SDNCloud: plataforma EAD com suporte à computação em nuvem e OpenFlow

Igor M. Ávila[1,2], Rhodney Simões[1], Kelvin L. Dias[1]

[1]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Av. Jornalista Aníbal Fernandes, s/n – 50.740-560 – Recife – PE – Brasil

[2]Instituto Federal do Sudeste de Minas Gerais (IFSudesteMG) – Muriaé – MG – Brasil

igor.avila@ifsudestemg.edu.br, {rambks2,kld}@cin.ufpe.br

Abstract. *The growing demand for distance learning (EAD) offering by Brazilian institutions requires a flexible and low cost solution, which is difficult to achieve with comercial / closed solutions. In addition, there are no resources for acquiring hardwares. This paper proposes an architecture tailored to EAD which integrates SDN to cloud computing (SDNCloud) and reuses hardware available at the institution. SDNCloud is parameterized and dimensioned through real traffic caracterization / modeling. Thus, a traffic model based on a real environment from EAD was developed. Performance results show the importance in the appropriate choice of metrics and thresholds and the benefits of SDNCloud compared to the traditional model.*

Resumo. *O crescimento da demanda de ensino a distância (EAD) ofertado por instituições brasileiras requer uma solução flexível e de baixo custo, o que dificulta a obtenção de soluções comerciais/fechadas. Além disso, não há recursos para aquisição de hardwares. Este artigo propõe uma arquitetura adaptada para o EAD que integra SDN à computação em nuvem (SDNCloud) e reusa o hardware disponível na instituição. SDNCloud é parametrizado e dimensionado através de modelagem/caracterização de tráfego real. Então, um modelo de tráfego baseado em um ambiente real de EAD foi desenvolvido. A análise dos resultados mostra a importância da escolha apropriada das métricas e limiares e os benefícios de SDNCloud comparado ao modelo tradicional.*

1. Introdução

Computação em nuvem (CN) surgiu da necessidade de prover recursos computacionais (memória, processamento, largura de banda de rede, etc) como forma de serviço NIST (*National Institute of Standards and Technology*)[Mell and Grance]. Assim, o usuário paga apenas pelo que utilizou, seguindo o modelo *pay-as-you-go* [Armbrust et al. 2010]. Dessa forma, não há a necessidade de aquisição de uma infraestrutura complexa, de alto custo e de difícil gerenciamento. Neste sentido, a utilização de elasticidade [Mell and Grance] em CN surge como requisito para otimizar o uso dos recursos existentes, além da reciclagem do parque tecnológico [Yazhou Hu et al. 2016], buscando o provimento de um ambiente eficiente sob demanda, de forma automática e orquestrada.

Em instituições federais de ensino, onde há sérias limitações orçamentárias e infraestrutura com poucos recursos computacionais, a CN surge como uma solução para

a hospedagem de Ambientes Virtuais de Aprendizagem (AVA). A elasticidade sob demanda e o balanceamento de carga permitem o aproveitamento do parque tecnológico, reutilizando de maneira distribuída os equipamentos descontinuados e com poucos recursos computacionais. Cabe destacar ainda que, além da escassez de recursos financeiro e computacional, a educação a distância (EAD), tem como previsão, uma taxa de crescimento anual composta de cerca de 11% até 2020 [Technavio 2016].

Redes definidas por software (SDN) [Nunes et al. 2014] surgem para alavancar a programabilidade de redes e o gerenciamento de conectividade para serviços em nuvem devido ao desacoplamento do plano de dados e do plano de controle, além de prover visão global da rede. A integração de SDN com CN, proporciona no nível de redes a flexibilidade necessária para este ambiente no contexto de CN. [Kreutz et al. 2015] Diante desta versatilidade, torna-se possível a inclusão de novas tecnologias e protocolos, fazendo com que seja possível a implementação do conceito de programabilidade rede, melhorias nas decisões dos encaminhamentos dos fluxos e diminuição no *overhead* associado ao gerenciamento dos switches virtuais.

Este artigo apresenta SDNCloud, sua arquitetura contempla a integração de um controlador SDN a um ambiente de CN e metodologia para realizar balanceamento de carga e elasticidade sob demanda de uma plataforma EAD. Esta investigação foi realizada comparando-se dois cenários distintos: um ambiente de CN tradicional; e, o outro desacoplando os planos de dados e controle, fazendo a inclusão de um controlador SDN.

As contribuições deste artigo são: implementação de ambiente experimental com tráfego realista e obtenção de resultados que demonstram que SDNCloud apresenta menor taxa de perda de requisição em comparação ao ambiente tradicional de CN; identificação da métrica mais apropriada para a elasticidade e dos parâmetros que mais influenciam o desempenho da métrica escolhida; caracterização de tráfego de um AVA; e, mostrar que SDNCloud é capaz de substituir infraestruturas sem virtualização e com mais recursos.

Na sequência, a Seção 2 trata dos trabalhos relacionados. A Seção 3 descreve a arquitetura do ambiente de experimentação e seu funcionamento. A Seção 4 apresenta a metodologia utilizada. A seção 5 expõe os resultados obtidos com os experimentos e discute questões relevantes. Por fim, a Seção 6 destaca as conclusões deste artigo.

2. Trabalhos Relacionados

[d. R. Righi et al. 2016] utilizaram OpenNebula como controlador de nuvens privadas para realizar o monitoramento e a elasticidade de máquinas virtuais executando uma aplicação em JAVA para *High-performance computing* (HPC). A utilização da CPU foi usada como métrica para elasticidade, enquanto tempo, em segundos, para executar a aplicação foi utilizada como métrica para medir a qualidade da elasticidade. [Kang and Lee 2016] aplicaram OpenStack *three-node*, empregando controlador, rede e computação. A utilização da CPU foi a métrica escolhida para elasticidade, enquanto tempo de resposta foi a métrica definida para medir a qualidade da elasticidade. Nos experimentos utilizou-se a ferramenta Apache JMeter[1] para simular as cargas de trabalho.

[Simões 2013] fez um estudo comparando elasticidade entre nuvens públicas, privadas e híbridas, utilizando-as para hospedar um serviço de Smart Grid. A utilização

[1] http://jmeter.apache.org/

da CPU não se mostrou um parâmetro adequado para a realização da elasticidade das instâncias virtuais, pois, em alguns casos, as instâncias eram criadas e excluídas desnecessariamente. [Bao et al. 2014] implementa elasticidade para nuvem privada com OpenStack. Desenvolveu-se um serviço web para testar a plataforma e um balanceador de carga foi utilizado para redirecionar as requisições entre as instâncias virtuais. Foi selecionada uma parte das solicitações de um *Content Delivery Network* (CDN) e ampliado esse número para simular a mudança na carga de trabalho do serviço web.

[Poddar et al. 2015] realizam elasticidade e balanceamento de carga utilizando os controladores de nuvens OpenStack e SDN OpenDayLight[2]. Empregam os módulos de telemetria e o serviço de balanceamento de carga do OpenStack em conjunto com a aplicação desenvolvida sobre controlador SDN. Utilização de CPU e memória são as métricas que podem ser utilizadas para orquestração da elasticidade. [He et al. 2016] apresentam uma plataforma aberta, baseada em OpenStack, que combina recursos de rede tradicionais de CN com SDN. Utilizam orquestração para criação automática VMs, Ryu[3] como controlador SDN e OpenVirteX[4] como plataforma de virtualização. Os experimentos deste trabalho utilizam geração de tráfego sintético para comparar seus resultados com o do mesmo ambiente emulado via Mininet[5].

Diante dos trabalhos apresentados, conclui-se que ainda existem questões a serem avaliadas e outras praticamente comum a muitos trabalhos. [d. R. Righi et al. 2016] e [Kang and Lee 2016] abordam apenas a questão da elasticidade em CN. [Simões 2013] e [Bao et al. 2014] apresentam trabalhos que realizam balanceamento de cargas de serviços web por intermédio de controlador de nuvem e com objetivo de auxiliar na elasticidade, mas nenhuma deles faz a integração de seus ambientes com controlador SDN. Apesar de [Bao et al. 2014] ter selecionado parte dos acessos de um CDN, nenhum dos trabalhos citados aplicou tráfego realista. [Poddar et al. 2015] e [He et al. 2016] realizam tanto as medições quanto o balanceamento por aplicação desenvolvida sobre controlador, enquanto nossa solução utiliza CN para realização destas ações. [He et al. 2016] utiliza balanceamento de carga e elasticidade em CN, integrada com SDN, validando seu ambiente através de emulação com Mininet. Este trabalho de pesquisa com SDNCloud utiliza as mesmas técnicas para um serviço web com tráfego realista e mostra que os parâmetros da elasticidade estão diretamente relacionados à carga de trabalho aplicada.

3. SDNCloud

Na Figura 1 observa-se uma visão geral sobre a arquitetura de funcionamento de SDNCloud. Nela está incluída a virtualização presente no *data center* (DC), bem como as VMs utilizadas pelo controlador de nuvens e pelo plano de controle (SDN). Destacando-se o módulo de computação com as VMs que hospedam a plataforma EaD, balanceador de carga e a comunicação com os usuários. Foi utilizado o controlador de nuvens OpenStack, versão Kilo e o controlador SDN OpenDayLight. A escolha pelo Kilo apoia-se sobre sua arquitetura com switch virtual *open source*, permitindo acoplamento com ODL através de *features* e biblioteca, todas livres. Versões recentes do ODL utilizam *Linux Bridge* com agentes proprietários em sua arquitetura, o que dificulta sua integração com OpenFlow

[2]https://www.opendaylight.org/
[3]https://osrg.github.io/ryu/
[4]ovx.onlab.us
[5]mininet.org

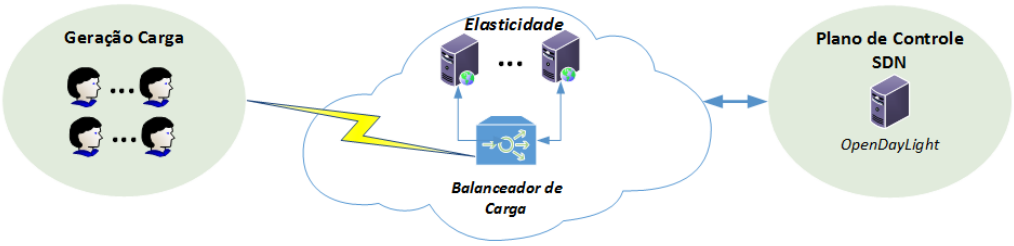

Figura 1. Arquitetura SDNCloud

(OF). ODL apresentou maior acoplamento do que outros controladores SDN testados nos experimentos preliminares, justificando assim sua escolha. Para a comunicação entre o plano de dados e de controle foi empregado o protocolo OF, versão 1.3.

A arquitetura de SDNCloud foi dividida em dois segmentos: inclusões modulares e de serviço (realizadas no ambiente de CN tradicional) e utilização de SDN e sua comunicação com CN. Na solução foram incluídos os módulos de orquestração,[6] telemetria[7] e o serviço LBaaS no nó controlador do OpenStack. Por parte do controlador ODL, a comunicação é realizada através da inclusão de recursos específicos para a comunicação com OpenStack, além de configurar os switches virtuais (OpenVSwitch[8]) dos nós de rede e computação para serem gerenciados pelo controlador ODL, e não mais pelo Neutron.

4. Metodologia

O ambiente de experimentação foi implementado utilizando um host físico e virtualizado através de *Virtual Machine Manager* (VMM). Foram criadas 08 VMs, todas utilizando Ubuntu Server 14.04. O ambiente de experimentação opera com um DC virtualizado com plataforma de gerenciamento de nuvens, permitindo que o AVA Moodle[9] possa provisionar recursos sob demanda, balanceando-os entre VMs que fazem parte do *pool* de recursos. Os experimentos contam com duas configurações diferentes. A primeira utiliza todas as funcionalidades nativas de uma nuvem tradicional, enquanto a segunda utiliza SDNCloud. A única diferença infraestrutural entre o ambiente tradicional de CN e SDNCloud é o uso da VM com o controlador ODL.

4.1. Cenário

O AVA é o sistema de informação que operacionaliza o funcionamento da EaD [Franco et al. 2003]. Moodle é o AVA utilizado como base para o trabalho. Os principais atores responsáveis pela troca de dados no processo de ensino e aprendizagem são professores, tutores, coordenadores de curso e alunos. Cada um possui um perfil bem definido, o que impacta diretamente nas suas atividades no Moodle e consequentemente no perfil do tráfego de dados.

Diante destes perfis de utilização do Moodle, tornou-se possível identificar quatro padrões de tráfego de dados, entre eles: *upload* de arquivos, *upload* de dados, *download* de arquivos e *download* de dados. *Uploads* de arquivos são limitados, via aplicação, em

[6]https://wiki.openstack.org/wiki/Heat
[7]https://wiki.openstack.org/wiki/Telemetry
[8]openvswitch.org
[9]https://moodle.org

no máximo 4 MB. Em consequência disso, os *downloads* de arquivos também só podem ser de no máximo 4 MB.

O padrão de tráfego de dados foi identificado através de análise dos logs do servidor web da aplicação e do banco do Moodle. A consulta utilizou os logs de um ano do AVA. Os padrões *download* de arquivos, *upload* de dados e *download* de dados foram identificados através dos logs do serviço web, enquanto *upload* de arquivos através de consultas ao banco de dados.

4.2. Caracterização de tráfego

De acordo com os testes de aderência (Tabela 1), utilizando a ferramenta R, a distribuição que melhor representa os intervalos entre as requisições foi generalização de pareto, tal distribuição foi identificada em [Lee and Gupta 2007]. Log normal é a distribuição que representa os tamanhos de arquivos que tiveram como ação *download* para o Moodle e os arquivos que tiveram como ação *upload* por parte dos usuários.

Tabela 1. Resultado teste de aderência

Métrica	Distribuição	Parâmetro		
Intervalo entre as requisições	Generalização de Pareto	$\xi = 0,73227$	$\mu = -0,66271$	$\beta = 2,485$
Arquivos para download	Log normal	$\mu = 1,2673684932$ $\sigma = 1,1899134559$		
Arquivo para upload	Log normal	$\mu = 3,73603006$ $\sigma = 1,56271747$		

Com a caracterização do tráfego foram identificados o intervalo entre as requisições, os tamanhos dos arquivos de *download* e *upload*, faltando caracterizar a frequência com que ocorrem *download* e *upload*s. Além disso, foi realizado pré-processamento e *outliers* foram identificados e removidos. Com o tratamento de dados foi possível identificar que 73% das requisições são de *download* e 27% de *upload*.

4.3. Geração de Carga

Foi criada de uma aplicação para geração de carga que executa um script em R para gerar número aleatório segundo a distribuição generalização de pareto, sendo este o intervalo entre as requisições. Gera um número aleatório, utilizando a distribuição log normal. Este número pode variar de 0 a 99. Se este número for menor que 73, será realizado *download*, se maior ou igual, *upload* deverá ser executado.

Para a geração de carga de trabalho foram utilizadas duas VMs que representam cada uma delas 100 usuários, solicitando cada um deles 100 requisições. A cada 4 segundos um novo usuário é incluído em cada módulo de geração de carga, perfazendo um total de 100 em cada uma, 200 no total. Como o intervalo das requisições é definido através de um números aleatórios, a geração de carga termina com 3600 segundos. Os valores para geração de carga foram definidos através de experimentos preliminares.

4.4. Métricas, fatores e níveis

Foram realizados experimentos preliminares medindo as métricas de utilização tanto de CPU quanto de memória e da quantidade de *bytes* de entrada e de saída. Tais métricas

foram usadas como base para a realização da elasticidade e cada um dos experimentos foram executados 30 vezes. Tanto a análise dos resultados, quanto os gráficos foram obtidos com o cálculo da média no intervalo de 30 segundos dos tempos de resposta (TR) de cada uma das iterações e posteriormente com o cálculo da média das 30 iterações. Sendo assim possível a determinação de qual métrica seria mais adequada para o emprego no ambiente proposto.

Elasticidade positiva (*scale-up*) é a ação em que o sistema expande sua capacidade horizontalmente, instanciando uma nova VM, enquanto a negativa (*scale-down*) tem como ação diminuir sua capacidade, excluindo VMs. Com base nos experimentos preliminares, foram definidos três fatores para elasticidade positiva (50%, 70% e 90%), três para elasticidade negativa (10%, 20% e 30%) e três intervalos de medição (240, 480 e 960 segundos), ou seja, quanto tempo o limiar deve permanecer verdadeiro para que a ação de elasticidade seja executada. Com os fatores e níveis escolhidos, foram executados 27 experimentos com ambiente tradicional de CN e a mesma quantidade com SDNCloud, totalizando 54 experimentos.

5. Resultados

Para demonstrar os resultados foram selecionados três experimentos que são classificados como: melhor, médio e pior, variando fatores e níveis para elasticidade positiva, negativa e intervalo entre elasticidades. Os experimentos são classificados como XX-YY-ZZZ, onde XX é o valor da utilização da CPU para elasticidade positiva, YY, o valor da utilização da CPU para elasticidade negativa e ZZZ, o intervalo entre as elasticidades. Adotou-se para todos os gráficos o intervalo de confiança de 99%.

Como primeiro resultado, foi identificado que o TR não se mostrou uma métrica eficiente para medir a qualidade da elasticidade. Praticamente todos os experimentos possuem TR médio abaixo de 1 segundo, não sendo possível classificar qual experimento teve melhor ou pior desempenho quanto a elasticidade. Na Figura 2 nota-se o TR médio dos melhores experimentos tanto em um ambiente tradicional de Cloud quanto utilizando a arquitetura SDNCloud. Tendo em vista que as barras do intervalo de confiança se sobrepõem não é possível afirmar que um resultado é melhor que outro.

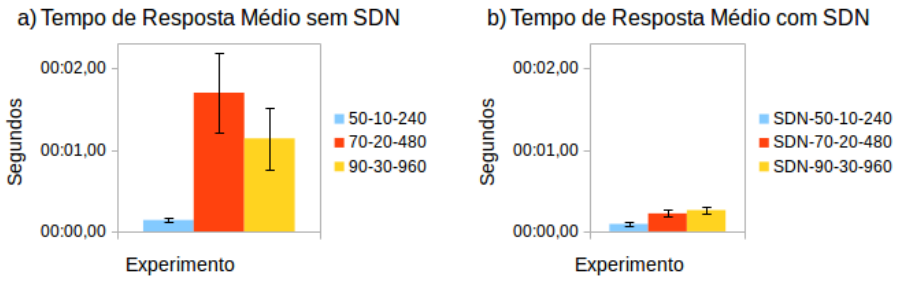

Figura 2. Tempo de resposta médio por experimento.

Na Figura 3 observa-se as taxas de perdas dos experimentos sem SDNCloud. Nela são mostradas as taxas de perda de requisições comparando os melhores, os médios e o piores casos para a variação de fatores e níveis para elasticidade positiva, negativa, intervalo entre as elasticidades e para todos os experimentos selecionados. Diante dos gráficos,

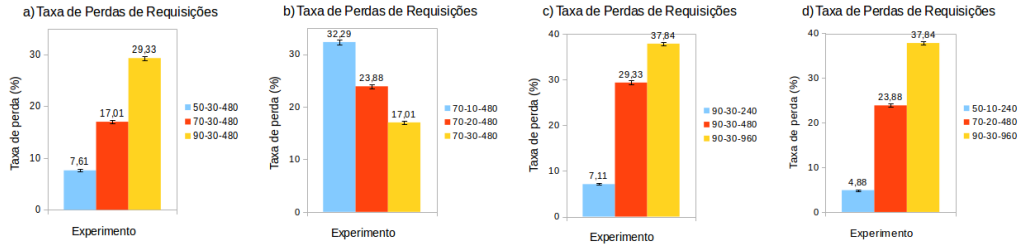

Figura 3. Taxa de perda de requisição sem SDNCloud.

conclui-se que para elasticidade positiva, os experimentos com XX=50 (Figura 3a) tiveram uma taxa de perda de dados 55,26% menor do que os experimentos com XX=70 e 74,05% menor que XX=90. Para elasticidade negativa (Figura 3b) os parâmetros YY=30 obtiveram melhores resultados quanto a taxa de perda de dados, sendo 47,32% menor que YY=10 e 28,76% menor que YY=20. O intervalo entre as requisições (Figura 3c) mostrou ser, nos experimentos executados, o parâmetro que mais influencia na taxa de perda da elasticidade, o parâmetro ZZZ=240 teve taxa de perda 75,75% melhor do que ZZZ=480 e 81,21% menor que ZZZ=960. Identificado que a métrica taxa de perda se mostrou útil para medir a qualidade da elasticidade.

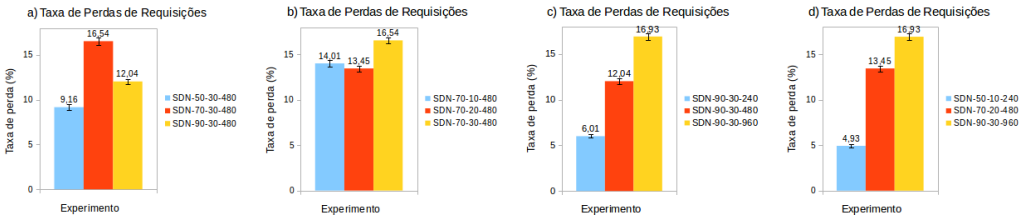

Figura 4. Taxa de perda de requisição com SDNCloud

Os resultados constatados na Figura 4 utilizam os mesmos valores para as métricas avaliadas da Figura 3. No entanto, o ambiente empregado passa a ser o de SDNCloud. Comparando os resultados apresentados nos gráficos dos experimentos com ambiente tradicional e SDNCloud, é possível perceber que, em 10 dos 12 casos, SDNCloud se mostrou mais eficiente, ou seja, apresentou uma menor taxa de perda de dados. Nota-se, ainda, que a métrica de taxa de perda continua sendo eficiente e que o intervalo entre as requisições é o parâmetro de elasticidade que mais influencia na taxa de perda de requisições, sendo seguido pelo parâmetro da elasticidade positiva. A combinação entre os menores valores de ZZZ e XX obtiveram a menor taxa de perda de requisições.

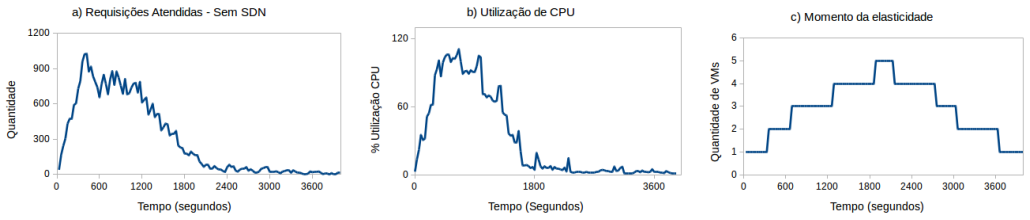

Figura 5. Resultado do experimento 50-10-240

Na Figura 5 é possível perceber que a quantidade de requisições atendidas mantém-se alta para qualquer carga de trabalho. Com as elasticidades, mesmo diante da grande quantidade de carga de trabalho, a utilização da CPU não ultrapassa por muito tempo 100% de utilização, valor que degrada o desempenho do sistema. Foram realizadas 4 elasticidades, fazendo com que 5 VMs estejam disponíveis para atender às requisições dos usuários.

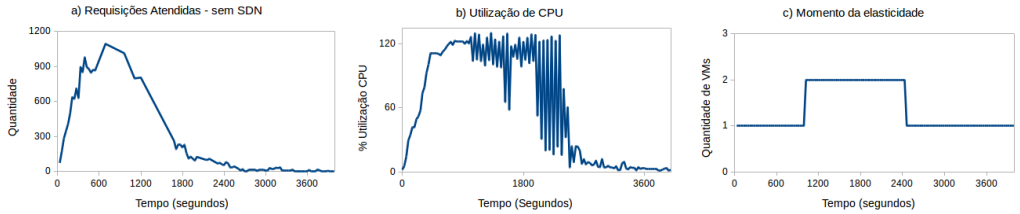

Figura 6. Resultado do experimento 90-30-960

Em contrapartida ao melhor experimento sem SDNCloud, na Figura 6 verifica-se o pior resultado sem SDNCloud. No primeiro gráfico, é possível perceber que no momento com maior carga de trabalho, há poucos pontos no gráfico, mostrando que o atraso na elasticidade ocasionou uma alta taxa de perda nas requisições. O gráfico de utilização de CPU deixa claro momentos em que o sistema não consegue atender às requisições. Quando o sistema está saturado, ele para de responder, abaixando a utilização da CPU. Após alguns instantes, ele volta a responder novamente às requisições, elevando a utilização do referido recurso. O gráfico com o momento da elasticidade demonstra que experimento com ZZZ=960 executou apenas uma elasticidade, não sendo suficiente para atender todas as requisições dos usuários no momento de maior carga de trabalho.

Na Figura 7 nota-se o melhor resultado com SDNCloud. Tanto a quantidade de requisições quanto a utilização de CPU se mantêm semelhantes ao ambiente sem SDNCloud, apresentando praticamente a mesma taxa de perda de requisições. Foram realizadas 6 elasticidades, alocando 20% mais recursos em VMs que o ambiente sem SDNCloud. Como os intervalos entre as requisições são gerados de forma aleatória, a quantidade de carga também é aleatória, influenciando diretamente na quantidade de elasticidades.

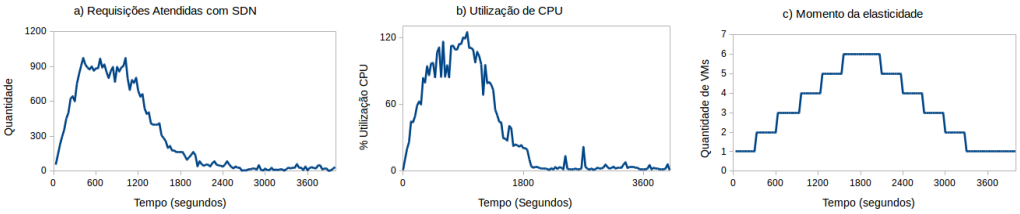

Figura 7. Resultado do experimento SDNCloud 50-10-240

Na Figura 8 observa-se os resultados do pior experimento com SDNCloud. Os vários pontos no gráfico da Figura 8a em relação à Figura 6a demonstraram que, no pior cenário, SDNCloud atendeu mais requisições na média, a cada 30 segundos, do que o ambiente sem SDNCloud, além de apresentar menor variação na utilização da CPU (Figuras 8b e 6b). Como a taxa de perda de requisições foi menor com SDNCloud, foi possível

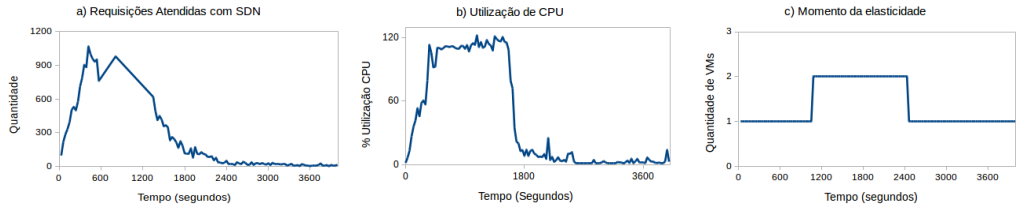

Figura 8. Resultado do experimento SDNCloud 90-30-960

perceber uma menor variação desta métrica. O experimento com SDNCloud, assim como no ambiente sem SDNCloud, também realizou apenas uma elasticidade.

A arquitetura SDNCloud propiciou menores taxas de perdas de requisições em comparação aos mesmos experimentos sem SDNCloud. Esses ganhos se justificam devido à configuração dos switches virtuais que deixaram de operar em modo reativo, na versão 1.0 do OF [Consortium et al. 2009], tornando-se proativos. A integração com o controlador ODL torna o OVS proativo, pois o referido controlador trabalha com a versão 1.3 do OF, ou seja, as decisões são tomadas antes das solicitações [Foundation 2014]. Como o OVS está sendo gerenciado pelo controlador SDN em SDNCloud, parte da gerência deixa de ser feita pelo OVS e passa para o controlador SDN, liberando o OVS para realizar apenas *matches* e encaminhamentos.

Analisando o uso de recursos, ficou claro que a quantidade de carga aplicada é o fator que mais impacta na definição dos parâmetros de elasticidade. A VM que responde as requisições web começa a ter seu serviço prejudicado, ou seja, necessitando replicação, quando a métrica para elasticidade (utilização de CPU) ultrapassa 100% de utilização. Como as medições eram feitas a cada 30 segundos, o processo de elasticidade como um todo, gasta em torno de 90 segundos. Sendo este o período necessário para criação de uma nova VM, atribuição de endereço IP e inclusão da mesma no balanceador de carga. Com base nesses valores médios, foram definidos os fatores e níveis. Esses valores definem o momento ideal para elasticidade. Se for feita com muita antecedência, há a utilização desnecessária de recursos e se atrasar, há comprometimento na qualidade do serviço.

6. Conclusão

A métrica de utilização da CPU se mostrou a mais útil na arquitetura com e sem SDNCloud e nos cenários avaliados. O TR médio não se mostrou uma métrica eficiente para medir a qualidade da elasticidade para a arquitetura e no cenário avaliado, sendo a taxa de perdas de requisições uma métrica mais eficiente. O desempenho da elasticidade está diretamente relacionado com a escolha dos parâmetros para elasticidade. A escolha dos parâmetros de intervalo entre as requisições e elasticidade positiva definem a qualidade da elasticidade, a qual encontra-se diretamente relacionada com a carga de trabalho que se deseja atender. A aplicação hospedada poderá ter seu desempenho prejudicado, caso a elasticidade atrase, ou utilizar recursos computacionais desnecessários, caso adiante. LBaaS se mostrou um serviço satisfatório para o balanceamento de cargas de um serviço web nos ambientes avaliados. Foi identificado que o serviço deixa de responder as requisições no momento que em que são incluídos ou removidos membros do balanceador. O módulo de orquestração se mostrou eficiente para o processo de elasticidade, inclusão e exclusão de membros do balanceador de cargas, criação de VMs e redes.

Referências

Armbrust, M., Stoica, I., Zaharia, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., and Rabkin, A. (2010). A view of cloud computing. *Communications of the ACM*, 53(4):50.

Bao, J., Lu, Z., Wu, J., Zhang, S., and Zhong, Y. (2014). Implementing a novel load-aware auto scale scheme for private cloud resource management platform. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–4.

Consortium, O. S. et al. (2009). Openflow switch specification version 1.0.0.

d. R. Righi, R., Rodrigues, V. F., da Costa, C. A., Galante, G., de Bona, L. C. E., and Ferreto, T. (2016). Autoelastic: Automatic resource elasticity for high performance applications in the cloud. *IEEE Transactions on Cloud Computing*, 4(1):6–19.

Foundation, O. N. (2014). Openflow switch specification version 1.3.4.

Franco, M. A., Cordeiro, L. M., and Castillo, R. a. F. D. (2003). O ambiente virtual de aprendizagem e sua incorporação na Unicamp. *Educação e Pesquisa*, 29(2):341–353.

He, Y., Li, W., Zhang, L., Wang, J., and Qi, Q. (2016). ECCN: An Elastic Customized Cloud Network Platform. In *2016 International Conference on Networking and Network Applications (NaNA)*, pages 429–432. IEEE.

Kang, S. and Lee, K. (2016). Auto-Scaling of Geo-Based Image Processing in an OpenStack Cloud Computing Environment. *Remote Sensing*, 8(8).

Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-Defined Networking : A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14 – 76.

Lee, J. J. and Gupta, M. (2007). A new traffic model for current user web browsing behavior. *Intel corporation*.

Mell, P. and Grance, T. The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology. *National Institute of Standards and Technology, Information Technology Laboratory*, page 7.

Nunes, B. A. A., Mendonca, M., Nguyen, X. N., Obraczka, K., and Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys and Tutorials*, 16(3):1617–1634.

Poddar, R., Vishnoi, A., and Mann, V. (2015). Haven: Holistic load balancing and auto scaling in the cloud. In *2015 7th International Conference on Communication Systems and Networks (COMSNETS)*, pages 1–8. IEEE.

Simões, R. A. M. B. K. (2013). *Gerenciamento de Elasticidade em Computação Em Nuvem : Estudo De Caso Usando Smart Grid*. PhD thesis.

Technavio (2016). Global Corporate E-learning Market 2016-2020. `htpp://www.technavio.com`. [Online; accessed 14-Dec-2016].

Yazhou Hu, Bo Deng, Fuyang Peng, Bin Hong, Yuchao Zhang, and Dongxia Wang (2016). A survey on evaluating elasticity of cloud computing platform. In *2016 World Automation Congress (WAC)*, pages 1–4. IEEE.