Alta Disponibilidade de um Gerenciador de Nuvem IaaS Baseada em Replicação: Experiência & Resultados

Gustavo B. Heimovski[1], Rogério C. Turchetti[1], Juliano A. Wickboldt[2], Lisandro Z. Granville[2], Elias P. Duarte Jr[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.018 – 81.531-980 – Curitiba – PR – Brasil

[2]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre - RS - Brasil

{gbheimovski,rcturchetti,elias}@inf.ufpr.br

{jwickboldt,granville}@inf.ufrgs.br

Abstract. *In this work we report the experience of adding a high availability solution to an IaaS (Infrastructure as a Service) cloud platform called Aurora. The proposed solution is based on the multi-master replication of the cloud manager, replicas form a cluster of cloud managers running on multiple datacenters. The implementation also includes a monitoring service for the replicas. The high availability solution is fully integrated to the cloud platform so that activation is done directly by pushing a button of the Graphical User Interface. The performance and robustness of the proposed solution were evaluated experimentally. The time to (1) incorporate a new manager instance to a cluster; (2) recover an instance after a failure and (3) replicate data in different scenarios were measured and are reported in this paper. The impact of the proposed solution cost is evaluated by measuring CPU and network usage. A stress test in which the link delay between two datacenters grows up to the operating limit of the replication solution is also reported. Finally, we present a table for the system availability as a function of the MTBF (Mean Time Between Failures).*

Resumo. *Este trabalho relata a experiência de acrescentar uma solução de alta disponibilidade à plataforma Aurora de gerência de recursos de nuvens IaaS (Infrastructure as a Service). A solução proposta facilita a ativação da alta disponibilidade, disparando a replicação multi-master do gerenciador, criando um cluster de gerenciadores sobre múltiplos datacenters. A solução também provê o monitoramento das múltiplas instâncias replicadas. A ativação de alta disponibilidade é feita diretamente na interface do gerenciador. O desempenho e a robustez da solução proposta foram avaliados experimentalmente. São reportados os tempos medidos para (1) incorporar uma nova instância do gerenciador a um cluster; (2) a recuperação após a ocorrência de uma falha e (3) a latência para a replicação dos dados, em diversas configurações. O custo da solução proposta foi avaliado em termos de uso de processador e rede. É reportado também um teste de stress em que o atraso de comunicação entre dois datacenters cresce até o limite de funcionamento da solução de replicação utilizada. Por fim, é apresentada uma tabela com a avaliação da disponibilidade do sistema como função do MTBF (Mean Time Between Failures).*

1. Introdução

A computação em nuvem permite, de forma conveniente, o acesso sob demanda a um conjunto compartilhado de recursos configuráveis de computação, incluindo, redes, servidores, unidades de armazenamento, aplicações e serviços [Armbrust et al. 2010, Zhang et al. 2010]. Estes recursos podem ser rapidamente provisionados e liberados com um esforço mínimo por parte do cliente. Um dos modelos de serviço da computação em nuvem é a Infraestrutura como Serviço (*Infrastructure as a Service* - IaaS), que provê ao usuário recursos como processamento, armazenamento e rede para execução de suas aplicações e sistemas [Mell and Grance 2011, Jhawar and Piuri 2012, Tsakalozos et al. 2011].

Atualmente, observam-se variados tipos de serviços oferecidos pelos provedores de nuvens computacionais, por exemplo, próximo de 61% das atividades comerciais no Reino Unido utilizam algum tipo de serviço em computação em nuvem [Arean 2013]. Entretanto, ainda existem diversos desafios que precisam ser tratados, como: segurança, gerenciamento de riscos, e a alta disponibilidade [Kholghi et al. 2014]. A alta disponibilidade é essencial para permitir que aplicações críticas façam uso da nuvem. Em particular, o gerenciador de nuvem sempre deve se manter acessível para que o cliente ou administrador da nuvem consiga utilizá-lo a qualquer instante. O cliente deve ser capaz de acessar os dados de qualquer gerenciador a qualquer momento, da forma mais transparente possível.

Desta forma, é desejável que uma infraestrutura em nuvem ofereça soluções que permitam o acesso aos recursos mesmo diante de falhas. A proposta do presente trabalho surge neste contexto: desenvolver uma infraestrutura de nuvem robusta, baseada na replicação do gerenciador, que seja ativada de forma simples, diretamente na interface do usuário. Apesar de existirem plataformas de nuvem que possuem mecanismos para tal replicação, todas exigem profundo conhecimento do sistema e um exaustivo esforço para configurar todos os procedimentos até a sua execução.

A estratégia proposta no trabalho envolve a replicação de diversos componentes que mantêm o gerenciador da nuvem disponível, mesmo diante de possíveis falhas. Desta forma, a estratégia permite a replicação e o monitoramento das múltiplas instâncias do gerenciador replicadas. A solução proposta contempla tanto cenários em que há um único *datacenter* com duas ou mais instâncias replicadas da nuvem, quanto cenários em que há dois ou mais *datacenters*, cada um com recursos próprios. Nos dois cenários, cada instância do gerenciador possui acesso a todos os recursos da nuvem. Em caso de falha afetando uma das instâncias, outra instância do gerenciador tem a capacidade de gerenciar seus recursos da nuvem.

No cenário em que existem dois ou mais *datacenters* remotos (conectados pela Internet), a principal motivação é que, caso ocorra falha em um dos *datacenters*, outras instâncias consigam acessar os recursos do *datacenter* que falhou. A ideia de permitir que um gerenciador tenha completo acesso de todos os recursos, não é somente para o caso em que acontecem falhas, mas para que também seja possível utilizar os recursos do outro *datacenter* a qualquer momento, permitindo o compartilhamento de recursos entre nuvens. Além disso, a estratégia proposta também realiza o balanceamento de carga entre vários gerenciadores dos *datacenters*.

A infraestrutura em nuvem utilizada para implementação da estratégia proposta é

denominada de *Aurora Cloud Manager* [Wickboldt et al. 2014]. O Aurora permite o gerenciamento dos recursos da nuvem de forma flexível pela adição de 'programabilidade'. A programabilidade é obtida através de uma API orientada a objetos, na qual os administradores da nuvem conseguem descrever e executar programas personalizados para a otimização e implantação da própria infraestrutura. Para tornar robusto o gerenciador, é implementada uma solução de replicação que utiliza uma abordagem de replicação *multi-master* [Charron-Bost et al. 2010]. A replicação é realizada em duas partes. A primeira parte é executada através da replicação do banco de dados que armazena informações importantes para realizar as ações de gerenciamento. A segunda parte da replicação é realizada utilizando a sincronização de diretórios. A replicação de diretórios é necessária, pois as imagens das máquinas virtuais, os programas de otimização e implantação, além das métricas, ficam armazenados em diretórios distintos, e não no banco de dados.

O desempenho e a robustez da solução proposta foram avaliados experimentalmente. São reportados os tempos medidos para (1) incorporar uma nova instância do gerenciador a um cluster; (2) a recuperação após a ocorrência de uma falha e (3) a latência para a replicação dos dados, em diversas configurações. O custo da solução proposta foi avaliado em termos de uso de processador e rede. É reportado também um teste de stress em que o atraso de comunicação entre dois datacenters cresce até o limite de funcionamento da solução de replicação utilizada. Por fim, é apresentada uma tabela com a avaliação da disponibilidade do sistema como função do MTBF (*Mean Time Between Failures*).

O restante deste trabalho está organizado da seguinte forma. Na seção 2 são descritos ambientes para execução e gerenciamento de recursos em nuvens computacionais e, com base nisto, é feita uma análise comparativa. Na seção 3 é descrito o gerenciador de nuvem *Aurora Cloud Manager*. Na seção 4 são apresentadas as definições e o modelo do gerenciador Aurora Robusto. Os experimentos realizados e suas análises são apresentados na seção 5. A seção 6 apresenta a conclusão e os trabalhos futuros.

2. Trabalhos Relacionados

Nesta seção são descritos ambientes para execução e gerenciamento de recursos em nuvens computacionais, com foco em suas estratégias de alta disponibilidade.

O OpenStack [1] é uma plataforma IaaS em nuvem que oferece capacidade para controlar um grande número de recursos e sua infraestrutura oferece a utilização dos recursos sob demanda. O OpenStack possui duas estratégias para a sua alta disponibilidade [2]. A primeira utiliza a replicação *master-slave*, utilizando o DRBD[3] para replicação do banco de dados. A segunda estratégia utiliza a replicação *multi-master*, onde o banco de dados é replicado através do Galera. Nesta estratégia também é utilizado um balanceador de carga, como o *HAProxy*.

O CloudStack [4] é uma plataforma que possibilita o desenvolvimento de aplicações e gerenciamento de recursos em uma infraestrutura IaaS. Para aumentar a disponibilidade do gerenciador, o próprio operador da nuvem pode replicar o servidor de gerenciamento.

[1] http://openstack.org/
[2] http://docs.openstack.org/ha-guide/
[3] http://drbd.linbit.com/
[4] https://cloudstack.apache.org/

Entretanto, eventos de datacenters diferentes não são integrados e a replicação deve ser habilitada em cada datacenter [5]. Em relação a disponibilidade do sistema de armazenamento, para evitar um único ponto de falha, pode ser implementado em cada zona, um *primary database* e várias réplicas que são sincronizadas com a réplica primária.

O Eucalyptus [6] é uma infraestrutura IaaS projetada para implementar, gerenciar e manter nuvens tanto privadas quanto híbridas. A arquitetura oferece mecanismos para tolerar falhas no nível dos hosts através da substituição imediata do host inoperante, tornando transparente a falha para o usuário. A alta disponibilidade dos serviços do gerenciador estão mapeadas para futuras versões do gerenciador [7], mas ainda não estão totalmente implementadas.

O OpenNebula [8] é uma plataforma em nuvem que traz diversas funcionalidades através de uma arquitetura simples, mas com diversos recursos para a construção de nuvens híbridas. O OpenNebula fornece um conjunto de ferramentas (*toolkit*) para o gerenciamento de infraestruturas heterogêneas sendo compatível com diversas plataformas e hipervisores. O OpenNebula possui suporte para tolerância a falhas e recuperação dos hosts ou das máquinas virtuais que deixarem de executar suas funções prematuramente. O OpenNebula utiliza somente a abordagem *master-slave* para a sua própria replicação. O banco de dados é replicado com o DRBD.

3. O Gerenciador de Nuvens IaaS Aurora

O *Aurora Cloud Manager* [Wickboldt et al. 2014] é uma plataforma de gerenciamento de nuvem que provê infraestrutura como um serviço e permite o gerenciamento dos recursos da nuvem de forma flexível pela adição de 'programabilidade'. A programabilidade é obtida através de uma API orientada a objetos, na qual os administradores da nuvem conseguem descrever e executar programas personalizados para otimização e implantação da própria infraestrutura.

Os recursos providos por uma plataforma de gerenciamento de nuvem são de três tipos básicos: computação, armazenamento e rede. O gerenciamento de recursos de computação é relacionado com a manipulação das máquinas virtuais, que devem ter CPU e memória alocados, por exemplo. A gerência de armazenamento envolve tarefas relacionadas com a alocação de volumes para armazenamento de dados persistentes e, por último, a gerência de recursos de rede é responsável por permitir a comunicação entre os recursos virtuais.

Para prover os recursos de computação e armazenamento, o Aurora suporta tecnologias de virtualização através da *Libvirt*[9], biblioteca responsável por implementar a comunicação com diversos *hypervisors*. Entre os *hypervisors* suportados estão

[5]http://docs.cloudstack.apache.org/projects/cloudstack-administration/en/4.8/reliability.html#limitations-on-database-high-availability

[6]http://www.eucalyptus.com/

[7]https://docs.eucalyptus.com/eucalyptus/4.0.2/install-guide/ha_planning.html

[8]https://opennebula.org/

[9]http://www.libvirt.org

KVM/QEMU[10], *XEN*[11], *LXC*[12], *VirtualBox*[13] e *VMware*[14]. Para prover os recursos de rede, o Aurora utiliza as redes definidas por software (*Software Defined Networks SDN*) através do *Openflow* [McKeown et al. 2008]. Existe também uma integração com o *framework* de monitoramento dos recursos da nuvem *FlexACMS* [Barbosa de Carvalho et al. 2013] e o tradicional sistema de gerência de redes *Nagios*[15].

Na arquitetura do Aurora existem dois tipos de usuários, o usuário final e o administrador do sistema, como vistos no topo da figura 1, que ilustra a arquitetura do gerenciador de nuvem. Os principais módulos do Aurora incluem, o módulo da interface gráfica, *Graphical User Interface (GUI)*, que é composta pela interface com o usuário e a interface com o administrador. O segundo módulo é o *Slice Space*, que possui o componente responsável por transformar as requisições do usuário em um formato interno. Outro módulo é o *Programmable Space*, que possui os componentes responsáveis pela programabilidade do Aurora. Existem outros módulos que são: o *Event Space*, que gerencia eventos, expressos em forma de condições ou alarmes; a *Unified API*, que é responsável pela manipulação dos recursos virtuais, como máquinas virtuais, interfaces de rede, alocação de volumes virtuais, entre outros; E, por último, os *Drivers*, que fazem a abstração dos dispositivos virtuais.

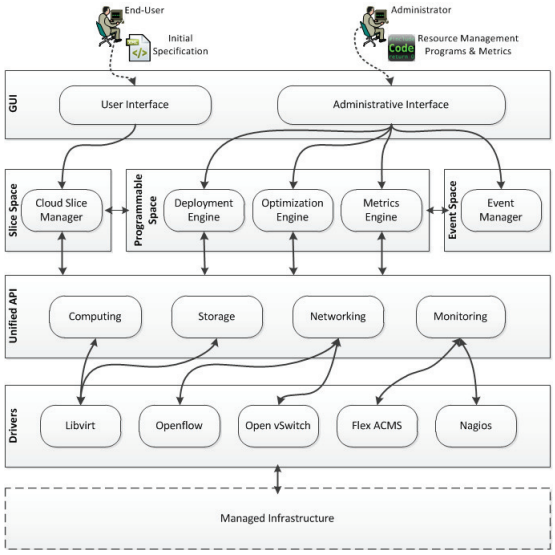


Figura 1. Arquitetura da plataforma de Gerenciamento de nuvem Aurora [Wickboldt et al. 2014].

O componente *Cloud Slice Manager* do módulo *Slice Space* transforma o documento de especificação inicial em um formato interno, chamado *Cloud Slice*. Um *Cloud Slice* agrega todos os diferentes tipos de recursos (computação, armazenamento e rede) que compõem a infraestrutura virtual sobre a qual a aplicação é implantada. Após a criação de um *slice*, os componentes dos módulos *Programmable Space*, *Event Space*

[10]http://www.linux-kvm.org/,http://qemu.org
[11]http://xenserver.org/
[12]https://linuxcontainers.org/
[13]https://www.virtualbox.org/
[14]http://www.vmware.com/
[15]http://www.nagios.org/

e *Slice Space* da arquitetura interagem para organizar o gerenciamento de recursos. Os componentes da arquitetura que adicionam flexibilidade diretamente ao núcleo da plataforma são *Deployment Engine*, *Optimization Engine* e *Metrics Engine*, presentes no módulo *Programmable Space*. Estes componentes operam com base nos programas e métricas de gerenciamento de recursos, na figura: *Resource Management Programs & Metrics*, escritos pelo administrador.

Todas as operações que envolvem a manipulação dos recursos virtuais realizadas durante implantação e otimização da *Cloud Slice* utilizam o módulo *Unified API*. A *Unified API* consiste dos componentes *Computing*, *Storage*, *Networking* e *Monitoring*, descritos a seguir. O componente *Computing* é responsável pelas funcionalidades das máquinas virtuais, como criação e migração, e também das imagens dos sistemas operacionais que são instaladas nas máquinas virtuais. O componente *Storage* lida com a alocação de volumes de armazenamento virtual. O componente *Networking* implementa interfaces para criação de links virtuais e roteadores virtuais. Existe um último componente, que é o *Monitoring*. Ele é responsável por configurar a infraestrutura de monitoração no exato momento em que uma *Cloud Slice* é implantada, além disso, implementa uma abstração de evento, que define alarmes para serem acionados a partir da infraestrutura de monitoração e utilizados pelo componente *Event Manager*.

As abstrações dos dispositivos virtuais são implementadas por um conjunto de *drivers* disponíveis no módulo localizado na parte inferior da arquitetura. Esses *drivers* são para tecnologias específicas e possuem dois papéis: abstraem a complexidade de configuração de parâmetros e protocolos de comunicação e deixam a plataforma do Aurora mais portável. Atualmente são utilizadas a *Libvirt*[16] para os componentes *Computing* e *Storage*, *Openflow* e *OpenVSwitch*[17] para *Networking*, e por último, *FlexACMS* e *Nagios* para o componente *Monitoring*.

O núcleo da plataforma Aurora é escrito na linguagem *Python* e implementado como uma aplicação *Web*, utilizando o *framework Django* [18].

4. Gerenciador Aurora Robusto

Esta seção descreve a plataforma robusta para gerência de recursos em nuvens IaaS baseada no gerenciador Aurora.

Concretamente, este trabalho tem como objetivo permitir a operação contínua do próprio ambiente de nuvem. A estratégia adotada é baseada na replicação do Aurora, mais especificamente do próprio gerenciador da nuvem. A replicação do Aurora é realizada em duas partes. A primeira parte é através da replicação do banco de dados MySQL, que é utilizado pelo Aurora para armazenar as informações de gerenciamento. Estas informações de gerenciamento são as seguintes: os *hosts*, com suas informações de endereço IP, nome, entre outras, a lista de máquinas virtuais instaladas nos *hosts*, roteadores, *switches*, a lista de programas de otimização e implantação das *Cloud Slices*, entre outras informações. A segunda parte é através da replicação de diretórios, onde o *rsync* [Tridgell 1999] é utilizado. Esta replicação de diretórios é necessária, pois as imagens utilizadas para criação de

[16]http://www.libvirt.org
[17]http://openvswitch.org/
[18]https://www.djangoproject.com

novas máquinas virtuais, os programas de implantação e otimização, além das métricas, ficam armazenados em diretórios, e não no banco de dados.

A figura 2 ilustra o funcionamento do Aurora para múltiplos *datacenters* com a replicação ativada. Nela existe o administrador da nuvem que acessa o sistema através de um único IP, mas ele pode acessar individualmente qualquer uma das três instâncias de Aurora. O acesso através de um único IP foi implementado utilizando-se o balanceador de carga *HAProxy*[19], que, quando recebe um acesso, redireciona o acesso para um dos três Auroras. Isto é, quando ocorrerem múltiplos acessos para o mesmo *datacenter*, o *HAProxy* redistribui os acessos entre os Auroras presentes no mesmo ambiente replicado, utilizando, por padrão, o algoritmo de balanceamento *round-robin*. No caso de múltiplos *datacenters*, é interessante utilizar o balanceamento por origem de comunicação, assim sempre um cliente acessa um Aurora específico, e somente se não houver mais Auroras replicados neste *datacenter*, o cliente acessa o Aurora do outro *datacenter*. O *HAProxy* também foi implementado com redundância de seus serviços, neste caso se um dos servidores do *HAProxy* falhar, o outro servidor consegue assumir e fazer o balanceamento dos acessos. A identificação de falha em um dos servidores do balanceador é feita pelo *Keepalived*[20].

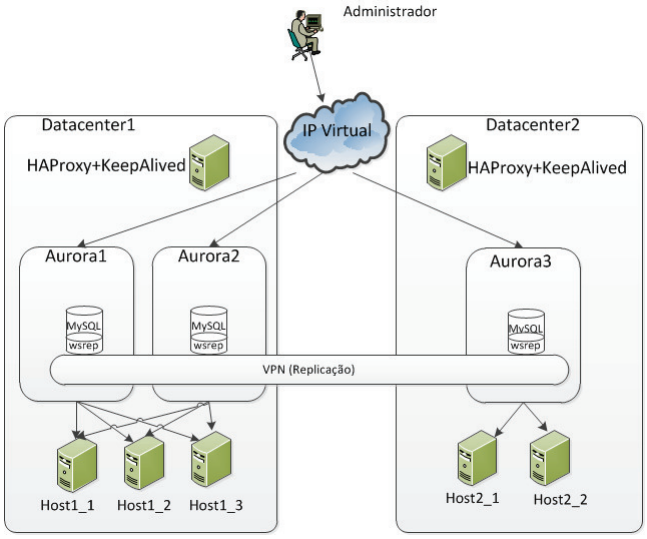


Figura 2. Arquitetura do Aurora em 2 *datacenters* com a replicação ativada.

Na figura, a replicação está ativada, então, cada instância do Aurora possui as informações das outras instâncias. Isto ocorre pois a cada inserção feita pelo administrador, por exemplo, a inserção de um novo *host* em uma instância do Aurora, ocorre a replicação para as outras instâncias. Todos os Auroras possuem acesso aos cinco *hosts*, *Host1_1*, *Host1_2*, *Host1_3*, *Host2_1* e *Host2_2*. Para a replicação funcionar é necessária a configuração de uma VPN (*Virtual Private Network*) entre os *datacenters*, pois é necessário manter um endereçamento de rede em que todos Auroras consigam se comunicar, além da segurança fornecida. Este cenário de múltiplos *datacenters* foi simulado forçando atrasos no enlace de comunicação entre eles. Esta simulação é descrita na próxima seção.

[19]http://www.haproxy.org/
[20]http://www.keepalived.org/

Neste trabalho, a abordagem de replicação escolhida foi a *multi-master*, pois permite a escrita e leitura de dados em qualquer instância independente do Aurora, mantendo os dados replicados e consistentes em todas as instâncias. A ideia principal de utilizar esta abordagem é de manter todos Auroras ativos e independentes, mas possuindo as informações dos outros, para prover a disponibilidade dos dados no caso em que ocorram falhas.

A replicação do banco de dados foi implementada através do *Galera Cluster*, que é um *plugin* do MySQL e é instalado juntamente com a plataforma Aurora. O Galera utiliza a abordagem de replicação *multi-master*, que é a abordagem de replicação escolhida para este trabalho.

A replicação do Aurora só é ativada quando o usuário habilita esta função, numa nova seção da interface gráfica do Aurora, chamada de *Aurora Cluster*. Esta seção contém a lista de Auroras participantes do *cluster*, com o estado (falho/não falho) de cada Aurora, assim como uma opção de ativação da função de replicação. Na ativação da função de replicação, o usuário é redirecionado para uma nova página informando sobre ativação da função, isto ocorre pois, para viabilizar a replicação, o serviço de *mysql* deve ser reiniciado para ser aplicado um novo arquivo de configuração do MySQL, com a informação do nome do *cluster*, endereços IP dos nós pertencentes ao *cluster*, entre outras informações. Na seção em que são listados os Auroras, o usuário deve cadastrar, em cada Aurora, quais são os Auroras participantes do *cluster* pois, assim, o sistema possuirá as informações dos participantes e também consegue ativar o serviço de replicação do Galera.

A solução proposta também possui a função de monitoramento dos nós participantes do *cluster* de Auroras. O monitoramento é feito através da ativação de um comando de notificação do Galera, que executa toda vez que existe uma alteração no *cluster* ou no estado do nó. Este comando então notifica alterações do estado dos nós para o Aurora. Quando é detectado que um Aurora falhou e não pertence mais ao *cluster*, é enviada uma notificação por *email* ao administrador com a informação do endereço IP do Aurora que falhou.

Vale ressaltar que por segurança, toda a comunicação entre os *hosts* e o Aurora é criptografada utilizando conexões via *ssh*. As chaves *ssh* já devem estar instaladas em todos os *hosts*, para que outra instância do Aurora consiga gerenciar qualquer *host* físico. Para a replicação que utiliza o *rsync*, também é necessário copiar estas chaves *ssh* para as outras instâncias de Auroras.

5. Experimentos

Nesta seção são apresentados os experimentos realizados para avaliação do gerenciador de nuvem Aurora robusto. O primeiro experimento mede o tempo necessário para que uma nova instância do Aurora seja incorporada a um *cluster* de Auroras já existente. O segundo experimento mede a latência de replicação de dados, tanto para as informações do banco de dados quanto para as imagens, programas e métricas. A latência de replicação é medida tanto em um *datacenter* quanto em dois *datacenters*. No cenário com um *datacenter*, o experimento foi realizado com dois Auroras em um primeiro momento e três Auroras em seguida. No cenário com dois *datacenters*, há um Aurora em cada *datacenter*. Este cenário, com dois *datacenters*, simula um enlace remoto. O terceiro experimento mede o tempo de detecção e recuperação de falhas, e tem o intuito de analisar a disponibilidade

do sistema. Para todos os experimentos assume-se que o canal de comunicação entre os Auroras é confiável. Os Auroras utilizados são instalados em máquinas virtuais com Sistema Operacional Ubuntu 14.04.02 LTS, que possuem 512MB de memória RAM. Os canais de comunicação remotos foram simulados oscilando atrasos progressivos no canal.

O primeiro experimento realizado mede o tempo necessário para que uma nova instância do Aurora seja incorporada a um *cluster* de dois Auroras. Quando uma nova instância é inserida num *cluster*, ela deve se comunicar com outra instância e receber todos os dados relevantes. Os tempos deste experimento foram medidos no relógio local e estão descritos no gráfico 3. Existe um tempo padrão de entrada de uma nova instância no *cluster*, que é de 15 a 18 segundos. O tamanho do banco de dados foi aumentado gradativamente. Este tamanho é representado pelo número de máquinas virtuais configuradas no Aurora. Na figura, é possível observar que o número de máquinas virtuais influencia diretamente no tempo de entrada do terceiro Aurora no *cluster*. Com somente uma máquina virtual configurada, o tempo de entrada de uma instância foi de 20 segundos. Com o número de 5000 máquinas virtuais configuradas, o tempo necessário para que uma nova instância fosse incorporada no *cluster* foi de 40 segundos. Com esses números, é possível concluir que, para um aumento no número de máquinas virtuais de 5000 vezes, o tempo de entrada de um novo Aurora apenas dobrou.

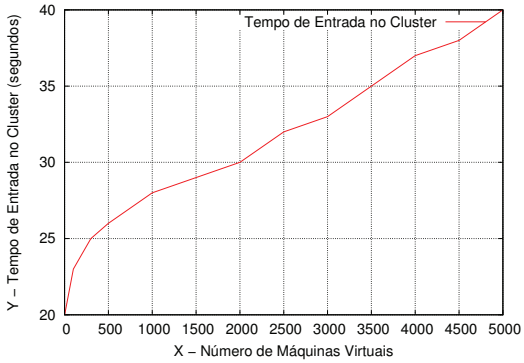


Figura 3. Gráfico de tempo de inserção de um novo Aurora.

O segundo experimento tem o propósito de medir a latência para replicação de dados e foi dividido em três partes. Na primeira parte, em um primeiro momento, é realizada a medição da latência de replicação dos dados mantidos pelo banco de dados MySQL, pela simples adição de configuração em um dos Auroras. Em sequência, é medida a latência de replicação das imagens das máquinas virtuais, após a inserção de uma imagem no Aurora. Na segunda parte, a latência de replicação foi medida com um aumento progressivo da carga de inserção de dados no Aurora. Na terceira parte, a latência de replicação foi medida pela adição de configuração em um dos Auroras, em um cenário com atrasos no canal de comunição entre as instâncias do Aurora, simulando um enlace remoto.

Na primeira parte do segundo experimento, a medição da latência foi realizada através dos *logs* do banco de dados. A imagem que foi inserida no Aurora possui um tamanho de 2,5MB. A medição da latência da replicação da imagem foi realizada com a utilização de relógios locais. Nesta primeira parte, o experimento foi executado em um cenário com dois gerenciadores Aurora em um primeiro momento, e três Auroras no

momento seguinte. O experimento foi executado em um único *datacenter*. No cenário com dois Auroras, o tempo de replicação dos dados mantidos no MySQL é de milésimos de segundos, com média de 21,63 ms, após 30 execuções. Na replicação das imagens, a latência média foi de 4,7 segundos, também para 30 execuções. No segundo cenário, com três Auroras, a replicação dos dados do MySQL também teve latência de milésimos de segundos, a média foi um pouco maior do que o cenário com dois Auroras, de 36,95 ms. A replicação das imagens teve tempo médio de 5,96 segundos. A tabela 1, resume o experimento.

Tabela 1. Experimentos de Latência de Replicação

Tipo de Replicação/Cenário	2 Auroras		3 Auroras	
	Média	Desvio Padrão	Média	Desvio Padrão
Replicação MySQL	21,63 ms	11,45 ms	36,95 ms	17,33 ms
Replicação Rsync	4,7 s	0,95 s	5,96 s	1,56 s

Na segunda parte do segundo experimento, a variação de latência foi medida com um aumento da carga da inserção de dados no banco de dados. O experimento também foi executado em ambientes com duas e três instâncias do Aurora. Além do tempo de replicação, a utilização de CPU e rede também foram medidas. O uso de memória é constante na máquina virtual. Na máquina com 512MB de memória, a utilização é de 95%. A inserção de dados foi executada através da configuração de novas máquinas virtuais através da importação de um arquivo *xml*.

A figura 4 ilustra o tempo de replicação em relação a um aumento progressivo de máquinas virtuais gerenciadas pelo Aurora. Existem duas linhas, uma representando o experimento com dois Auroras e outra representando o experimento com três Auroras. Nota-se que o tempo de replicação aumenta conforme aumenta-se a carga de inserção de máquinas virtuais e também quando existem mais gerenciadores Aurora no *cluster*, sendo exatamente este o comportamento esperado. Com a inserção de dados de 4000 máquinas virtuais, o tempo de replicação com um *cluster* de duas instâncias do Aurora foi de aproximadamente 750 ms e com um *cluster* de três instâncias foi de aproximadamente 1290 ms.

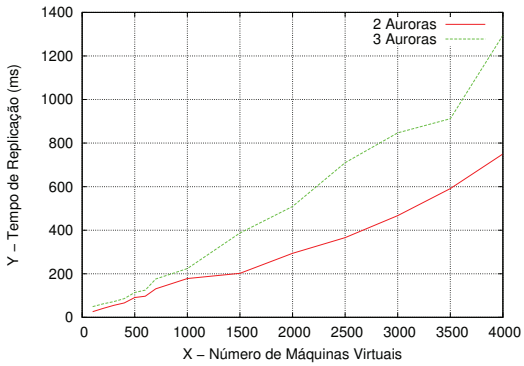
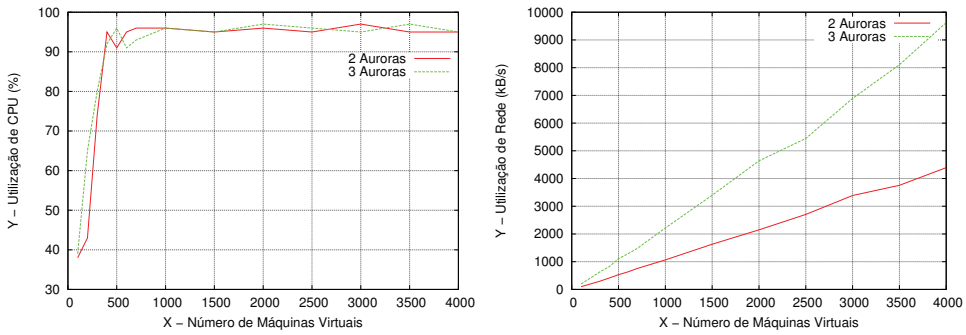


Figura 4. Gráfico de tempo de replicação conforme aumento do banco de dados.

As figuras 5(a) e 5(b), ilustram a utilização de CPU durante o processo de inserção de máquinas virtuais e a utilização da interface de rede, respectivamente. No gráfico de utilização de CPU é possível notar que a partir da inserção de 400 máquinas virtuais, a utilização de CPU se manteve constante, com mais de 92%, tanto para um *cluster* de duas instâncias do Aurora quanto para um *cluster* de três instâncias. No gráfico de utilização

da rede, conforme o número de inserções de máquinas virtuais aumenta, a utilização de rede também aumenta, o que é um comportamento normal, pois existem mais dados para serem replicados entre os membros do *cluster* de Auroras. A utilização de rede sempre foi pelo menos o dobro em um *cluster* com três instâncias do Aurora em relação a um *cluster* com duas intâncias. Isto acontece pois a replicação dos dados com três instâncias sempre será duplicada, uma vez que a instância que recebe os dados deve replicar os mesmos dados para as outras duas instâncias. Após a inserção de 4000 máquinas virtuais, a utilização de rede com três instâncias do Aurora foi de 9600 kB/s e com duas instâncias foi de 4360 kB/s.



(a) Utilização de CPU . (b) Utilização de Rede.

Figura 5. Avaliação de utilização de CPU e rede durante a replicação.

Na terceira parte do segundo experimento foi realizada a simulação de dois *datacenters* remotos, com um Aurora em cada um dos *datacenters*. Para simular a comunicação entre os dois *datacenters*, foram adicionados atrasos no enlace entre os dois Auroras, através do aplicativo *tc (Traffic Control)*. A inserção dos dados para replicação do banco de dados foi realizada através da simples adição de configuração no Aurora. Os tempos de replicação foram obtidos através dos *logs* do MySQL.

A figura 6 mostra a relação entre a latência de replicação e o atraso gradual no canal de comunicação entre dois Auroras. O principal objetivo deste experimento é conseguir verificar o limite do sistema para o caso de dois *datacenters* remotos. O limite do sistema acontece com um atraso de 4000 ms no enlace que conecta os dois *datacenters* simulados. Com esse atraso, a interface web do Aurora começa a apresentar lentidão e não consegue progredir com sua execução. Acontece este travamento, porque com a lentidão, o processo de replicação do Galera não consegue escrever a transação no banco de dados (não consegue obter o *lock*) e ocorre o *timeout*. Na figura, também é possível notar que a partir do atraso de 1000 ms, a latência de replicação começa a aumentar mais rapidamente, isto acontece, pois o tempo para o Galera adquirir o *lock* para escrever a transação no banco de dados é maior.

Foram também realizados testes, aumentando-se a quantidade de memória da máquina virtual do Aurora, para 1024MB. A ideia destes testes é verificar se, com um aumento de memória, o tempo de replicação diminui. Com esse aumento aconteceu uma pequena melhora nos tempos de replicação do experimento e não ocorreu o travamento do sistema com 4000 ms, mas com um atraso de 5000 ms.

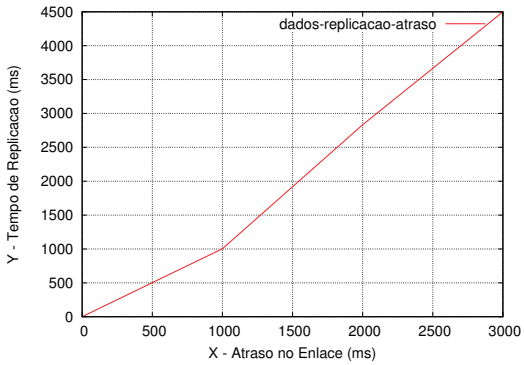


Figura 6. Gráfico de tempo de replicação com atraso no enlace entre dois *data-centers*.

A tabela 2 mostra o tempo médio de acesso a uma instância Aurora, com o aumento progressivo de atraso no enlace. Foram executadas 50 capturas do tempo de RTT (*Round-Trip Time*) para verificar o tempo de acesso e são mostradas as médias obtidas. A solução proposta funciona corretamente para atrasos de até 3 segundos, demonstrando sua viabilidade em redes reais.

Tabela 2. Experimentos de Tempo de Acesso/ Tempo de Replicação - Com atraso no Enlace

Atraso (ms)	Latência de Acesso (ms)	Latência de Replicação (ms)
0	0.485	0.521
10	10.834	10.914
50	50.753	50.824
100	100.814	101.894
500	500.768	502.681
1000	1000.883	1002.478
2000	2000.740	2832.474
3000	3000.905	4498.797

O terceiro experimento tem o intuito de analisar a disponibilidade do sistema. O tempo de detecção de falha e o tempo de recuperação após uma falha são medidos neste experimento. O MTBF (*Mean Time Between Failures*), ou período médio entre falhas, é usado para calcular a disponibilidade. O tempo de detecção de falha é o tempo configurado no *HAProxy*. Na configuração utilizada neste trabalho, a verificação de falha no Aurora acontece a cada 2000 ms e, se acontecem dois resultados de falha, o serviço é considerado falho. Assim, o tempo de detecção é de 4000ms (4s). A quantidade de resultados falhos para considerar que um servidor está falho, além dos intervalos de verificação, são configuráveis. Se o serviço é considerado falho, então qualquer nova tentativa de acesso ao Aurora, é realizada somente no Aurora que está ativo.

Existe o tempo de detecção do próprio Galera. Se um processo do MySQL em uma máquina do *cluster* sofre falha, a detecção ocorre em milésimos de segundos. Analisando-se *logs* obtidos durante a execução dos experimentos, a detecção ocorre em até 15 ms. No caso em que ocorre uma falha de comunicação com outro Aurora, o Galera considera o nó como suspeito em 5 segundos, mas somente o retira do *cluster*, se não obtiver resposta no tempo de 15 segundos. Vale ressaltar que estes parâmetros são configuráveis.

No terceiro experimento também foi medido o tempo de recuperação de um Au-

rora após uma falha. A falha provocada é a parada do serviço MySQL e a recuperação é o retorno deste serviço. Os experimentos foram realizados com duas e três instâncias de Auroras. O tempo de inicialização do serviço no cenário com dois Auroras foi de 16 segundos. No cenário com três Auroras, o tempo de inicialização foi de 18 segundos. Os tempos deste experimento também foram medidos através de um relógio local.

Na tabela 3, são mostrados os valores de disponibilidade do sistema, calculados a partir do MTBF. No cálculo do MTBF é considerado que o período de indisponibilidade do sistema é o intervalo de monitoramento do *HAProxy*. Nos cálculos, o tempo do intervalo de monitoramento é representado por im e a quantidade de verificações para considerar que um servidor está falho vf. Outra medida representada é o tempo de MTBF $tmtbf$, que representa o tempo em que são consideradas as falhas, por exemplo, o $tmtbf$ sendo 30 minutos significa que considera-se o tempo de 30 minutos entre duas falhas consecutivas. A quantidade de vf e $tmtbf$ foram variadas nos cálculos. O cálculo usado para a disponibilidade é o seguinte:

$$disponibilidade = \frac{(tmtbf(segundos) - (im * vf)) * 100)}{tmtbf}$$

Tabela 3. Tempo de disponibilidade.

Tempo MTBF	Intervalo de Monitoramento (im)	Quantidade de Verificações (vf)	Disponibilidade (%)
1 hora	2 segundos	1	99,94
		2	99,88
		5	99,72
2 horas	2 segundos	1	99,97
		2	99,94
		5	99,86
4 horas	2 segundos	1	99,98
		2	99,97
		5	99,93

Nota-se na tabela, que apesar da disponibilidade ser sempre menor quando o parâmetro vf é igual a 5, a precisão da detecção é maior, com probabilidade menor de gerar um falso-positivo. Um exemplo é: se acontecer uma falha em uma instância do Aurora e ele permanecer indisponível por 5 segundos e logo retornar, se o vf conter valor de 5, a instância não será considerada falha. No entanto, se o vf for igual a 2, a instância será considerada falha, gerando um falso-positivo. Este experimento demonstra que os parâmetros de detecção de falhas devem ser bem analisados, tanto o intervalo de monitoramento, quanto quantidade de resultados falhos para considerar que um servidor está falho. A análise deve ser tomada, considerando o aspecto de disponibilidade, e/ou precisão na detecção de uma falha.

6. Conclusão

Este trabalho propôs a implementação de uma plataforma robusta para gerência de recursos em nuvens IaaS (*Infrastructure as a Service*). A solução foi implementada no gerenciador de nuvens *Aurora Cloud Manager*. A estratégia é baseada na replicação do Aurora, mais especificamente o próprio gerenciador da nuvem. A principal contribuição deste trabalho é oferecer, no próprio gerenciador de nuvem, a *feature* de alta disponibilidade do próprio gerenciador, com a opção de ativação da replicação em sua interface e monitoramento das outras instâncias replicadas.

O desempenho e a robustez da plataforma foram avaliados experimentalmente. Foram realizados experimentos para a medição do tempo necessário para que uma nova

instância do Aurora seja incorporada a um *cluster* de Auroras já existente; medição da latência de replicação de dados, tanto para as informações do banco de dados quanto para as imagens, programas e métricas. A latência de replicação foi medida tanto em um *datacenter* quanto em dois *datacenters* remotos. E, por último, a medição do tempo de detecção e recuperação de falhas, com o intuito de analisar a disponibilidade do sistema. Os resultados mostraram que o gerenciador se tornou robusto e capaz de ser executado com múltiplas instâncias replicadas.

Entre os trabalhos futuros, pretende-se implementar o cenário com múltiplos *datacenters* remotos utilizando federações de nuvens para o compartilhamento de recursos.

Referências

Arean, O. (2013). Disaster Recovery in The Cloud . *Network Security*, 2013(9):5 – 7.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A View of Cloud Computing. *Communications of The ACM*, 53(4):50–58.

Barbosa de Carvalho, M., Pereira Esteves, R., da Cunha Rodrigues, G., Zambenedetti Granville, L., and Rockenbach Tarouco, L. (2013). A Cloud Monitoring Framework for Self-Configured Monitoring Slices Based on Multiple Tools. In *9th International Conference on Network and Service Management (CNSM)*.

Charron-Bost, B., Pedone, F., and Schiper, A., editors (2010). *Replication: Theory and Practice*. Springer-Verlag, Berlin, Heidelberg.

Jhawar, R. and Piuri, V. (2012). Fault Tolerance Management in IaaS Clouds. In *2012 IEEE First AESS European Conference on Satellite Telecommunications (ESTEL)*, pages 1–6.

Kholghi, M. A. K., Abdullah, A., Latip, R., Subramaniam, S., and Othman, M. (2014). Disaster Recovery in Cloud Computing: A Survey. *Computer and Information Science*, 7(4):39–54.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing.

Tridgell, A. (1999). *Efficient Algorithms for Sorting and Synchronization*. PhD thesis, Australian National University Canberra.

Tsakalozos, K., Roussopoulos, M., and Delis, A. (2011). VM Placement in non-Homogeneous IaaS-Clouds. In Kappel, G., Maamar, Z., and Motahari-Nezhad, H., editors, *Service-Oriented Computing*, volume 7084 of *Lecture Notes in Computer Science*, pages 172–187. Springer Berlin Heidelberg.

Wickboldt, J. A., Esteves, R. P., de Carvalho, M. B., and Granville, L. Z. (2014). Resource Management in IaaS Cloud Platforms Made Flexible Through Programmability . *Computer Networks*, 68(0):54 – 70. Communications and Networking in the Cloud.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud Computing: State-of-The-Art and Research Challenges. *Journal of Internet Services and Applications*, 1(1):7–18.