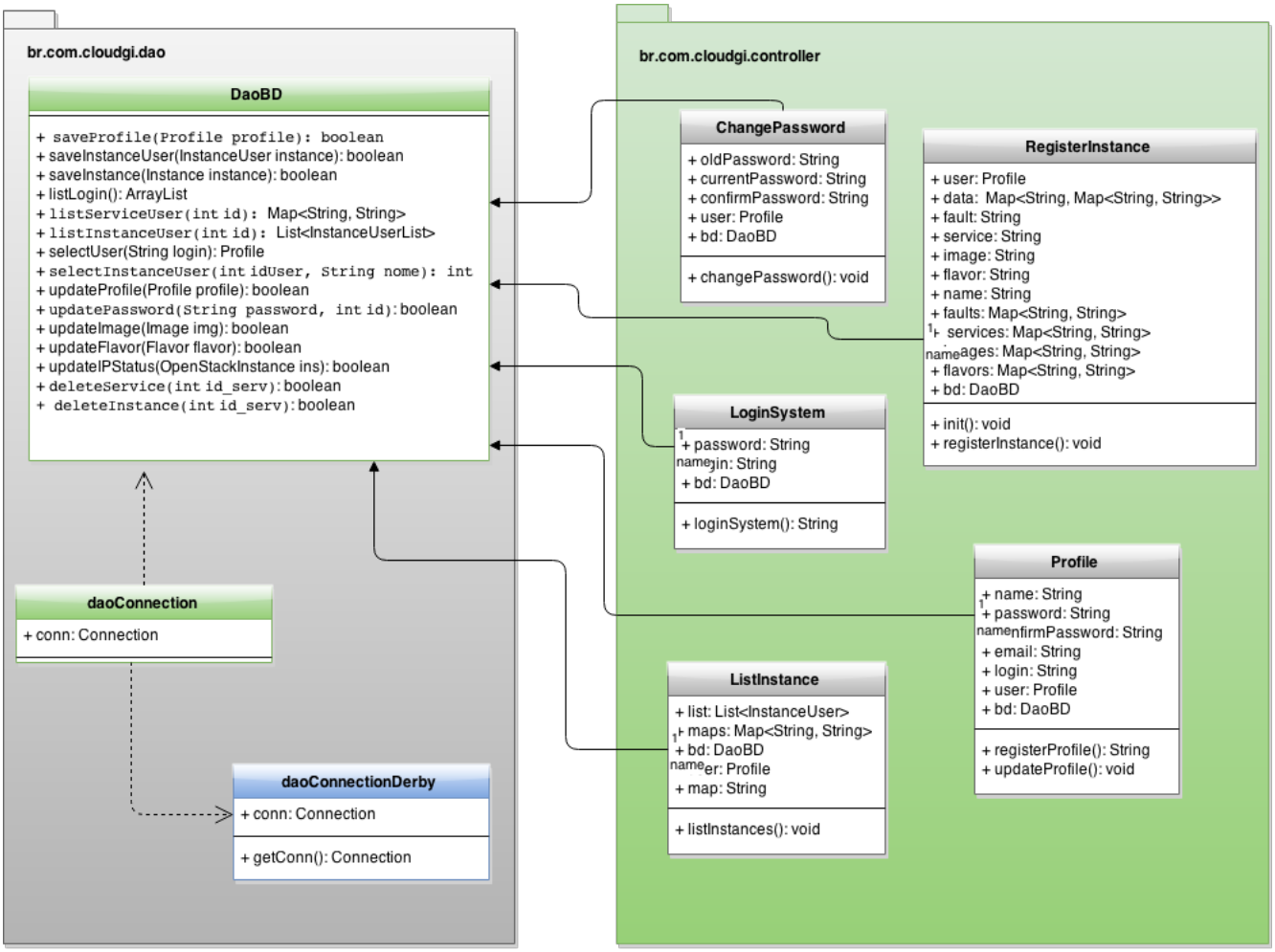

Figure 5: Diagrama de Classes

9. Monitorar as instâncias: O sistema deve ser capaz de monitorar todas as réplicas para atender aos requisitos propostos pela aplicação;

10. Persistir os dados dos clientes no banco: A aplicação deve ser capaz de gravar os dados na base de dados;

11. Atualizar perfil: O sistema deve fornecer ao usuário alterar informações em seu perfil.

6.2.2 Requisitos Não-Funcionais

A Tabela 2 apresenta os requisitos não-funcionais do sistema. Os mesmos abordam características como: segurança, usabilidade, manutenabilidade, entre outros.

Requisito Não Funcional	Categoria
O sistema deverá prover uma interface de fácil manuseio para o usuário	Usabilidade
Usuários não devem ter acesso a réplicas e informações de outros clientes	Segurança
Réplicas devem ser monitoradas periodicamente para garantir que as mesmas estejam ativas	Integridade
O sistema será desenvolvido na linguagem de programação orientada a objetos Java e o banco de dados JDBC.	Software

Tabela 2: Requisitos não-funcionais

6.3 Base de Dados

Esta seção apresenta o modelo do banco de dados do CloudGI. A Figura 6 ilustra a modelagem física da base de dados.

6.4 Tecnologias Utilizadas

Para atender aos requisitos funcionais do sistema o CloudGI desenvolvido na plataforma Java EE 7 - (*Java Plataform*

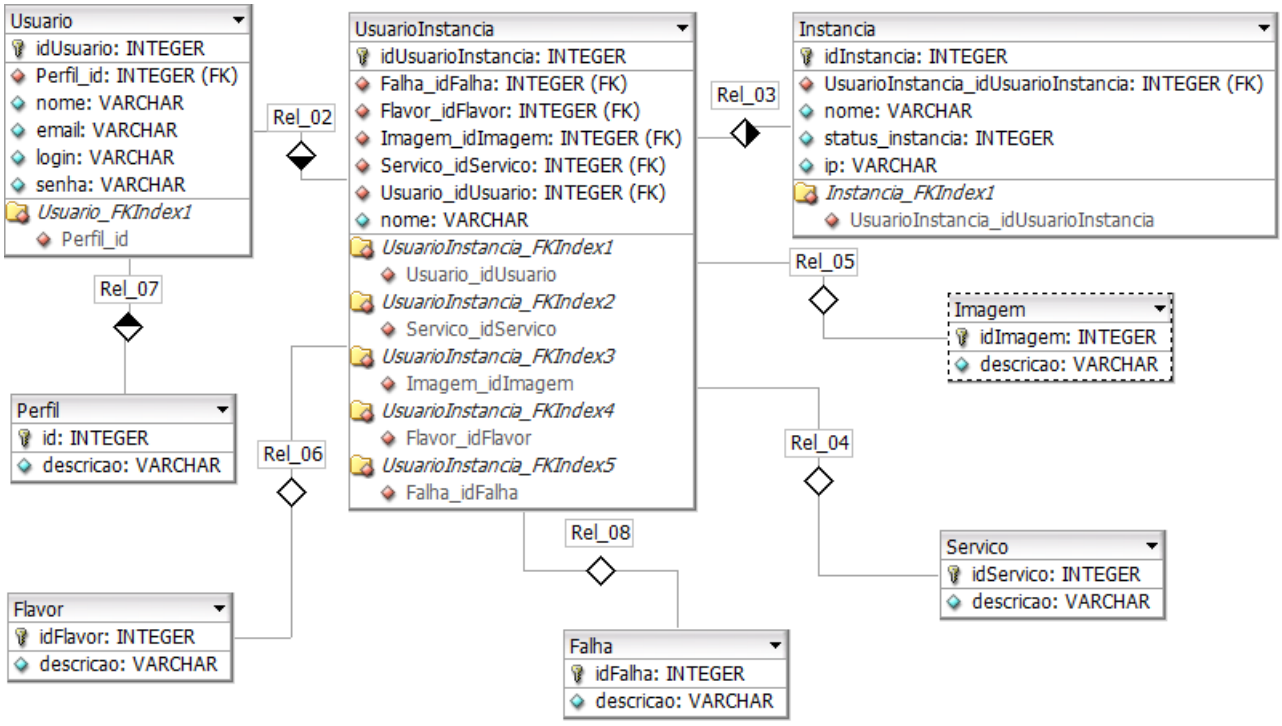

Figure 6: Modelo físico do banco de dados do CloudGI

Enterprise Edition 7. O Java EE é uma plataforma de desenvolvimento da *Oracle* que possui um conjunto de recursos voltados para desenvolvimento, gerenciamento e implantação de aplicações. A linguagem de programação adotada é o *Java*. Oferece muitas ferramentas para o desenvolvimento de APIs e incentiva o uso de padrões para construção das aplicações[26].

O servidor de aplicação utilizado foi o *Glassfish*. Um servidor *open source* que suporta especificações da API Java EE como JDBC, RMI, JavaMail, JMS, JMX [28].

Para armazenamento e consulta dos dados foi utilizado à tecnologia *Java Database Connectivity* - JDBC. Uma API Java, distribuição da *Oracle*, do banco de dados *open source Apache Derby*. Por ser de fácil uso, tem vários recursos completos para gerenciamento de dados, podendo acessar qualquer tipo de dados tabulares, como os armazenados em um SGBD. Por ser desenvolvido em Java, o mesmo é uma poderosa ferramenta para desenvolvimento de aplicações na linguagem Java [26].

O ambiente de desenvolvimento integrado utilizado foi o *Netbeans*. Framework genérico que possibilita o desenvolvimento de aplicações robustas e extensíveis. Suporta diversas linguagens como *Java, C, C++, PHP, Groovy, Ruby*, entre outras [27].

Para o gerenciamento do ambiente de réplicas nas nuvens foi utilizado o sistema e a nuvem da *OpenStack*, sendo necessário utilizar, para instalação deste serviço, o sistema operacional *Ubuntu 14.04 (Trusty)* em uma máquina. Diante disto, são necessárias algumas especificações para utilização deste serviço. Para o gerenciamento das réplicas foi preciso utilizar os comandos do serviço *Nova*, expostos na Tabela 3. Estes comandos possibilitaram a criação, exclusão e listagem de instâncias, imagens, *flavors* e par de chaves.

6.5 Funcionalidades

Dois módulos foram desenvolvidos para o CloudGI: A aplicação *web* e a *desktop*. A primeira foi desenvolvida com foco no cliente, possibilitando ao mesmo o gerenciamento das suas réplicas. Já o segundo foi desenvolvido com o objetivo de monitorar todas as instâncias.

As funcionalidades do módulo *web* são:

- **Cadastrar usuário:** Esta opção está disponível para qualquer usuário que acessar o módulo web. O formulário de inscrição é ilustrado na Figura 7.
- **Iniciar serviço:** Para iniciar um serviço o usuário necessita está cadastrado no sistema e possuir um *login* e senha de acesso. Com isso, ele poderá criar instâncias determinado o tipo de serviço e falha. Como ilustrado na Figura 20, para um cliente criar instâncias será necessário o mesmo o nome do serviço a ser criado, o tipo de falha (*Crash* ou Bizantina), a quantidade de réplicas a serem instanciadas, a imagem da máquina (*Cirros ou Fedora*) e o *flavor*, que por enquanto se limitam apenas aos *m1.nano e m1.micro*.
- **Listar Instâncias do usuário:** Está função permite ao usuário visualizar todas as instâncias por ele já criadas, incluindo informações como o tipo de falha que

Comando	Descrição
$ nova list	Lista todas as instâncias que estão na nuvem
$ nova image-list	Lista todas as imagens de sistemas operacionais que estão no servidor da *OpenStack*
$ nova flavor-list	Lista todos os *flavors* que estão no servidor da *OpenStack*
$ nova boot –flavor FLAVOR_ID –image IMAGE_ID –key-name KEY_NAME –security-groups SEC_GROUP_NAME INSTANCE_NAME	Inicia uma instance
$ nova delete INSTANCE_NAME	Exclui uma instância
$ nova reboot INSTANCE_NAME	Reinicia uma instância
$ nova pause/unpause INSTANCE_NAME	Pausa/despausa uma instância
$ nova shelve/unshelve INSTANCE_NAME	Arquiva/desarquiva uma instância
$ nova suspend/resume INSTANCE_NAME	Suspende/resume uma instância
$ nova stop/start INSTANCE_NAME	desliga/liga uma instância
$ nova keypair-add KEY_NAME > MY_KEY.pem	Cria um par de chaves

Tabela 3: Requisitos não-funcionais

tolera cada réplica, a imagem, a qual serviço a mesma pertence, o IP da máquina na nuvem da *OpenStack* e o *status*. Ver Figura 9.

- **Excluir serviço:** Esta função possibilita que um usuário finalize um serviço. Com isso, são excluídas todas as instâncias do serviço em questão. A Figura 8 ilustra esta opção.

- **Gerenciar perfil do usuário:** Está função permite o usuário modificar informações do seu perfil, como senha, nome, entre outros.

A Figura 10 ilustra a tela de *login* do CloudGI web.

As principais funcionalidades do módulo *desktop* são:

- **Listar todas as instâncias presentes no servidor:** Esta função lista todas as instâncias presentes na nuvem da *OpenStack*. A Figura 11 ilustra a simulação da tela que apresenta a lista de instâncias.

- **Iniciar o serviço de monitoramento das réplicas**: Esta é uma das principais funções deste módulo, ela garante que as réplicas estejam em estado ativo, monitorando uma por uma para garantir o perfeito funcionamento das réplicas. Caso uma destas instâncias esteja em um *status* diferente de ativo, esta funcionalidade tomará as devidas providências para reaver a máquina.

- **Rejuvenescimento de instâncias:** Reinicia, periodicamente, replicas de serviços voltados à tolerância de falhas bizantinas.

- **Parar o monitoramento dos réplicas**: Função que permite parar o monitoramento das instâncias.

- **Excluir instâncias manualmente:** Esta função está presente no caso de haver a necessidade de excluir uma instância manualmente.

Figure 7: CloudGI: Tela de cadastro de usuário da aplicação web

Figure 8: CloudGI: Tela de exclusão de serviço da aplicação web

- **Reiniciar instâncias manualmente:**Esta função está presente no caso de haver a necessidade de reiniciar

Gerenciar

Lista de Instâncias

Lista de Instâncias						
Nome	IP	Status	Falha	Serviço	Imagem	Tamanho
user2_1_82_1	10.0.0.9	ACTIVE	Crash	Bronze	CIRRUS	m1.nano
user2_1_82_2	10.0.0.10	ACTIVE	Crash	Bronze	CIRRUS	m1.nano
user2_2_1_84_1	10.0.0.2	ACTIVE	Bizantina	Bronze	CIRRUS	m1.nano
user2_2_1_84_2	10.0.0.3	ACTIVE	Bizantina	Bronze	CIRRUS	m1.nano
user2_2_1_84_3	10.0.0.4	ACTIVE	Bizantina	Bronze	CIRRUS	m1.nano
user2_2_1_84_4	10.0.0.11	ACTIVE	Bizantina	Bronze	CIRRUS	m1.nano

Excluir Serviço

Figure 9: CloudGI: Tela de listagem de instâncias da aplicação web

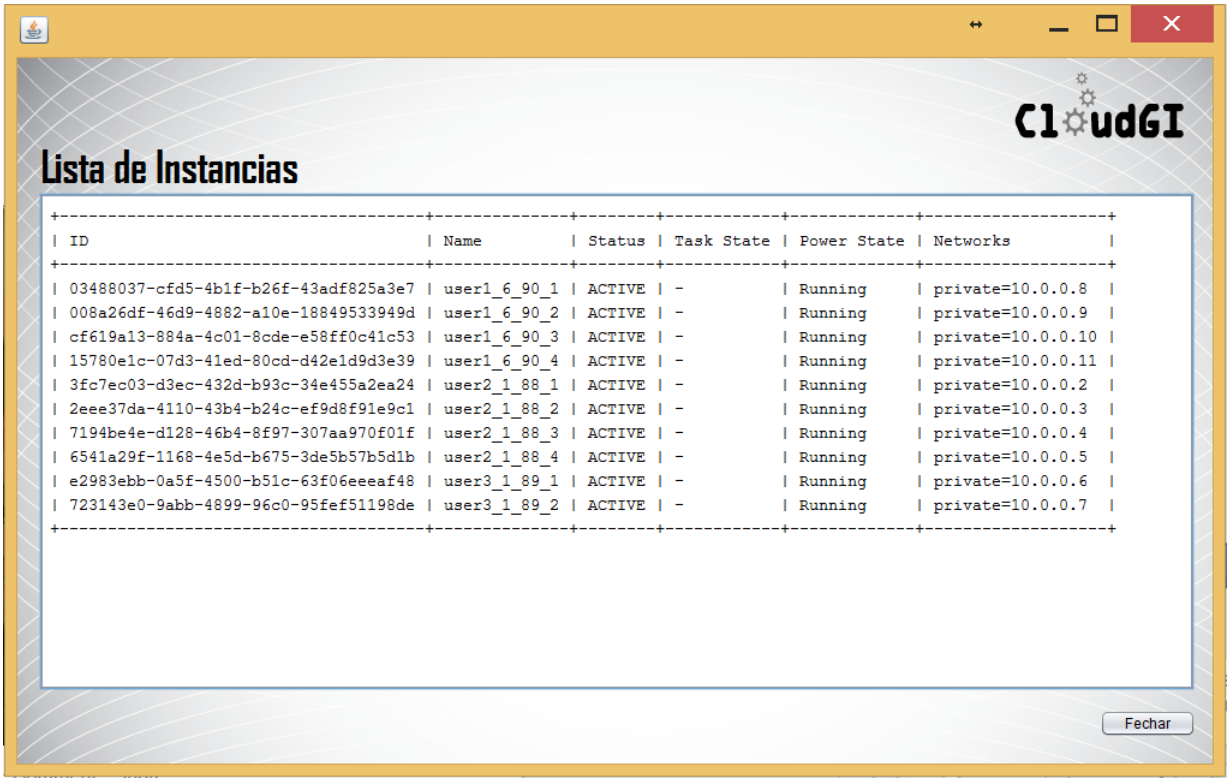

Figure 11: CloudGI Desktop: Tela de Listagem de Réplicas

uma instância manualmente.

- **Iniciar instâncias manualmente:**Esta função está presente no caso de haver a necessidade de criar uma instância manualmente. Ver Figura 12.

A Figura 13 ilustra a tela inicial do CloudGI Desktop.

6.6 Implementação do CloudGI

O módulo web do CloudGI oferece funcionalidades importantes para a manipulação dos serviços por parte do usuário. Para que um cliente tenha acesso a todas as funções é neces-

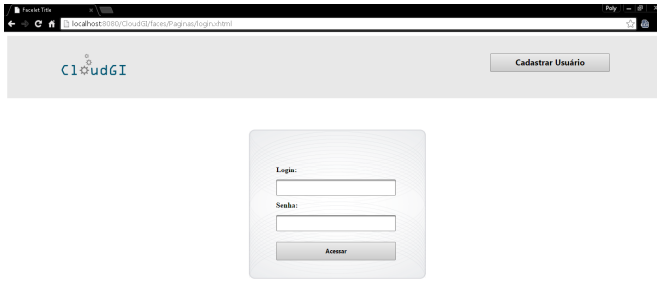

Figure 10: CloudGI: Tela de login da aplicação web

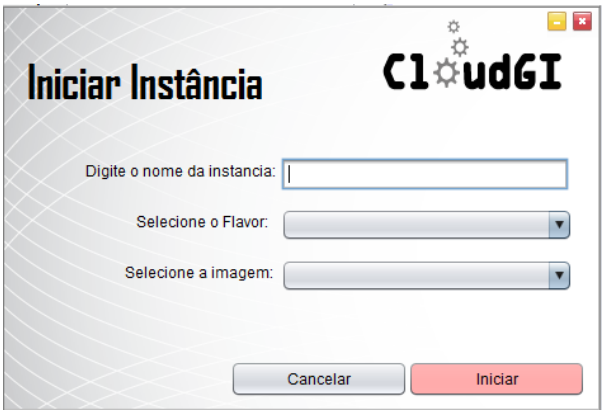

Figure 12: CloudGI Desktop: Criar instâncias

Figure 13: CloudGI Desktop: Tela Inicial

sário o mesmo realizar um cadastro, passando como principais informações o usuário e a senha. Com isso, o serviço de criação e monitoramento de instancias pode ser iniciado.

Para iniciar um serviço, o usuário precisa fornecer o nome do grupo de instâncias, o tipo de falha (Crash ou bizantina) que deseja tolerar, o serviço (ouro, prata ou bronze) que define o número de instâncias a ser criadas, a imagem (fedora ou cirros) e por fim o *flavor (nano ou micro)*. Ver Figura 20. Após fornecer estas informações, que serão armazenadas

Nome	Memória	Disco
m1.nano	64	0
m1.micro	128	0
m1.heat	512	0
m1.tiny	512	1
m1.small	2048	20
m1.medium	4096	40
m1.large	8192	80
m1.xlarge	16384	160

Tabela 4: Memória e espaço em disco alocado para cada Flavor

na base de dados, o processo de criação de instâncias será iniciado.

Para iniciar as instâncias são necessários os seguintes atributos:

- *Id do Flavor*: este atributo especifica a memória a ser utilizada pela instância e o espaço alocado em disco. Na tabela 4, são especificados os requisitos citados acima de acordo com o tipo de flavor;
- *Id da Image*: Este atributo especifica a imagem para criar as instância. No caso do CloudGI são utilizadas as imagens padrões do *OpenStack* que são a Fedora e a Cirros;
- *Security-Groups*: Atributo de segurança que determina, para administradores e clientes, o tipo de trafego e direção de uma porta especifica. Para o CloudGI, foi utilizado o grupo de segurança *default* do *OpenStack*;
- Key-name: Par de chaves que dá acesso SSH para as instâncias. Para O CloudGI foi criado um especifico, denominado KeyPair01;
- Nome da réplica: Que é composto pelo nome fornecido pelo usuário para iniciar o grupo de instâncias, o *id* do cliente, o *id* do serviço e o número da instância.

Após criação das instâncias é necessário atualizar no banco o *status* e o *IP* de cada uma. Para isto, é necessário buscar na nuvem da *OpenStack* as réplicas criadas para encontrar estas informações e realizar a atualização. Finalizando este processo, o serviço foi iniciado com sucesso.

Outra funcionalidade disponibilizada para o usuário é a exclusão de um serviço, que elimina todas as réplicas agregadas a um determinado serviço. Primeiro, todos os dados da base de dados são excluídos, depois cada instância é deletada da nuvem com o comando (*nova delete nome_instancia*).

O módulo *desktop* foi criado para garantir a estabilidade das instâncias criadas. Para isto é necessário monitorá-las constantemente.

Para monitoramento das instâncias o CloudGI verifica o *status*. Para isto, se torna necessário atualização constante do estado das réplicas, onde o gerenciador de tempo em tempo busca no servidor a lista de todas as instâncias criadas para atualização na base de dados.

Para verificar se uma réplica está em perfeito funcionamento, periodicamente, o CloudGI faz *ping* para cada instância que está na nuvem, enviando 4 mensagens. Caso a réplica não receba nenhum dos pacotes enviados, o *status* da mesma é verificado e a devida ação para restaurar a instância será executada.O status das réplicas e as ações necessárias para recuperação das mesmas esta descrito na tabela 5. A Tabela 3 apresenta os comandos necessário para excluir, reiniciar, criar e etc.

Este módulo também realiza o rejuvenescimento de instâncias criadas para tolerar falhas bizantinas. Periodicamente é reiniciado um replica de cada serviço.

A listagem das réplicas neste módulo, retorna todas as instâncias que constam na nuvem da *Opensatck*, com informações como nome, status, IP, id e etc.

7. AVALIAÇÃO DE FUNCIONALIDADES

7.1 Ambiente de experimentação

O CloudGI foi testado em uma máquina com processador *Intel Core i7, 8 Gb de RAM*, placa de vídeo *Geforce 2 Gb e 1 Tb de HD*. O sistema operacional utilizado foi o *Windows 8*. Neste, foi instalado o *VMware Play 7*, e criada uma máquina virtual com o sistema operacional *Ubuntu Server 14.04 (Trust)* com as seguinte especificações: *2 Gb* de RAM, processador *Virtualize Intel VT-x EPT or AMD-V/RVI*, *30 Gb de HD* e rede *Bridget*. Depois de instalado o SO, o script da *devstack* [37] foi executado para instalar a nuvem e serviços da *OpenStack* [5].

7.2 Cenários avaliados

- Teste Funcional: Neste cenário foram avaliados as funcionalidades do sistema. Para isso, foram provocados, propositalmente, nas máquinas ativas, estados nos quais haveriam a necessidade de restauração, reinicialização ou outras ações para retomar as instâncias. Para este teste, 10 réplicas foram iniciadas por n clientes. Estas instâncias foram pausadas, desligadas, arquivadas, reiniciadas e suspensas. Também foi analisado o que acontece quando uma réplica está em estado de erro 5.

7.3 Resultados e discussão

7.3.1 Teste Funcional

Nesta análise, instâncias foram pausadas, arquivadas, suspensas, desligadas e reiniciadas, com o intuito de comprovar as funcionalidades do sistema. Primeiramente, foram criadas 10 réplicas por 3 usuários, com tipo de falhas e serviços aleatórios. Todas foram criadas com sucesso, apresentando estado ativo, como ilustra a Figura 14. O *display* de *log* do sistema apresenta as seguintes informações: o *Check Time*, que apresenta o número da verificação, juntamente com a hora e a data; o nome da máquina; o *IP*; e o número de pacotes recebidos, sendo que para cada maquina são dados 2 *pings* para verificar se a mesma está respondendo, caso o resultado seja 0, para o número de pacotes, significa que algum problema está acontecendo.

A primeira análise teve como objetivo provar que ao encontrar uma instância pausada, a aplicação restauraria para seu estado ativo. Uma réplica foi escolhida aleatoriamente e

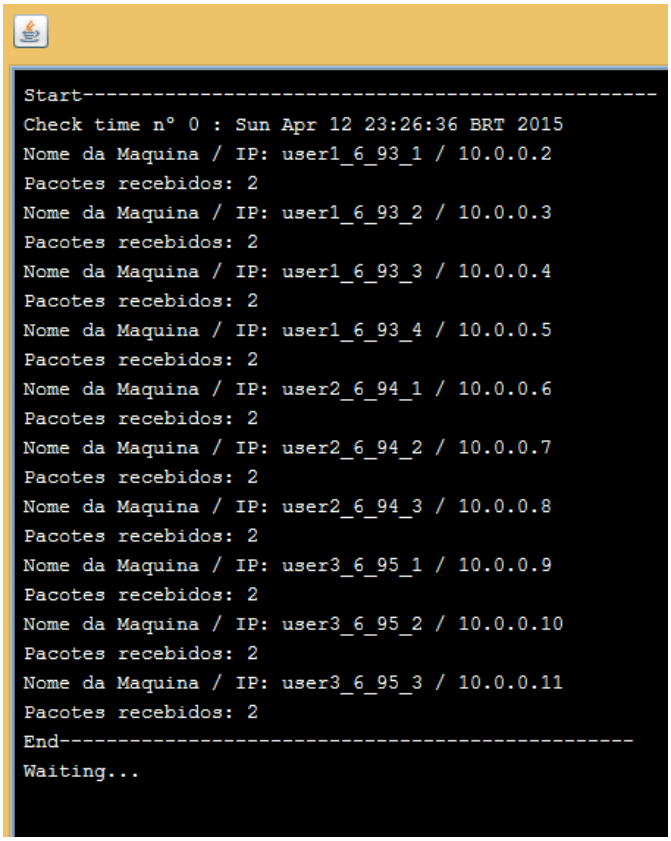

Figure 14: CloudGI Desktop: Display de *log* do sistema. Réplicas Ativas.

pausada. Como ilustra a Figura 15, após detecção do estado da máquina denominada *user1_6_93_1* como *PAUSED* e por não ter recebido nenhum dos pacotes enviados, a mesma foi despausada e na próxima verificação seu estado já foi restabelecido.

Logo depois, uma máquina denominada *user2_6_93_2* foi suspensa. Após detecção o CloudGI, restaurou o seu estado. Como ilustra a Figura 16.

Em seguida uma réplica denominada *user2_6_93_3* foi desligada, apresentando o estado *SHUTOFF*. Ver Figura 17, que ilustra que na próxima verificação seu estado foi restaurado.

A Figura 18 ilustra uma instância que foi arquivada e logo em seguida desarquivada pelo CloudGI.

Para testar o caso de uma instância ser criada com erro ou apresentar este estado após um tempo, duas réplicas foram criadas. As mesma formam iniciadas com o estado de *ERROR*. Ver Tabela 5. A Figura 21, mostra as máquinas após a criação, apresentando o *status* avaliado em questão, com isso a réplica foi excluída e depois recriada com o comando *nova boot* 3. Pode-se notar que na próxima verificação a o estado da réplica já foi restabelecido.

Para finalizar a análise, foi verificado como o CloudGI se comporta diante de uma instância que está sendo reiniciada.

Status	Descrição	Restauração
ACTIVE	A réplica está ativa	-
BUILD	A réplica está em processo de inicialização	Aguardar a mesma ser iniciada para verificar o *status*
ERROR	a réplica falhou em sua criação ou ocorreu algum problema durante a sua execução	reiniciar a réplica / (*reboot*) excluir e iniciar uma nova instância com o mesmo nome (*delete e boot*)
PAUSE	uma réplica foi pausada	despausar a instância (*unpause*)
SHELVE	uma réplica esta arquivada	desarquivar réplica (unshelve)
SUSPEND	uma réplica foi suspensa	acordar a maquina (resume)
SHUTOFF	uma réplica foi desligada	ligar a maquina (start)

Tabela 5: Status das instâncias

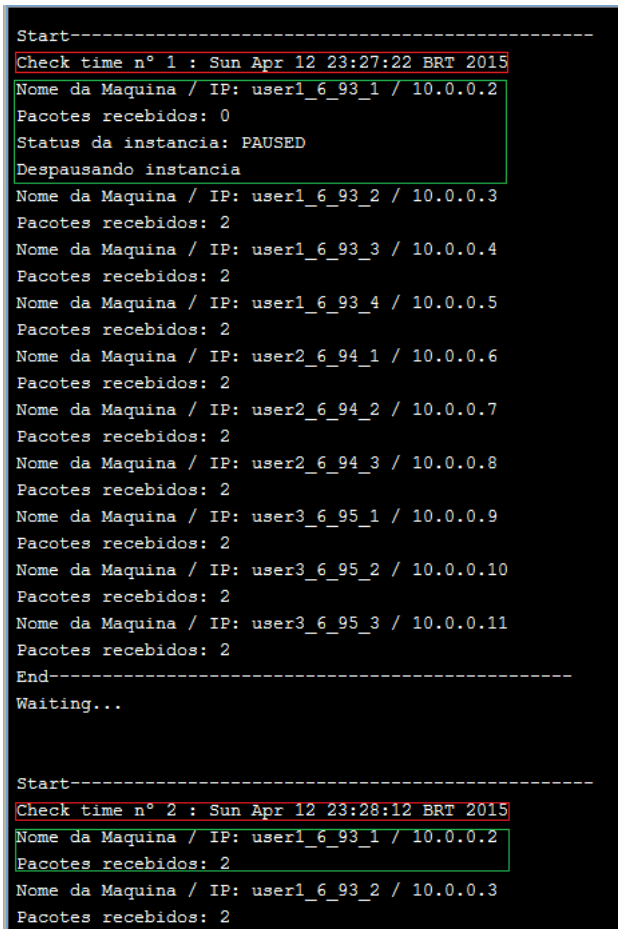

Figure 15: CloudGI Desktop: Display de *log* do sistema. Réplica Pausada.

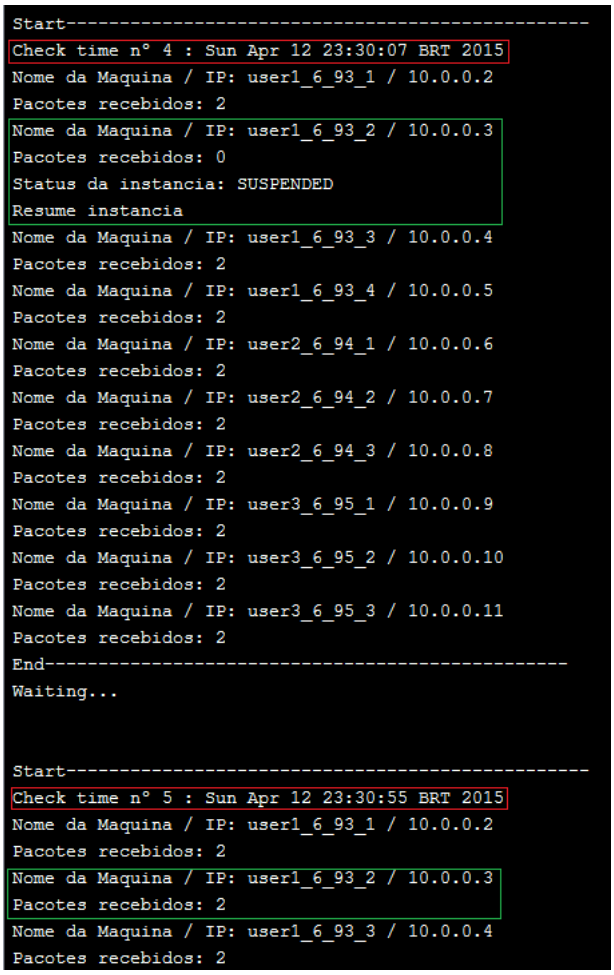

Figure 16: CloudGI Desktop: Display de *log* do sistema. Réplica Suspensa.

Pode-se constatar com a Figura 19 que o mesmo aguarda a máquina reiniciar.

8. CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho, apresentamos o CloudGI, um gerenciador de instâncias de réplicas em um ambiente de computação em nuvem IaaS. Esta aplicação oferece um serviço de criação, monitoramento e gestão de instâncias de máquinas virtuais em um ambiente de computação em nuvem, que atuam como réplicas de um serviço, suportando os modelos de falhas *crash-recovery* ou bizantino e diferentes níveis de serviço, que caracterizam o número f de nós falhos que podem ser tolerados. Esta ferramenta foi desenvolvida em JAVA para *clouds OpenStack*, uma das mais utilizadas plataformas IaaS para nuvens privadas ou comunitárias.

Neste trabalho apresentamos a ferramenta, suas funcionalidades e uma avaliação de funcionalidade. CloudGI pode auxiliar na implantação de um serviço replicado, ao possi-

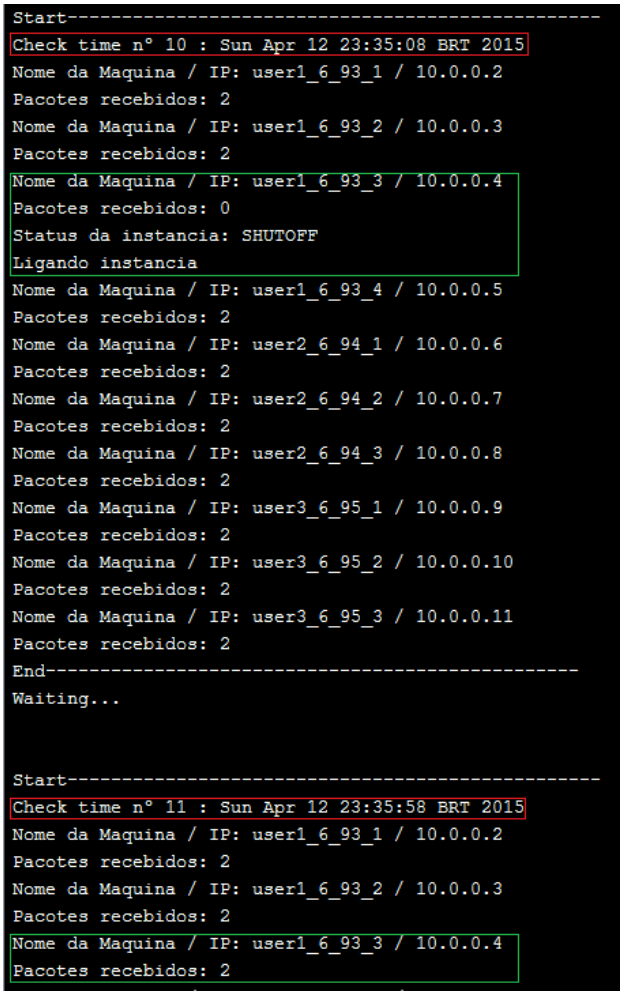

Figure 17: CloudGI Desktop: Display de *log* do sistema. Réplica Desligada.

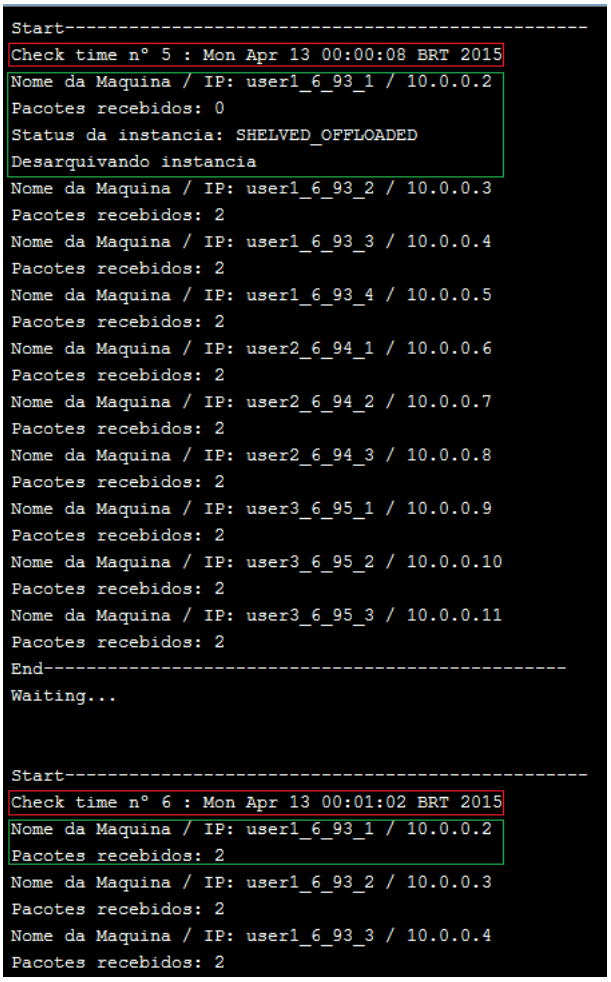

Figure 18: CloudGI Desktop: Display de *log* do sistema. Réplica Arquiva.

bilitar monitorar as máquinas virtuais em que residem as réplicas da aplicação, bem como interagir com a plataforma de computação em nuvem, recuperando réplicas quando possível.

CloudGI, desta forma, interage com o usuário, ao configurar o serviço replicado, de acordo com o tipo de falha a ser tolerada. A mesma cria e apresenta o estado das instâncias, e com a plataforma de computação em nuvem *OpenStack*, atua sob o funcionamento das máquinas virtuais, quando necessário.

Como funcionalidade futura, o CloudGI deve permitir que haja interação entre o middleware de replicação utilizado e a ferramenta proposta, de modo a prover uma interface que apresente uma visão e permita monitoria da infraestrutura de nuvem diretamente para o serviço replicado.

9. REFERÊNCIAS

[1] HANNA, E. M., MOHAMED, N., AL-JAROODI, J. (2012). **The Cloud: Requirements for a Better Service**. 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing.

[2] HORRIGAN, J. B. Use of cloud computing applications and services. The Pew Internet and American Life Project, 2008, Abril-Maio, p. 9.

[3] PRETI, J., FILGUEIRAS, L. (2010). **Interação em Nuvens**. Proceedings of the IX Symposium on Human Factors in Computing Systems. p. 209-212

[4] VAQUERO, L. M., ROMERO-MERINO, L., CACERES, J., LINDNER, M. (2009). **A break in the clouds: Towards a cloud definition**. SIGCOMM Comput. Commun. Rev., vol. 39, pp. 50-55.

[5] OpenStack: http://www.openstack.org/ - Acesso: janeiro, 2015.

[6] ROSADO, T., BERNARDINO, J.**An Overview of OpenStack Architecture**. Polytechnic Institute of Coimbra. ISEC – Coimbra Institute of Engineering. 2014, july.

[7] CASTILLO, J. A. L., MALLICHAN, K., AL-HAZMI, Y. **OpenStack Federation in Experimentation Multi-cloud Testbeds**. IEEE International Conference on Cloud Computing Technology and Science, 2013.

[8] BIST, M., WARIYA, M., AGARWAL, A. **Comparing Delta, Open Stack and Xen Cloud Platforms: A**

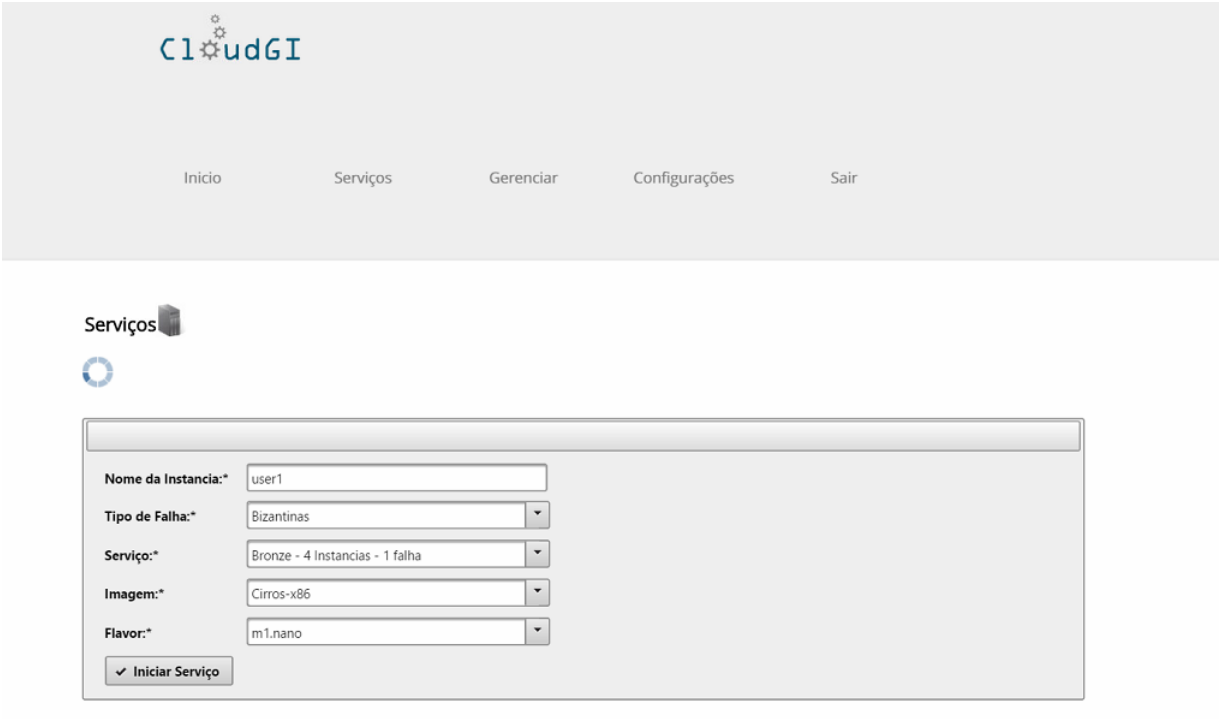

Figure 20: CloudGI WEB: Tela de cadastro de serviço

```
Start--------------------------------------------------
Check time n° 0 : Tue Apr 14 10:39:35 BRT 2015
Status da instancia: ERROR
Excluindo instancia
nova boot --flavor 42 --image 90b23698-66e0-42ef-9f72-5830077da934 --security-groups default --key-name KeyPai
r01 user1_6_97_1

Status da instancia: ERROR
Excluindo instancia
nova boot --flavor 42 --image 90b23698-66e0-42ef-9f72-5830077da934 --security-groups default --key-name KeyPai
r01 user1_6_97_2

End----------------------------------------------
Waiting...


Start--------------------------------------------------
Check time n° 1 : Tue Apr 14 10:41:05 BRT 2015
Nome da Maquina / IP: user1_6_97_1 / 10.0.0.2
Pacotes recebidos: 2
Nome da Maquina / IP: user1_6_97_2 / 10.0.0.3
Pacotes recebidos: 2
End----------------------------------------------
Waiting...
```

Figure 21: CloudGI Desktop: Display de log do sistema. Réplica com estado de Erro.

Survey on Open Source IaaS. IEEE, 2012.

[9] LITVINSKI, O., GHERBI, A. **Openstack Scheduler Evaluation using Design of Experiment Approach**. Department of Software and IT Engineering. École de technologie supérieure. IEEE, 2013.

[10] WEBER, T. S. **Um roteiro para exploração dos conceitos básicos de tolerância a falhas**. Instituto de Informática – UFRGS.

[11] LUNKAD, S. **Working with Devstack**. January 18,

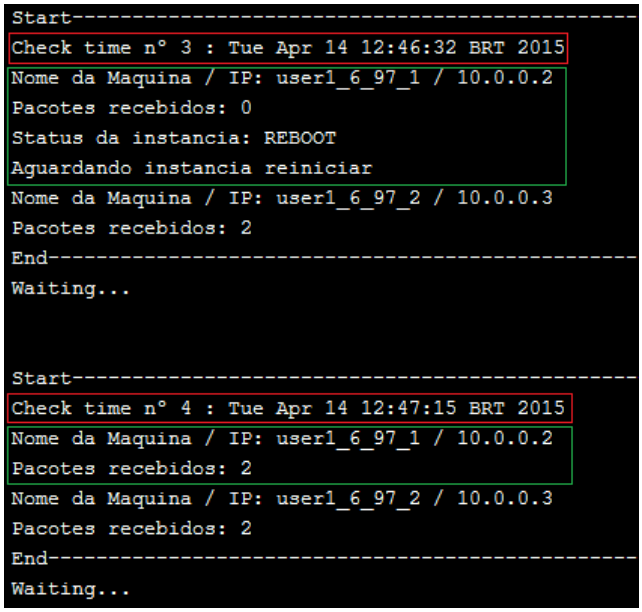

Figure 19: CloudGI Desktop: Display de *log* do sistema. Réplica reiniciando.

2014. Disponível em: http://sayalilunkad.github.io/posts/Devstack/

[12] THAMPI, S. M. **Introduction to Distributed Systems**. L.B.S INSTITUTE OF TECHNOLOGY FOR WOMEN. India

[13] ASPNES, J. **Notes on Theory of Distributed Systems**. Spring, 2014.

[14] COSTAN, A., DOBRE, C., POP, F., LEORDEANU, C., CRISTEA, V.**A Fault Tolerance Approach for Distributed Systems Using Monitoring Based replication**. IEEE. 2010.

[15] ANDERSON, R. **Security Engineering**. 2º Edição. Capitulo 6.

[16] LAMPORT, L.**The weak byzantine generals problem**. J. ACM, 1987.

[17] LIMA, M. S., GREVE, F. G. P. **Detectando Falhas Bizantinas em Sistemas Distribuídos Dinâmicos**. SBRC, 2009.

[18] KIHLSTROM, K. P., MOSER, L. E., MELLIAR-SMITH, P. M. **Byzantine Fault Detectors for Solving Consensus**. THE COMPUTER JOURNAL, Vol. 46, No. 1, 2003.

[19] GORENDER, S., MACÊDO, R. J. A. **Um Modelo para Tolerância a Falhas em Sistemas Distribuídos com QoS**. LaSiD - UFBA.

[20] HADZILACOS, V., TOUEG, S. **A modular approach to the specification and implementation of fault-tolerant broadcasts**. Technical Report TR94-1425. Department of Computer Science, Cornell University, Ithaca NY. May 1994.

[21] CHARRON-BOST, B., PEDONE, F., SCHIPER, A. **replication: Theory and Practice**. Springer, 1998.

[22] MA, T., Hillston, J., ANDERSON, S. **Evaluation of the QoS of Crash-Recovery Failure Detection**. LFCS, School of Informatics The University of Edinburgh Edinburgh, UK.

[23] FRAGA, E., BRASILEIRO, F., BRILHANTE, J., COSTA, R., SENGER, H., SILVA, P. A., NAVAUX, P., SÁ, A. S., MACÊDO, R. J. A. **Just-in-Time Clouds: Uma abordagem para Federação de Clouds Privadas**. In: XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - Salão de Ferramentas, 2013, Brasilia. Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. Porto Alegre: SBC, 2013. v. 1. p. 1-10.

[24] JHAWAR, R., PIURI, V., SANTAMBROGIO, M. **A comprehensive conceptual system-level approach to fault tolerance in cloud computing**. In: Systems Conference (SysCon), 2012 IEEE International. IEEE, 2012. p. 1-5.

[25] RODERO-MERINO, Luis et al. *From infrastructure delivery to service management in clouds*. Future Generation Computer Systems, v. 26, n. 8, p. 1226-1240, 2010.

[26] **Java SE Technologies - Database**. Oracle. Disponível em: http://www.oracle.com/technetwork /java/javase/jdbc/index.html. 27 de fevereiro de 2015.

[27] **Netbeans**. Netbeans. Disponível em: https://netbeans.org/. 01 de março de 2015.

[28] **Glassfish**. Glassfish. Disponível em: https://glassfish.java.net/. 01 de março de 2015.

[29] **Model View Controller (MVC) architecture**. Disponível em: http://poincare.matf.bg.ac.rs/ andjelkaz/pzv/cas4/mvc.pdf. 01 de março de 2015.

[30] Castro, M., Liskov, B. **Practical Byzantine fault tolerance**. Proc. of the 3rd Symp. on Operating Systems Design and Implementation (OSDI). p. 173-186, 1999.

[31] Fontán, J., Vázquez, T., Gonzalez, L., Montero R. S., Llorente, I. **OpenNEbula: The open source virtual machine manager for cluster computing**. Book of Abstracts of Open Source Grid and Cluster Software Conference, 2008.

[32] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D. **The eucalyptus open-source cloud-computing system**. Proc. of 9th IEEE/ ACM Int. Symp. on Cluster Computing and the Grid (CCGRID). p. 124-131, 2009.

[33] Keahey, K., Figueiredo, R., Fortes, J., Freeman, T., Tsugawa, M. **Science clouds: Early experiences in cloud computing for scientific applications**. Cloud computing and applications. v. 2008. p. 825-830, 2008.

[34] Zhang, Q., Cheng, L., Boutaba, R. **Cloud computing: state-of-the-art and research challenges**.Journal of internet services and applications. Springer, v. 1. p. 7-18,2010.

[35] , Sefraoui, O., Aissaoui, M., Eleuldj, M.**OpenStack: toward an open-source solution for cloud computing**. Int. Journal of Computer Applications. Citeseer, v. 55. p. 38-42, 2012.

[36] Amazon. **Amazon Elastic Computing Cloud**. http://aws.amazon.com/ec2, 2015.

[37] DevStack: http://docs.openstack.org/developer/devstack/ - Acesso: março, 2015.

[38] Gonçalves, L. A. *SARA-ES: Sistema para Acompanhamento de Registros Acadêmicos do Ensino Superior*.

[39] Wen, Xiaolong and Gu, Genqiang and Li, Qingchun and Gao, Yun and Zhang, Xuejie. **Comparison of open-source cloud management platforms: OpenStack and OpenNebula**. Proc. of 9th Int. Conf. on Fuzzy Systems and Knowledge Discovery (FSKD). p. 2457–2461, 2012.

[40] Sempolinski, Peter and Thain, Douglas. **A comparison and critique of eucalyptus, opennebula and nimbus**. Proc. of the IEEE 2nd Int. Conf. on Cloud Computing Technology and Science (CloudCom). p. 417–426, 2010.

[41] **Grouppac**. Departamento de Automação e Sistemas (DAS). Universidade Federal de Santa Catarina (UFSC). Disponível em: http://grouppac.sourceforge.net/. Maio, 2015.

[42] **JOnAS**. Disponível em: http://jonas.ow2.org/xwiki/bin/view/Main/. Maio, 2015.