Avaliação do Custo por Usuário de uma Aplicação de Rede Social na Amazon EC2

Matheus Cunha[1,2], **Nabor Mendonça**[1], **Américo Sampaio**[1]

[1]Mestrado em Informática Aplicada (MIA) – Universidade de Fortaleza (UNIFOR)
Av. Washington Soares, 1321, Edson Queiroz, CEP 60811-905 Fortaleza, CE

nabor@unifor.br, americo.sampaio@unifor.br

[2]Secretaria da Fazenda do Estado do Ceará (SEFAZ)
Av. Pessoa Anta, 274, Centro, CEP 60060-430 Fortaleza, CE

matheus.cunha@sefaz.ce.gov.br

Resumo. *Um dos principais desafios enfrentados pelos atuais clientes de nuvem que oferecem infraestrutura como serviço (IaaS) são as dificuldades para dimensionar os recursos da nuvem (tais como computação, armazenamento e rede) necessários para suas aplicações. Ainda é comum que clientes de nuvens provisionem os recursos da aplicação para mais (overprovisioning) ou para menos (underprovisioning), resultando, em ambos os casos, em prejuízos financeiros. Apesar do fato das plataformas de nuvem serem elásticas e proverem formas rápidas de adquirir ou liberar recursos, é importante entender a melhor maneira de fazer isto considerando a existência de vários provedores de nuvem oferecendo muitos serviços com preços diferentes. Este trabalho mostra alguns resultados bem interessantes de experimentos conduzidos em um benchmark popular de nuvem baseado em uma aplicação de rede social que executa na Amazon EC2. Nossos experimentos visam encontrar formas econômicas de escolher diferentes instâncias da EC2 baseado na demanda imposta na aplicação (medida pela quantidade de usuários concorrentes) e como escolher a instância que dá o melhor retorno em termos do seu custo por usuário.*

Abstract. *One of the main challenges faced by current users of infrastructure-as-a-service (IaaS) clouds are the difficulties to estimate cloud resources (such as computation, storage, and networking) according to their application needs. It is common that cloud users still overprovision or undeprovision (i.e. get more or less resources than needed, respectively), resulting, in both cases, in financial loss. Even though cloud platforms are elastic and provide fast ways to acquire or release resources it is important to understand the best ways to do that considering a vast amount of providers with many different service prices. This work shows some very interesting results from experiments conducted on a popular cloud benchmark based on a social network application running on top of the Amazon EC2 cloud. Ours experiments aim at finding cost-effective ways to select the different EC2 instance types based on the demand imposed to the application (measured in number of simultaneous users) and the instance that gives the best return in terms of its cost per user.*

1. Introdução

A computação em nuvem vem crescendo em popularidade nos últimos anos por fornecer um novo e atrativo modelo de negócios que se baseia em oferecer recursos computacionais (ex.: computação, armazenamento, aplicações, plataformas) como serviços que são pagos conforme sua utilização [Armbrust et al. 2009, Foster et al. 2009]. Diversas soluções para oferecer serviços de nuvem já existem, variando desde softwares *open source* que permitem configurar e gerenciar uma nuvem privada dentro de uma organização [Eucalyptus 2009, OpenNebula 2010] até os populares provedores comerciais de serviços de nuvem [Azure 2010, EC2 2010, Force 2010, AppEngine 2010, Rackspace 2009]. Alguns destes provedores, também conhecidos como nuvens de infra-estrutura (*IaaS clouds*) [EC2 2010, Rackspace 2009], oferecem recursos computacionais como servidores virtuais (máquinas virtuais contendo uma certa capacidade de CPU, memória e disco) e espaço de armazenamento na forma de serviços acessíveis via interfaces de programação (APIs). O principal objetivo é facilitar a aquisição de infraestrutura computacional para o cliente da nuvem (aquele que disponibiliza sua aplicação na nuvem) de modo que ele tenha que se preocupar cada vez menos com detalhes de gerenciamento de infraestrutura para se focar no seu negócio e no desenvolvimento da sua aplicação.

No entanto, para a maioria dos clientes, a grande oferta de provedores de nuvem traz um novo desafio que é o de escolher aquele que é o mais vantajoso considerando as características da sua aplicação. Por exemplo, caso a *Amazon EC2* seja a nuvem escolhida, o usuário terá um conjunto de mais de 10 tipos de instância (máquina virtual) para hospedar a sua aplicação com preços bastante variados. Cada instância possui sua própria configuração de em termos de memória (podendo variar, por exemplo, de 613 MB a 68,4 GB de RAM), processamento, armazenamento e performance de entrada e saída. Essas configurações muito distintas podem ser utilizadas para atender variados perfis de aplicação. Assim, para conseguir escolher a melhor configuração para uma determinada aplicação é importante que se conheça bem as características de cada tipo de instância oferecido pela plataforma de nuvem, bem como as necessidades específicas da aplicação, como, por exemplo, a potencial demanda (carga) imposta pelos usuários. Dependendo do tipo da aplicação, a demanda pode ser bastante variável. Por exemplo, uma aplicação de comércio eletrônico pode ter vários picos de uso, como em momentos de períodos festivos (ex.: natal, dia das mães), e também oscilar ao longo do dia. Apesar das plataformas de nuvem serem elásticas e seus clientes poderem ajustar mais facilmente os recursos necessários para atender a demanda da aplicação, a tarefa de dimensionar corretamente as instâncias ainda é complicada. Por isso, o risco de implantar aplicações com recursos além do necessário (*overprovisioning*) ou recursos aquém do necessário (*underprovisioning*) é muito alto. Ambos os casos implicam em perdas financeiras, só que por motivos distintos. No primeiro, o usuário pagará por recursos que não está utilizando; já no segundo, o usuário terá menos recursos do que necessita, o que poderá aumentar o tempo de resposta da sua aplicação e consequentemente a insatisfação de seus usuários.

Os problemas de *overprovisioning* ou *underprovisioning* não estão restritos ao momento de implantação da aplicação, e também podem ocorrer durante a sua operação no ambiente de nuvem. Portanto, é importante ter mecanismos que permitam entender a melhor maneira de ajustar os recursos da nuvem elasticamente para atender a possíveis flutuações na demanda da aplicação. Algumas questões que merecem ser investigadas neste sentido são:

- Qual o melhor tipo de instância (do ponto de vista de custo) que atende a uma demanda específica (por exemplo, até X usuários concorrentes)?
- Caso ocorra uma situação de baixíssima demanda, qual a melhor opção de instância?
- Qual a melhor estratégia: concentrar a carga em uma única instância de tamanho grande, ou distribuí-la em várias instâncias de tamanho menor?

Este trabalho procura descobrir respostas para questões como essas, que são bastante comuns no dia-a-dia dos clientes de nuvens na atualidade. Para isto, um *benchmark* de nuvem proposto recentemente [Sobel et al. 2008] foi utilizado para investigar a relação entre a demanda imposta a uma aplicação de rede social (tipo de aplicação bastante comum em nuvens) no ambiente de nuvem *Amazon EC2* (escolhido devido a sua alta popularidade). A avaliação demonstra como o custo por usuário varia de acordo com a demanda da aplicação e como a escolha correta do tipo de instância reflete diretamente no custo pago pelo uso da nuvem. Este estudo inicial conseguiu apontar resultados bastante interessantes e encorajadores no sentido de entender a melhor configuração possível para uma determinada demanda (baseada no número de usuários simultâneos da aplicação). Por exemplo, foi possível perceber que quando se tem uma demanda muito baixa (menor que 50 usuários simultâneos), a instância mais barata da EC2 pode ser suficiente, a um custo baixíssimo de 0,02 dólares por hora. Além disso, caso o usuário opte por utilizar 3 instâncias de tamanho médio ele consegue atender satisfatoriamente até 600 usuários simultâneos, economizando 25% em relação ao custo de 1 instância de tamanho extra grande.

O restante do artigo é organizado da seguinte forma. A seção 2 descreve outros *benchmarks* para ambientes de nuvem e suas limitações. Já a seção 3 apresenta as ferramentas utilizadas para a execução do *benchmark* e da aplicação de rede social utilizados neste trabalho. A seção 4 apresenta os experimentos que foram realizados e a seção 5 discute os resultados que foram observados. Finalmente, a seção 6 conclui o artigo e sugere trabalhos futuros.

2. Trabalhos Relacionados

Para realizar os experimentos deste trabalho foram estudadas diversas opções de *benchmark* de sistemas distribuídos disponíveis. Uma característica relevante é o fato da ferramenta de *benchmark* ser preparada para simular variações de carga (demanda) dado que os ambientes de computação em nuvem são baseados no modelo de pagar pelo uso. Além disso, é importante que o *benchmark* se baseie em aplicações comumente implantadas na nuvem para que as descobertas encontradas estejam próximas do perfil de aplicações reais.

Existem diversas ferramentas populares para avaliação de performance de sistemas distribuídos, como o SPECvirt [SPECvirt 2010]. O SPECvirt avalia a performance do hardware, da plataforma de virtualização, do sistema operacional convidado e de aplicações hospedadas (web, JEE e email) nesses sistemas operacionais convidados. Porém, a ferramenta fixa um conjunto específico de serviços virtualizados e a carga, quando variada, é aplicada em todos os componentes de software. Portanto, não é possível escalar apenas um determinado tipo de serviço ou desabilitar algum componente. Esta falta de flexibilidade na configuração do SPECvirt inviabilizou a escolha desta ferramenta para nossos experimentos.

Existem também algumas iniciativas de *benchmarks* específicos para computação em nuvem. Uma delas é a ferramenta implementado pela CloudSleuth [CloudSleuth 2011], que monitora diversos serviços disponibilizados em nuvens de infraestrutra, por exemplo, serviços de pagamento, mapas, propagandas e estatísticas de acesso. Sua metodologia consiste em disponibilizar uma aplicação simples de referência hospedada na Amazon EC2 que testa as funcionalidades de mapas, pagamento e outras descritas acima. A ferramenta permite a execução de testes de acesso que são disparados de mais de 168 países para analisar e comparar as variações de tempo de resposta. Embora interessante, essa ferramenta não permite variação na arquitetura da aplicação, o que é importante para nossos testes.

Outro projeto voltado para realizar *benchmarks* em serviços hospedados na nuvem é o *bitcurrent*[1], que apenas disponibiliza um relatório com o resultado de diversos testes que executou em provedores de nuvem. De forma similar, o CloudHarmony [CloudHarmony 2009], cujo objetivo é "tornar-se a principal fonte independente, imparcial e útil de métricas de desempenho dos provedores de nuvem", agrega dados de testes de performance realizados desde 2009 em mais de 60 provedores de nuvem. O projeto também oferece uma ferramenta para iniciar testes de performance a qualquer momento, denominada *Cloud Speed Test*.

O projeto CloudCmp [Li et al. 2010] visa realizar uma comparação da performance e do custo de diversos provedores de nuvem públicas, analisando a elasticidade, os mecanismos de persistência de dados e os serviços de rede oferecidos pelos provedores de nuvem. Para isso, desenvolveram uma ferramenta, a CloudCmp, e realizaram testes específicos para cada uma das três funcionalidades citadas anteriormente. O trabalho aponta que não há nenhum provedor que se destaque com relação aos demais e que os resultados obtidos apenas refletem o momento em que foram testados, uma vez que a estrutura utilizada para hospedar os serviços sofre modificações e a demanda nos recursos computacionais é bastante variável. Esta abordagem poderia ser adequada para realizar os experimentos deste trabalho, mas, como não está publicamente disponível, optamos por escolher o *benchmark* descrito na seção seguinte.

3. Cloudstone

O projeto Cloudstone [Sobel et al. 2008] disponibiliza um conjunto de ferramentas que permitem a execução de testes de performance com diferentes pilhas de software. Estas ferramentas incluem a aplicação Web 2.0, Olio, que implementa uma rede social, e a ferramenta responsável pela geração de carga e medição de performance da aplicação, Faban. Como a arquitetura da aplicação é facilmente configurável (ex.: pode-se variar facilmente o número de servidores de aplicação), já tendo sido testada com sucesso em nuvens como a EC2, optamos por utilizá-la para nossos experimentos.

3.1. Aplicação Olio

A aplicação Olio[2] é um projeto de código aberto que implementa uma rede social de calendário de eventos. Alguns exemplos de funcionalidade da aplicação são:

[1] http://www.bitcurrent.com

[2] http://incubator.apache.org/projects/olio.html

- Cadastro de usuários. Os usuários realizam seu cadastro para poder cadastrar eventos.
- Cadastro de eventos. Usuários registrados podem cadastrar eventos fornecendo informações como endereço e fotos sobre o local do evento. Os usuários também podem postar comentários acerca de um evento.

Quanto as tecnologias utilizadas, a ferramenta está disponível nas plataformas *Ruby on Rails* e PHP. A camada de apresentação utiliza AJAX e a a camada de dados pode ser configurada com os banco de dados MySQL ou PostgreSQL.

A figura 1 mostra a arquitetura da aplicação Olio organizada em três camadas: servidor web, servidor de aplicação e banco de dados. O conteúdo estático pode ficar hospedado em um servidor Apache ou Nginx[3] e esses dois ficam como proxy reverso e balanceador de carga para as intâncias do PHP ou Thin[4] para a versão Ruby on Rails. O uso dessa arquitetura torna possível a separação das camadas em diferentes máquinas virtuais, o que possibilita a investigação de diversos cenários e configurações, além de permitir um alto grau de elasticidade para atender às variações de demanda.

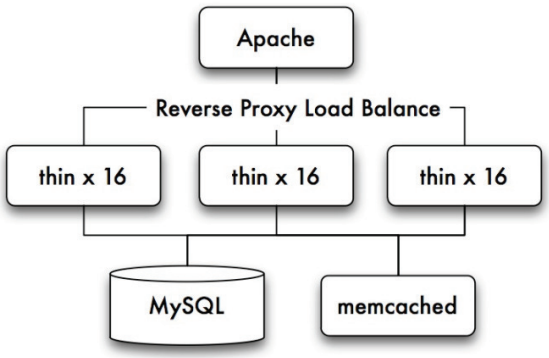

Figura 1. Arquitetura do Olio

3.2. Gerador de Carga Faban

Para realizar os testes de carga, o Cloudstone faz uso do Faban[5], que tem código aberto e é dividido em duas partes:

- Faban Driver Framework – é responsável pelo controle do ciclo de vida do *benchmark* e possui componentes para executar testes de aplicações implantadas em diversos servidores como Apache, Nginx, Thin, memcached, mysql e outros. Utiliza um modelo estocástico para simular a ação dos usuários.
- Faban Harness – é a ferramenta que automatiza a execução dos *benchmarks*. Cada *benchmark* implantado nessa ferramenta tem a sua execução acompanhada e configurada através de uma interface web que também disponibiliza os resultados das execuções.

[3]http://wiki.nginx.org/Main

[4]Disponível no endereço http://code.macournoyer.com/thin/, o Thin é um servidor web para aplicações Ruby

[5]http://java.net/projects/faban

Através da utilização do Faban é possível definir fluxos de trabalho completos, como cadastrar usuário ou cadastrar evento, que podem ser compostos por diversas requisições HTTP que configuram uma tarefa no sistema. Buscando se aproximar mais do comportamento de um usuário, a sequência de execução desses fluxos de trabalho é estocástica.

Outra possibilidade que o Faban oferece é a dos testes partirem de diferentes máquinas, todas coordenadas por um mesmo agente, permitindo uma grande quantidade de execuções em paralelo. Também a quantidade de usuários em cada execução é parametrizável e realizada através da interface web do Faban Harness. Durante uma execução, o Faban grava os tempos de resposta de cada requisição realizada pelo gerador de carga, do momento em que foi disparado até a hora da chegada do último byte da resposta. De posse dessas métricas, o Faban considera um teste bem sucedido aquele em que no mínimo noventa porcento das respostas tenham chegado dentro do tempo de resposta pré-estabelecido. Essa informação é bastante relevante pois a utilizamos como base do critério de sucesso para nossos experimentos.

4. Experimentos

Uma vez definida a aplicação alvo dos experimentos e o gerador de cargas para submetê-la a diferentes níveis de demanda, é preciso escolher a plataforma de nuvem onde os testes serão realizados. Para os nossos experimentos foi escolhida a *Amazon EC2*, região *East*, localizada no estado americano da Virgínia. Essa escolha se deu pela *Amazon EC2* ser uma referência no modelo de infraestrutura como serviço, e pelo fato dos preços dos recursos oferecidos nessa região serem os mais baixos cobrados pela *Amazon*. Porém, a *Amazon EC2* oferece diversos tipos de instância, com variações no custo, na quantidade de memória, no número de processadores, dentre outros recursos. Além disso, a opção por um determinado tipo de instância depende também da demanda esperada para a aplicação, que para uma aplicação nova é difícil de determinar. Portanto, os nossos experimentos tiveram como objetivo facilitar o entendimento de como esses diferentes tipos de instância se comportariam sendo expostos a variações na demanda pela aplicação. Dessa forma, seria possível identificar o tipo de instância mais adequado para hospedar a aplicação levando-se em consideração diferentes níveis de demanda.

Para os experimentos foram definidos duas faixas de demanda: demanda baixa (com o número de usuários concorrentes variando entre 25 e 150) e demanda moderada (com o número de usuários concorrentes variando entre 200 e 700). Sendo assim, a aplicação Olio foi hospedada em cada um dos diferentes tipos de instância da *Amazon EC2* avaliados, a qual foi então submetida aos diferentes níveis de demanda descritos acima. A mesma quantidade de usuários era executada três vezes durante os experimentos. Cada execução durava cerca de 800 segundos e era considerada realizada com sucesso quando noventa porcento dos tempos de resposta medidos estivessem dentro do tempo de resposta esperado. É importante deixar claro que os valores que serão reportados nas subseções seguintes apenas refletem o momento em que foram gerados, uma vez que o desempenho de uma instância da nuvem pode sofrer influência de outras instâncias hospedada no mesmo servidor físico.

Tipo de instância	Preço (US$ p/hora)	Diferença p/ anterior (%)	Diferença p/ menor (%)
t1.micro	0,02	—	—
m1.small	0,085	325	325
c1.medium	0,17	100	750
m1.large	0,34	100	1600
m2.xlarge	0,50	47	2400
c1.xlarge	0,68	36	3300
m1.xlarge	0,68	0	3300
m2.4xlarge	2,00	94	9900

Tabela 1. Preço absoluto (em dolar por hora de uso) e relativo (em percentual) dos diferentes tipos de instância oferecidos pela *Amazon EC2 East*.

Representação	Significado	# execuções com sucesso
○	Não atendeu à demanda	0
⊙	Atendeu à demanda esporadicamente	1
◎	Atendeu à demanda parcialmente	2
●	Atendeu à demanda plenamente	3

Tabela 2. Representação utilizada na análise do desempenho da aplicação.

4.1. Experimento 1: configuração com um único servidor de aplicação

Neste experimento foram utilizadas três instâncias. Na primeira era executado o gerador de carga (Faban), na segunda o banco de dados MySQL e na terceira o servidor de aplicação THIN e o balanceador de carga NGINX, necessários para a execução da aplicação Olio. Como o objetivo era conhecer como as instâncias se comportam sendo expostas a variações na demanda e qual o impacto de uma má escolha no custo da solução, foram fixadas as máquinas virtuais, do gerador de carga e do banco de dados, em instâncias do tipo *c1.xlarge*. Essa escolha foi baseada nos informações disponibilizadas pelo projeto Cloudstone. Já a aplicação Olio foi hospedada nos diversos tipos de instância oferecidos pela *Amazon EC2* (ver tabela 1), e submetida aos dois níveis de demanda descritos anteriormente.

Para facilitar o entendimento dos resultados, as tabelas subsequentes ilustram o desempenho da aplicação observado para cada instância e nível de demanda avaliados seguindo a representação visual descrita na tabela 2.

A tabela 3 mostra o desempenho da aplicação Olio, configurada com uma única instância hospedando o servidor de aplicação THIN, para a primeira faixa de demanda. Observando essa tabela é possível constatar que para o nível mais baixo de demanda, com até 25 usuários simultâneos, a aplicação Olio poderia ficar hospedada na menor instância disponibilizada pela *Amazon EC2*, do tipo *t1.micro*, ao custo de módico de US$ 0,02 por hora de uso. Porém, nesse tipo de instância, a aplicação só atendeu à demanda uma única vez, o que demonstra a instabilidade dos serviços da *Amazon* para instâncias com poucos recursos. Nesse sentido, a instância do tipo *m1.small* dá uma maior segurança, a um custo ligeiramente maior. Seguindo para 50 usuários e indo até 150, todos os tipos

Tipo de instância	Demanda (# usuários)					
	25	50	75	100	125	150
t1.micro	⊙	○	○	○	○	○
m1.small	●	○	○	○	○	○
c1.medium	●	●	●	●	●	●
m1.large	●	●	●	●	●	⊙
m2.xlarge	●	●	●	●	●	●
c1.xlarge	●	●	●	●	●	●
m1.xlarge	●	●	●	●	●	●
m2.4xlarge	●	●	●	●	●	●

Tabela 3. Desempenho da aplicação na nuvem sob demanda baixa (configuração com um único servidor de aplicação).

Tipo de instância	Demanda (# usuários)					
	200	300	400	500	600	700
c1.medium	○	○	○	○	○	○
m1.large	○	○	○	○	○	○
m2.xlarge	◎	○	○	○	○	○
c1.xlarge	●	●	◎	○	○	○
m1.xlarge	●	○	○	○	○	○
m2.4xlarge	●	●	⊙	○	○	○

Tabela 4. Desempenho da aplicação na nuvem sob demanda moderada (configuração com um único servidor de aplicação).

de instância avaliados satisfizeram o nível de serviço esperado, com exceção da instância *m1.large*, que não logrou êxito em todas as execuções com 150 usuários. Em vista da grande diferença de preço entre as instâncias, fica claro que uma escolha inadequada nessa faixa de demanda pode implicar em um custo até 1000% maior que o necessário. A instância que se destaca com o melhor custo/benefício é a *c1.medium*, que dentro dessa faixa de usuários passou em todos os testes.

A tabela 4 mostra os resultados do experimento para a faixa de demanda moderada. Nessa faixa, o custo inicial para hospedar a aplicação Olio sobe para US$ 0,5 por hora, com uma instância do tipo *m2.xlarge*. Porém, esse tipo de instância apresentou falha em um dos testes. Portanto, para maior estabilidade, as instâncias do tipo *c1.xlarge* são recomendadas. É possível notar que instâncias de tipos diferentes e com o mesmo preço, como é o caso das instâncias *c1.xlarge* e *m1.xlarge*, apresentam desempenhos bem diferentes. A razão é que esses tipos de instâncias são específicas para um determinado perfil de aplicação (por exemplo, com maior necessidade de memória ou processamento). Assim, pelos resultados obtidos, vê-se que a aplicação Olio apresenta um perfil que se beneficia de instâncias com maior poder de processamento. Também fica claro que pagar mais por uma instância com maior capacidade não implica necessariamente em um melhor serviço. Isso pode ser visto com as instâncias do tipo *m2.4xlarge*, que custam US$ 2,0 por hora de uso (a mais cara entre as instâncias avaliadas) e têm desempenho inferior às do tipo *c1.xlarge*, que custam quase três vezes menos.

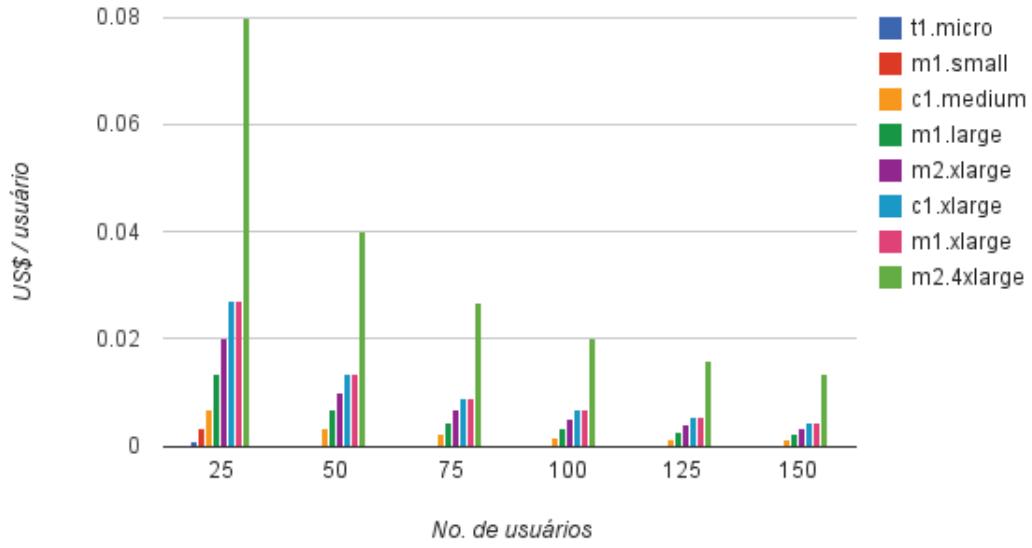

Figura 2. Custo por usuário da aplicação na nuvem sob demanda baixa (configuração com um único servidor de aplicação).

Uma vez conhecida a relação entre os tipos de instância e os níveis de demanda que eles conseguem suportar, é possível realizar uma análise da utilização desses recursos do ponto de vista econômico. Dessa forma, fica mais fácil entender o impacto da escolha de uma determinada instância no custo de operação da aplicação. Para isso, foi calculado o *custo por usuário* da aplicação, que é a relação entre o preço por hora de uso do tipo de instância escolhido e a quantidade de usuários atendidos por esse tipo.

Para entender melhor o cálculo desse custo, considere o custo por usuário da aplicação Olio para instâncias do tipo *m1.small*. Como nos testes essa instância executou com sucesso apenas para demandas de até 25 usuários, o valor do custo por usuário para esse tipo é calculado dividindo-se US$ 0,085 por 25, o que dá um custo de US$ 0,0034 por usuário.

O gráfico da figura 2 mostra os custos por usuário calculados para a aplicação Olio sob a faixa de demanda baixa. Através dele é possível observar que para 25 usuários, que simboliza o nível mais baixo de demanda considerado nos experimentos, a diferença dos preços é bastante expressiva. Variando de US$ 0,0008 (instância do tipo *t1.micro*) a US$ 0,08 (instância do tipo *m2.4xlarge*) por usuário, ou seja, a instância mais barata tem o custo 100 vezes menor que a instância mais cara. Essa diferença vai caindo à medida em que o número de usuários cresce. Com 150 usuários a instância mais cara é 11,76 vezes o valor da instância mais barata. Entre 50 e 150 usuários a instância *c1.medium* é a que oferece o melhor custo por usuário.

Já para a faixa de demanda moderada, cujos resultados são exibidos no gráfico da figura 3, fica claro o destaque das instâncias *c1.xlarge*, que oferecem uma melhor relação de custo quando a demanda passa dos 200 usuários. Com exatos 200 usuários, a instância

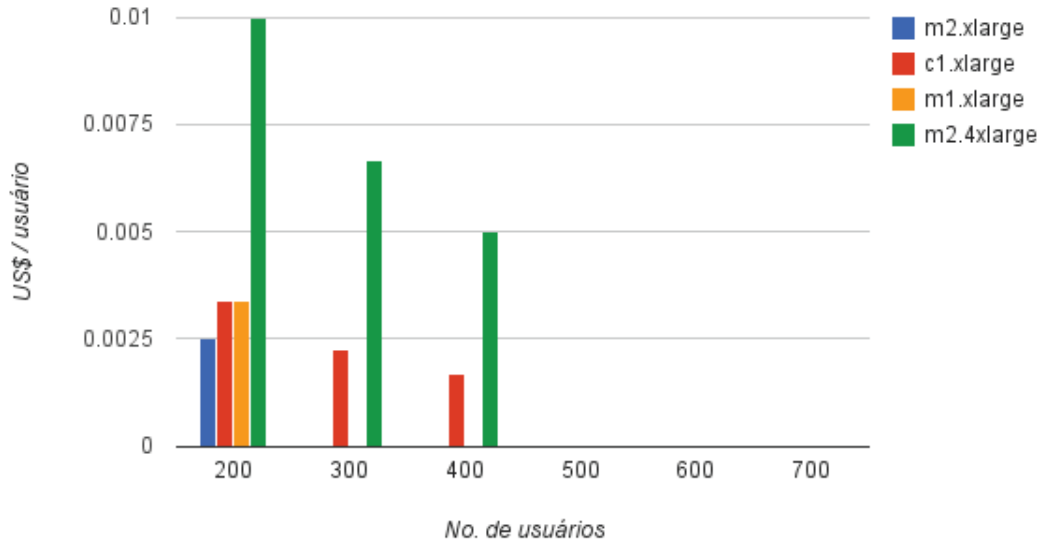

Figura 3. Custo por usuário da aplicação na nuvem sob demanda moderada (configuração com um único servidor de aplicação).

mais vantajosa é a *m2.xlarge*. Observe também que como as instâncias *c1.xlarge* e a *m1.xlarge* têm o mesmo preço, os seus custos para 200 usuários são os mesmos. A partir de 300 usuários a *m1.xlarge* some do gráfico, que é quando essa instância começa a não mais atender à demanda. A instância *m2.4xlarge*, por sua vez, consegue atender total ou parcialmente até 400 usuários, mas em nenhum momento é a opção mais vantajosa, uma vez que seus valores de custo por usuário são os maiores entre as instâncias avaliadas.

4.2. Experimento 2: configuração com múltiplos servidores de aplicação

Sendo conhecidos os limites de cada uma das instâncias, era preciso saber se mais de uma máquina virtual hospedando a aplicação Olio poderia resultar em uma melhor relação custo por usuário, particularmente sob níveis de demanda mais elevados. Portanto, mais uma vez foram fixadas as instâncias das máquinas virtuais do gerador de carga e do banco de dados nas instâncias do tipo *c1.xlarge*. E foram submetidos testes de desempenho variando-se o número de instâncias do servidor de aplicação. Os testes foram feitos com o servidor de aplicação THIN organizado em grupos de duas e três instâncias do tipo *c1.medium*.

A tabela 5 mostra os resultados para esse cenário. Nela pode ser observado que duas instâncias *c1.medium*, a um custo de US$ 0,34 por hora de uso, suportam o tráfego de até 300 usuários simultâneos. A partir dessa demanda só três *c1.medium* ou uma *c1.xlarge* conseguem atender no tempo de resposta acordado. Porém, acontece que a *c1.xlarge* custa US$ 0,68 por hora de uso e só executa com sucesso até 400 usuários, enquanto que três instâncias *c1.medium* custam US$ 0,51 por hora e atendem até 600 usuários.

O gráfico da figura 4 mostra os valores do custo por usuário para este experimento. Através dele é possível destacar que o paralelismo na camada de aplicação permitiu que

Tipo de instância	Demanda (# usuários)					
	200	300	400	500	600	700
m2.xlarge	◎	○	○	○	○	○
c1.xlarge	●	●	◎	○	○	○
m1.xlarge	●	○	○	○	○	○
m2.4xlarge	●	●	⊙	○	○	○
c1.medium (x2)	●	●	○	○	○	○
c1.medium (x3)	●	●	●	●	●	○

Tabela 5. Desempenho da aplicação na nuvem sob demanda moderada (configuração com múltiplos servidores de aplicação).

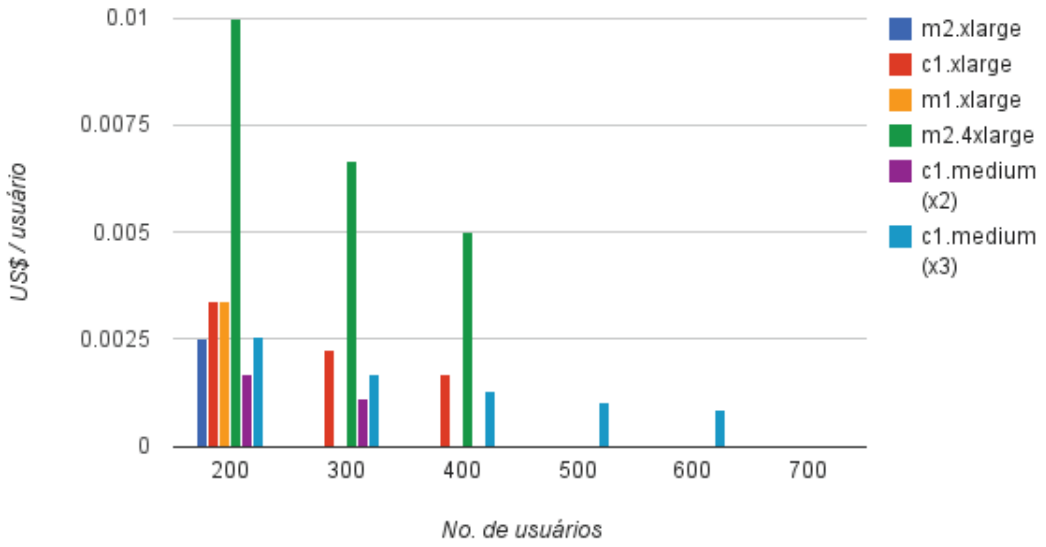

Figura 4. Custo por usuário da aplicação na nuvem sob demanda moderada (configuração com múltiplos servidores de aplicação).

uma demanda maior fosse atendida. Com 200 usuários a melhor instância tinha sido a *m2.xlarge*, mas com a utilização de duas *c1.medium* foi possível atender a essa mesma demanda a um menor custo, sendo possível atender até 300 usuários com esse grupo de instâncias. Já a *c1.xlarge*, que se destacou no experimento anterior, possui valores de custo por usuário superiores aos das outras configurações nos três níveis de demanda aos quais conseguiu atender. A partir de 400 usuários apenas duas instâncias *c1.medium* não são mais suficientes, e uma nova instância *c1.medium* se faz necessária. Esta nova configuração consegue atender até 600 usuários, com o custo das três instâncias somadas ainda sendo 25% inferior ao da instância *c1.xlarge*, a mais barata entre as instâncias com o melhor desempenho individual nessa faixa de demanda.

5. Discussão

Durante a realização dos experimentos, percebeu-se que o provedor de nuvem utilizado, *Amazon EC2*, apresentava uma grande flutuação na qualidade dos serviços oferecidos. Daí a necessidade de executar mais de uma vez os testes com a mesma quantidade de usuários. Em alguns casos, os mesmos testes executados com diferença de algumas horas, envolvendo os mesmos tipos de instância, produziam resultados bastantes diferentes. Esse comportamento, que também ocorre com outros provedores de nuvem, já foi documentando em outros trabalhos, como [Wang and Ng 2010] e [Li et al. 2010].

Em todo caso, os experimentos realizados neste trabalho fornecem dados que servem para entender melhor a relação entre a carga de uma aplicação e os tipos de instância que melhor atendem a um determinado nível demanda sob o ponto de vista de custos. Além disso, uma vez que o *benchmark* escolhido representa um perfil de aplicação que é bastante utilizado atualmente em plataformas de nuvem, esses dados podem auxiliar na implantação de novas aplicações que estão sendo disponibilizadas, ajudando, assim, a evitar o excesso ou mesmo a falta de provisionamento de recursos computacionais.

Tanto o excesso quanto a falta de provisionamento implicam em perdas financeiras, sendo que por motivos diferentes. No primeiro caso, provisionamento acima da demanda, uma aplicação pode estar sendo executada a um custo de US$ 5956,8/ano, em uma instância *c1.xlarge*, quando na verdade sua demanda poderia ser atendida por uma instância muito mais barata, por exemplo, do tipo *t1.micro*, o que faria o custo despencar para US$ 175,20/ano. Já no segundo caso, o custo com os serviços da nuvem será baixo mas o tempo de resposta da aplicação não atenderá a expectativa do usuário, o que pode fazer com o mesmo passe a não mais utilizar a aplicação e possivelmente fazer propaganda negativa do serviço prestado.

Portanto, a escolha correta da instância reflete diretamente em melhores tempos de resposta para os usuários e em uma grande economia nos custos. Um outro exemplo disso pode ser tirado dos experimentos, onde três instâncias do tipo *c1.medium* puderam atender até 600 usuários, enquanto que uma instância do tipo *c1.xlarge* conseguiu atender apenas 400 usuário. Acontece que as três instâncias *c1.medium*, juntas, são 25% mais baratas que uma instância *c1.xlarge* sozinha. Ou seja, gastando-se 25% menos é possível atender 33% mais usuários.

6. Conclusão

Como foi descrito nas seções anteriores, a escolha correta dos recursos computacionais necessários para hospedar uma aplicação na nuvem implica diretamente em menores custos com infraestrutura. Investir em uma instância sem conhecer o seu potencial de desempenho e o das demais pode significar um proviosionamento inadequado de recursos, o que também vai se refletir em perdas financeiras. Portanto, antes de uma aplicação ser implantada na nuvem, é preciso entender qual a configuração que melhor a atende, qual o tipo de instância que se adequa à sua necessidade computacional e como sua configuração pode ser adaptada para conseguir atender flutuações na demanda. Essa definitivamente não é uma tarefa simples para o cliente da nuvem, que muitas vezes precisa escalar rapidamente a sua aplicação.

Espera-se que o resultados apresentados neste trabalho possam servir de base para auxiliar o dimensionamento dos recursos computacionais de outras aplicações de perfis si-

milares ao da aplicação Olio, que poderão ser implantadas na nuvem da *Amazon* de forma mais criteriosa, conhecendo um pouco melhor a relação entre os custos das instâncias e seu potencial de desempenho. Nesse sentido, há a necessidade de realização de mais testes com perfis diferentes de aplicação e de utilização de recursos computacionais. Também pretende-se executar mais experimentos envolvendo outros provedores de nuvem, onde vai ser possível verificar se as relações entre o custo e capacidade computacional das instâncias observadas na nuvem da *Amazon* também ocorrem nessas outras plataformas.

Agradecimentos

Este trabalho é parcialmente financiado pela Fundação Edson Queiroz, Universidade de Fortaleza, através do Projeto OW2.

Referências

AppEngine (2010). Google app engine. `http://code.google.com/appengine/`.

Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al. (2009). Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*.

Azure (2010). Windows azure platform. `http://www.microsoft.com/windowsazure`.

CloudHarmony (2009). Cloudharmony. `http://cloudharmony.com/about/`.

CloudSleuth (2011). Cloudsleuth. `https://www.cloudsleuth.net/web/guest/home/`.

EC2 (2010). Amazon elastic compute cloud. `http://aws.amazon.com/ec2`.

Eucalyptus (2009). Eucalyptus the open source cloud plataform. `http://open.eucalyptus.com/`.

Force, S. (2010). Sales force. `http://www.salesforce.com/`.

Foster, I., Zhao, Y., Raicu, I., and Lu, S. (2009). Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee.

Li, A., Yang, X., Kandula, S., and Zhang, M. (2010). CloudCmp: Comparing Public Cloud Providers. In *Internet Measurement Conference*.

OpenNebula (2010). Open source tookit for cloud computing. `http://www.opennebula.org`.

Rackspace (2009). The rackspace cloud. `http://www.rackspacecloud.com/`.

Sobel, W., Subramanyam, S., Sucharitakul, A., Nguyen, J., Wong, H., Patil, S., Fox, A., and Patterson, D. (2008). Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0. In *Proc. of CCA*. Citeseer.

SPECvirt (2010). Specvirt. `http://www.spec.org/virt_sc2010/`.

Wang, G. and Ng, T. (2010). The impact of virtualization on network performance of amazon ec2 data center. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE.