Cloud Crawler: Um Ambiente Programável para Avaliar o Desempenho de Aplicações em Nuvens de Infraestrutura

Matheus Cunha, Nabor Mendonça, Américo Sampaio

[1]Programa de Pós-Graduação em Informática Aplicada (PPGIA)
Universidade de Fortaleza (UNIFOR)
Av. Washington Soares, 1321, Edson Queiroz, CEP 60811-905 Fortaleza, CE

mathcunha@gmail.com, nabor@unifor.br, americo.sampaio@unifor.br

Resumo. *Um dos principais desafios enfrentados pelos usuários de nuvens que oferecem infraestrutura-como-serviço (IaaS) é a dificuldade para dimensionar adequadamente os recursos virtuais necessários às suas aplicações. Embora muitos provedores de nuvem permitam adquirir e liberar recursos virtuais rapidamente, é importante que os usuários entendam o impacto que cada tipo de recurso oferecido pelo provedor pode exercer no desempenho das aplicações. Este trabalho apresenta um novo ambiente programável para apoiar os usuários na execução automática de testes de desempenho de aplicações em nuvens IaaS. O ambiente, denominado* Cloud Crawler*, é composto pela linguagem declarativa* Crawl*, usada para descrever diversos cenários de avaliação de desempenho de uma aplicação na nuvem, e o motor de execução* Crawler*, que executa os cenários descritos em* Crawl *e coleta os resultados das avaliações. O uso do ambiente é ilustrado através da avaliação do desempenho de uma aplicação de rede social, utilizando vários perfis de recursos virtuais sob diferentes níveis de demanda, em dois provedores IaaS comerciais.*

Abstract. *One of the main challenges faced by users of infrastructure-as-a-service (IaaS) clouds is the difficulty to adequately estimate the virtual resources necessary to their applications. Although many cloud providers offer fast ways to acquire and release resources, it is important that users have a prior understanding of the impact that each virtual resource type offered by the provider may impose on the performance of their applications. This work presents a new programmable environment to support users in automatically executing application performance tests in IaaS clouds. The environment, called* Cloud Crawler*, includes the* Crawl *declarative language, used to describe different performance evaluation scenarios for a given cloud application, and the* Crawler *execution engine, which executes the evaluation scenarios described in* Crawl *and collects their results. The use of the environment is illustrated through an evaluation of the performance of a social network application, using several virtual resources types under varying demand levels, in two commercial IaaS providers.*

1. Introdução

A computação em nuvem está revolucionando a indústria da Tecnologia da Informação, tendo sido cada vez mais adotada como uma alternativa mais econômica, escalável e robusta à aquisição de infraestrutura computacional própria por parte das

organizações [Armbrust et al. 2010]. Nesse contexto, tem havido um crescente interesse por nuvens do tipo IaaS (*infrastructure-as-a-service*), que oferecem recursos de infraestrutura (como máquinas virtuais, redes e espaço de armazenamento) na forma de serviço [Zhang et al. 2010]. Esse sucesso deve-se principalmente ao atrativo modelo de negócios oferecido pelos provedores desse tipo de nuvem, onde o preço cobrado pelos recursos é proporcional ao tempo de utilização, e ao fato de oferecerem poucas restrições a seus usuários quanto às tecnologias e aplicações que podem ser implantadas na infraestrutura virtual da nuvem [Rhoton 2009].

Por outro lado, o aumento do número de provedores de nuvens IaaS,[1] com cada provedor oferecendo uma ampla variedade de recursos virtuais com diferentes preços e perfis de configuração, traz à tona um novo desafio aos usuários interessados em disponibilizar suas aplicações na nuvem: como identificar os provedores e os perfis de recursos virtuais com o melhor custo/benefício para uma determinada aplicação? Para dimensionar os recursos virtuais a serem adquiridos de um provedor de nuvem, é necessário que o usuário, dono da aplicação, conheça bem não apenas as características (por exemplo, configuração, preço e localização geográfica) de cada perfil de recurso virtual oferecido pelo provedor, mas também o impacto que cada um desses perfis poderá ter no desempenho de sua aplicação. Se não houver um planejamento adequado por parte do usuário, a aplicação poderá vir a ser implantada na nuvem com recursos *aquém* ou *além* do necessário. Ambos os casos podem resultar em prejuízos para o usuário, embora por motivos distintos. No primeiro caso, a aplicação disporá de menos recursos do que de fato necessita, podendo afetar negativamente o seu desempenho ou até mesmo inviabilizar a sua execução; já no segundo, haverá desperdício de recursos, com o usuário tendo que pagar por algo que não irá efetivamente utilizar [Armbrust et al. 2010].

Embora vários trabalhos tenham sido publicados com foco na avaliação do desempenho de serviços e aplicações na nuvem (por exemplo, [Sobel et al. 2008, Ostermann et al. 2010, Malkowski et al. 2010, Li et al. 2010, Jayasinghe et al. 2011, Cunha et al. 2011], relativamente pouca atenção tem sido dada até o momento ao suporte ferramental necessário para facilitar a execução automática de diferentes testes de desempenho de aplicações por parte dos próprios usuários [Jayasinghe et al. 2012]. Este trabalho apresenta um novo ambiente programável para apoiar os usuários de nuvens IaaS na realização de testes automáticos de desempenho de aplicações na nuvem. As principais contribuições do ambiente são: a linguagem declarativa *Crawl*, com a qual os usuários podem especificar, através de uma notação simples e de alto nível de abstração, uma grande variedade de cenários de avaliação de desempenho de uma aplicação na nuvem; e o motor de execução *Crawler*, que automaticamente executa e coleta os resultados dos cenários descritos em *Crawl* em um ou mais provedores. Essas duas ferramentas, denominadas conjuntamente de *Cloud Crawler*, serão descritas em mais detalhes na Seção 2.

Para ilustrar as facilidades oferecidas pelo novo ambiente, foram realizados dois experimentos de avaliação de desempenho de uma aplicação de rede social em dois provedores IaaS comerciais, cujos cenários envolveram diferentes níveis de demanda pela aplicação e diferentes configurações de máquinas virtuais de cada provedor. Os resultados desses experimentos, descritos na Seção 3, mostram que o ambiente proposto, ao

[1] O sítio `cloudorado.com`, que oferece um serviço de busca e comparação de nuvens IaaS, lista, na data de elaboração do artigo, mais de 20 provedores localizados em diferentes partes do mundo.

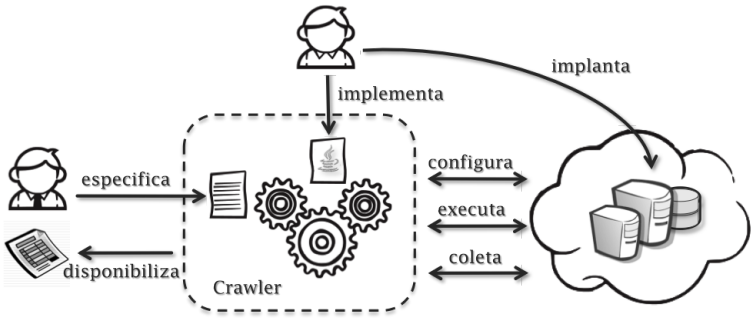

Figura 1. Visão geral do ambiente *Cloud Crawler*.

facilitar a especificação e a automação de diversos testes de desempenho da aplicação, pode ser uma importante ferramenta de suporte ao planejamento e à alocação de recursos virtuais em nuvens IaaS.

A última parte do artigo contém uma discussão mais pontual das contribuições do trabalho à luz de outros trabalhos relacionados (Seção 4), seguida das conclusões e sugestões para trabalhos futuros (Seção 5).

2. Ambiente *Cloud Crawler*

O ambiente *Cloud Crawler* é composto pela linguagem de descrição de cenários, *Crawl*, e o motor de execução dos cenários descritos em *Crawl*, de nome *Crawler*. O uso do ambiente envolve a execução de uma série de atividades por parte dos usuários, conforme ilustra a Figura 1.

Inicialmente, o usuário implanta os componentes da aplicação a ser avaliada (como sevidor web, servidor de banco de dados, etc.) na infraestrutura virtual do provedor de nuvem. Essa atividade é necessária haja visto que o ambiente ainda não oferece suporte à implantação automática de aplicações. Em seguida, o usuário implementa a interface a ser utilizada pelo *Crawler* para se comunicar com os componentes da aplicação na nuvem. Essa interface contém métodos básicos para interação com os recursos previamente implantados na nuvem, os quais devem ser customizados pelo usuário para atender as características específicas de sua aplicação. Após isso, o usuário especifica os cenários de avaliação da aplicação, utilizando a linguagem *Crawl*, e os repassa ao *Crawler* para que sejam executados na nuvem. De posse da especificação dos cenários em *Crawl*, e da implementação da interface de comunicação, o *Crawler* configura os recursos virtuais da nuvem de acordo com as necessidades de cada cenário, passando então a executá-los e a coletar seus resultados. Ao final, o *Crawler* consolida os resultados obtidos e os disponibiliza ao usuário na forma de planilhas, facilitando, assim, a sua posterior análise e visualização.

Vale ressaltar que a implantação dos componentes na nuvem e a implementação da interface de comunicação só precisam ser executas uma única vez por aplicação. Já a especificação e a execução dos cenários podem ser realizadas repetidas vezes, a critério do usuário. Dessa forma, é possível imaginar duas categorias (ou papéis) de usuários para o ambiente: especialistas no ambiente, que ficariam responsáveis pela sua extensão e customização para diferentes aplicações e provedores de nuvem; e especialista nas aplicações, que ficariam responsáveis por especificar os cenários de avaliação de desem-

penho e analisar e validar os seus resultados. Obviamente, nada impede que um mesmo usuário possa exercer ambos os papéis.

2.1. A Linguagem *Crawl*

Crawl[2] é uma linguagem declarativa criada com o objetivo de permitir a especificação das informações necessárias à execução automática de testes de desempenho em nuvens IaaS — tais como os componentes da aplicação e seus parâmetros de configuração, o provedor da nuvem, a quantidade e o perfil dos recursos virtuais a serem alocados a cada componente, e a carga de trabalho à qual a aplicação será submetida durante os testes —, através de um modelo sintático flexível e de alto nível de abstração.

Algumas características inerentes às nuvens IaaS tornam a especificação de cenários de avaliação para esse tipo de infraestrutura particularmente desafiadora. Uma delas é a necessidade de configurar certos componentes da aplicação com informações sobre os recursos virtuais da nuvem que só são conhecidos em tempo de execução. Por exemplo, em aplicações web é comum configurar os componentes da camada de negócio com o IP do servidor de banco de dados. Porém, na maioria dos provedores de nuvem IaaS, o IP de uma máquina virtual só é conhecido após a máquina ter sido efetivamente criada e iniciada, o que impede que os parâmetros de configuração dos componentes da aplicação possam ser totalmente definidos *a priori* (isto é, em tempo de especificação dos cenários de avaliação).

Outro desafio é a dinamicidade relacionada ao perfil dos recursos virtuais alocados à aplicação. Isso porque um mesmo componente da aplicação pode ter que ser configurado de diferentes maneiras, dependendo do perfil do recurso virtual que lhe for alocado. Por exemplo, o número adequado de *workers* (ou *threads*) que devem ser criados em um servidor web vai depender fortemente das características de hardware (como a quantidade de CPUs e o tamanho da memória RAM) da máquina virtual onde o servidor será executado. Para dar suporte a esse caráter dinâmico das aplicações na nuvem, *Crawl* oferece mecanismos que permitem (re)configurar, em tempo de execução dos cenários, diversos aspectos relacionados à utilização dos recursos da nuvem e de seus componentes de software.

2.1.1. Modelo de Entidades

Crawl provê um conjunto de dez tipos de entidades, conforme ilustrado na Figura 2, com as quais o usuário do ambiente pode especificar uma ampla variedade de cenários de avaliação de desempenho em nuvens IaaS. Estes tipos constituem o modelo de entidades da linguagem e estão descritos na Tabela 1. Para definir as entidades de *Crawl* que comporão os cenários de avaliação da aplicação, o usuário do ambiente faz uso da representação textual apresentada a seguir.

2.1.2. Representação Textual

Em princípio, qualquer notação textual estruturada poderia ser adotada para representar as entidades de *Crawl*. Nesse sentido, a linguagem de marcação XML surge

[2]O nome da linguagem é um acrônimo para **C***loud* **r***esource* **a***pplication and* **w***orkload* **l***anguage*.

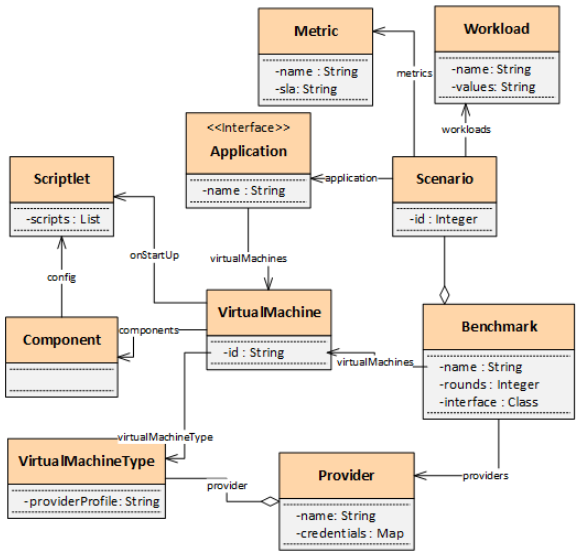

Figura 2. Modelo de entidades de *Crawl*.

como uma candidata natural, devido à sua universalidade e ao seu amplo suporte ferramental. Recentemente, porém, XML tem perdido espaço como formato de representação de dados para outras notações textuais mais simples e compactas, como JSON [JSON 2011] e Yaml [Ben-Kiki et al. 2001], particularmente no ambiente da web [Fonseca and Simões 2007]. Yaml, em especial, tem se destacado pela sua sintaxe extremamente simples e flexível, além de ser um superconjunto funcional de JSON [Fonseca and Simões 2007]. Por essas razões, Yaml foi escolhida como base para a representação textual da linguagem *Crawl*.

Yaml usa indicadores estruturais limpos e minimalistas, baseados fortemente na identação de elementos textuais encadeados [Ben-Kiki et al. 2001]. A Tabela 2 apresenta os principais indicadores de Yaml e suas respectivas funções sintáticas dentro da linguagem. Além da simplicidade, outra vantagem de Yaml é a sua flexibilidade, permitindo que novas entidades sejam definidas a qualquer momento e referenciadas sempre que necessário. Essas características se refletem em uma grande vantagem para o usuário da linguagem *Crawl*, que passa a ter em mãos uma simples mas poderosa notação para especificar os componentes de sua aplicação e os cenários de avaliação de seu interesse.

A Figura 3 mostra um exemplo de um cenário de avaliação especificado em *Crawl*. Nesse exemplo, o primeiro bloco da especificação (linhas 1-4) contém a definição de uma entidade do tipo *Benchmark* e três de seus atributos, os quais definem, respectivamente, o nome do *Benchmark* (linha 2), o número de vezes que este deverá executado pelo *Crawler* (linha 3), e o intervalo de tempo, em milisegundos, a ser aguardado entre cada uma de suas execuções (linha 4).

Em seguida, nas linhas linhas 7-16, é definido o provedor de nuvem a ser utilizado (no caso, Rackspace), cujos atributos incluem as credencias de acesso ao provedor, tal como fornecidas pelo usuário (linha 9), e dois perfis de máquinas virtuais, de nomes *flavor1* (linhas 11-13) e *flavor2* (linhas 14-16). As linhas 19-37, por sua vez, contêm a definição de duas máquinas virtuais do provedor Rackspace. A primeira, do perfil *flavor1*, está configurada com uma instância do servidor web Apache (linha 24), enquanto

Tipo	Descrição
Benchmark	Representa a entidade maior do modelo. Contém o contexto global de uma avaliação de desempenho especificada em *Crawl*. Nela, são definidos os provedores de nuvem, os cenários de avaliação e demais atributos de escopo global.
Provider	Representa um provedor de nuvem. Possui como atributos as credenciais de acesso à nuvem, tal como fornecidas pelo usuário, e uma lista com os diferentes perfis de máquinas virtuais fornecidos pelo provedor.
VirtualMachineType	Representa um perfil de máquina virtual oferecido pelo provedor de nuvem.
VirtualMachine	Representa uma máquina virtual a ser utilizada para executar um ou mais componentes de uma aplicação na nuvem. A associação entre a entidade declarada em *Crawl* e a máquina virtual previamente criada no provedor é feita através do atributo *ID*, cujo valor deve ser idêntico ao ID da máquina virtual tal como atribuído pelo provedor. Pode ser declarada com escopo global, associada à entidade *Benchmark* (nesse caso, a máquina virtual será ligada no início da execução dos cenários e desligada após o último cenário ter sido executado), ou local, associada à entidade *Application* (nesse caso, a máquina virtual será ligada e desligada apenas no contexto da execução do cenário onde foi declarada).
Component	Entidade que permite descrever os componentes da aplicação (por exemplo, servidor web, servidor de banco de dados, gerador de carga) que serão executados nas respectivas máquinas virtuais.
Scriptlet	Representa um *script* que será executado em uma máquina virtual como parte da configuração inicial de um componente ou serviço alocado a essa máquina.
Application	Representa a aplicação cujo desempenho será avaliado na nuvem. Através dessa entidade é possível definir a arquitetura de implantação da aplicação em termos de suas máquinas virtuais, que podem ser tanto de escopo local quanto global.
Metric	Representa uma métrica de desempenho (por exemplo, tempo médio de resposta, tempo total de execução, etc.) que será utilizada como critério de avaliação em um ou mais cenários.
Workload	Representa uma carga de trabalho a ser submetida à aplicação. O mecanismo de geração de carga é definido pelo próprio usuário, como parte da implementação da interface de comunicação com a nuvem utilizada pelo *Crawler*.
Scenario	Representa um cenário de avaliação de desempenho que será executado na nuvem pelo *Crawler*. Cada cenário é especificado envolvendo uma única aplicação. Possui como atributos um conjunto de métricas e um conjunto de cargas de trabalho, cujos valores definem como o *Crawler* irá avaliar o desempenho da aplicação na nuvem.

Tabela 1. Descrição das entidades de *Crawl*.

Símbolo	Função
:	Atribui o valor especificado no lado direito à chave declarada no lado esquerdo
–	Declara um elemento de uma sequência
!	Denota um tipo de entidade
&	Denota uma entidade referenciável
*	Referencia uma entidade previamente declarada

Tabela 2. Principais indicadores sintáticos de Yaml.

a segunda, do perfil *flavor2*, está configurada com uma instância do servidor de banco de dados Postgres (linha 36). Note que as duas máquinas virtuais estão associadas aos seus respectivos recursos virtuais na nuvem, os quais devem ter sido previamente criados junto ao provedor, através dos identificadores 7HJ38K (linha 21) e YH838K (linha 33). Note, ainda, que foi definida uma entidade do tipo *Scriptlet* como atributo da máquina virtual configurada com o servidor Apache (linhas 27-30). Essa entidade contém os dados do *script* que será executado pelo *Crawler* como parte do procedimento de inicialização dessa máquina virtual, e que tem como finalidade configurar o servidor web com o IP da máquina virtual onde será executado o servidor de banco de dados.

A última parte da especificação (linhas 40-52) contém a definição do cenário propriamente dito. Note que o cenário inclui a definição de uma carga de trabalho, cujos valores variam entre 100 e 400, representando o número de usuários concorrentes da aplicação (linhas 43-46) e de uma métrica de desempenho baseada no tempo de resposta da aplicação, com um acordo de nível de serviço (SLA) de 90%[3] (linhas 47-49). Por

[3]Um SLA de 90% indica que pelo menos 90% das requisições enviadas à aplicação pelo gerador de carga devem ser respondidas em um tempo igual ou inferior ao tempo máximo estabelecido pelo usuário.

```
1  !benchmark
2  name: bench_1
3  rounds: 3
4  interval: 5000
5
6  # definição do provedor
7  providers:
8   -&rackspace !provider
9     credentialPath: Rackspace.txt
10    -virtualMachineTypes:
11       -&flavor1 !virtualMachineType
12          providerProfile: 1
13          provider: *rackspace
14       -&flavor2 !virtualMachineType
15          providerProfile: 2
16          provider: *rackspace
17
18 # definição das máquinas virtuais
19 virtualMachines:
20  - &web_vm !virtualMachine
21    id: 7HJ38K
22    type: *flavor1
23    components:
24     - &apache !component
25        id: 1
26        onStartup:
27         -&apacheConfig !scriptlet
28           id: 1
29           scripts:
30            db.sh ${db.IP}
31
32  - &database_vm !virtualMachine
33    id: YH838K
34    type: *flavor2
35    components:
36       - &postgres !component
37          id: 2
38
39 # definição do cenário
40 scenarios:
41   - !scenario
42     id: 1
43     workloads:
44      - &load !workload
45         id: 1
46         values: [100,200,300,400]
47     metrics:
48      - &responseTime !metric
49        sla: 90
50     web_app: !application
51       db_layer: *database_vm
52       web_layer: *web_vm
```

Figura 3. Exemplo da especificação de um cenário de avaliação em *Crawl*.

fim, as linhas 50-52 definem a arquitetura de implantação da aplicação, a qual é composta pelas duas máquinas virtuais configuradas com os servidores Apache e Postgres, respectivamente.

De modo a facilitar ainda mais a especificação dos cenários de avaliação por parte dos usuários, *Crawl* oferece duas extensões à sintaxe original de Yaml. A primeira permite a inclusão de entidades definidas em arquivos externos, o que torna as especificações em *Crawl* mais modulares, reutilizáveis e fáceis de compreender, particularmente para cenários envolvendo um grande número de entidades. A segunda extensão permite a definição de laços de interação, através dos quais é possível definir, de uma forma simples e compacta, um conjunto de entidades similares que compartilham vários elementos em comum. Essas duas extensões são ilustradas na Figura 4, que mostra um exemplo de uma especificação em *Crawl* com as definições dos conjuntos de perfis de máquinas virtuais fornecidos pelos provedores Amazon EC2 e Rackspace. Note que a definição dos perfis do provedor Amazon EC2 é incluída diretamente de um arquivo externo, disponível no caminho /lib/providers/ec2/vm_profiles.crawl, através da definição da entidade especial include (linhas 2-3).

Já a definição dos perfis do provedor Rackspace é feita utilizando um laço de iteração, através da declaração da entidade especial foreach (linhas 6-14). Essa entidade possui quatro atributos: list, que define a lista com os valores a serem iterados no laço; count, que corresponde ao contador de iterações do laço, cujo valor inicial é definido pelo usuário e incrementado de uma unidade cada nova iteração; var, que identifica o nome da variável que receberá cada item da lista de valores do laço; e statement, que contém a definição de todos elementos que fazem parte do escopo do laço. No escopo do laço é possível usar um segundo delimitador de variável, $[], através do qual pode-se referenciar os atributos count e var, os quais serão substituídos por seus respectivos valores a cada iteração do laço. Dessa forma, a cada iteração são geradas novas definições para os elementos contidos no escopo do laço, com cada definição diferindo das anteriores apenas nos trechos que referenciam os valores dos atributos count e var.

No caso do exemplo da Figura 4, note que a lista de valores do laço (linha 8) corresponde exatamente à sequência de inteiros que identificam cada um dos perfis disponibilizados pelo provedor Rackspace. Note também que apenas uma entidade do tipo *VirtualMachineType* está sendo explicitamente definida dentro do escopo do laço (linhas 12-14), e que tanto o nome dessa entidade (linha 12) quanto o valor de seu atributo de nome providerProfile (linha 13) fazem referência à variável do laço, no caso, flavor. Durante o

```
1  # definição dos perfis do provedor Amazon EC2
2  !include
3    file: /lib/providers/ec2/vm_profiles.crawl
4
5  # definição dos perfis do provedor Rackspace
6  virtualMachineTypes:
7  - !foreach
8    list: [1, 2, 3, 4, 5, 6, 7, 8, 9]
9    count: 0
10   var: flavor
11   statement:
12 - &flavor${flavor} !virtualMachineType
13     providerProfile: ${flavor}
14     provider: *rackspace
```

Figura 4. Especificação de múltiplos perfis de máquinas virtuais em *Crawl*.

processamento da especificação *Crawl*, realizado pelo motor *Crawler*, essa variável será substituída por cada um dos valores da lista em cada iteração do laço (no caso, 1, 2, 3 e assim sucessivamente), resultando na definição de nove novas entidades do tipo *VirtualMachineType*, diferentes entre si apenas nos trechos que foram substituídos pelos valores das variáveis do laço.

2.2. O Motor *Crawler*

O motor *Crawler* é o mecanismo encarregado de validar, executar e coletar os resultados dos cenários de avaliação descritos em *Crawl*. Também fazem parte de suas funções gerenciar o ciclo de vida das máquinas virtuais na nuvem e configurar os componentes da aplicação de acordo com as necessidades de cada cenário. A versão atual do motor foi desenvolvida inteiramente em Java, na forma de um serviço RESTful que pode ser executado tanto em uma infraestrutura local quanto em algum provedor de nuvem.

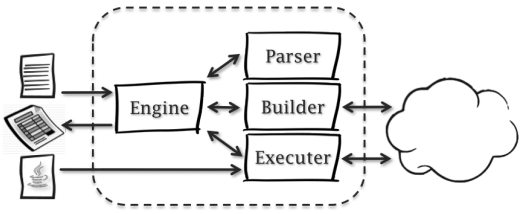

Figura 5. Arquitetura do *Crawler*.

A arquitetura do *Crawler* é composta de quatro módulos: *Engine*, *Parser*, *Builder* e *Executer* (ver Figura 5). *Engine* é o módulo principal da ferramenta, e tem como função responder aos comandos de invocação do usuário e coordenar as ações dos demais módulos. O *Parser* tem a função de interpretar e validar a especificação dos cenários descritos em *Crawl*, também sendo responsável pela resolução das variáveis declaras dinamicamente na linguagem. A função do *Builder* é preparar os recursos virtuais na nuvem e configurar os componentes da aplicação que serão executados nesses recursos. Para a preparação dos recursos, o *Builder* utiliza a biblioteca de código aberto *jclouds*,[4] e para a configuração dos componentes na nuvem é utilizada a ferramenta *JSch*.[5] O módulo *Executer* se encarrega de executar os cenários na nuvem com os recursos preparados pelo *Builder* e de coletar os resultados das avaliações. A interação do *Executer* com os componentes da aplicação durante a execução dos cenários é feita exclusivamente através da interface de comunicação com a nuvem, cuja implementação é fornecida pelo usuário. Ao final, o módulo *Engine* consolida os dados coletados em cada cenário pelo *Executer* e em seguida os disponibiliza ao usuário na forma de planilhas.

[4] http://www.jclouds.org/
[5] http://www.jcraft.com/jsch/

Uma característica importante do *Crawler* é a forma como ele gerencia o ciclo de vida das máquinas virtuais na nuvem. Em vez de solicitar a criação de novas máquinas virtuais ao provedor de nuvem, o *Crawler* reutiliza as máquinas virtuais previamente criadas pelo usuário, apenas alterando os seus perfis, caso necessário. Essa estratégia tem como principal vantagem a diminuição do custo de execução dos cenários na nuvem, uma vez que, como a maioria dos provedores IaaS atuais adota um modelo de precificação onde é cobrado um mesmo valor por hora ou fração de hora utilizada, dependendo do tempo de execução dos cenários, reutilizar uma máquina virtual existente pode sair muito mais barato do que solicitar a criação de uma nova máquina ao provedor.

3. Experimentos

Esta seção descreve dois experimentos realizados com o intuito de ilustrar o uso e as facilidades propiciadas pelo ambiente *Cloud Crawler*.

3.1. Metodologia e Especificação dos Cenários

Os dois experimentos envolveram a avaliação do desempenho de uma mesma aplicação de rede social de código aberto, Olio,[6] em dois provedores IaaS comercias, Amazon EC2 e Rackspace, os mais utilizados atualmente.[7] O objetivo era investigar quais configurações de recursos virtuais de cada provedor ofereceriam a melhor relação custo/desempenho para a aplicação, considerando diferentes níveis de demanda.

A Olio foi originalmente desenvolvida no âmbito do projeto Cloudstone [Sobel et al. 2008], com parte de um novo *benchmark* para nuvens IaaS. A aplicação está disponível em duas versões, PHP e Ruby on Rails (RoR), sendo que apenas essa última foi utilizada nos experimentos. Ambas as versões adotam uma arquitetura web tradicional, composta de três camadas: apresentação, aplicação e dados. No caso da versão RoR, os componentes utilizados em cada camada são, respectivamente, o servidor web Nginx,[8] o servidor de aplicação Thin,[9] e o servidor de banco de dados MySQL.[10] Em princípio, cada um desses componentes pode ser executado em uma máquina diferente, embora a arquitetura de implantação típica da Olio seja executar os servidores Nginx e Thin concorrentemente, em uma mesma máquina, com o Nginx servindo de *proxy* reverso e balanceador de carga para as múltiplas instâncias do Thin, deixando o servidor de banco de dados em uma máquina separada [Sobel et al. 2008]. Essa mesma arquitetura foi utilizada nos experimentos.

Para simular os diferentes níveis de demanda a que a Olio seria submetida nos diferentes cenários, foi escolhida a ferramenta de geração de carga Faban,[11] também utilizada no projeto Cloudstone. Visando limitar o número total de configurações de recursos da nuvem a serem investigados em cada provedor, decidiu-se implantar a aplicação fixando-se os perfis das máquinas virtuais onde seriam executados o gerador de carga e o servidor de banco de dados. No caso da Amazon, utilizou-se o perfil *c1.xlarge*, o mesmo utilizado no Cloudstone; já no caso do Rackspace, utilizou-se o perfil *flavor 5*,

[6] http://incubator.apache.org/projects/olio.html
[7] http://gigaom.com/cloud/amazon-is-no-1-whos-next-in-cloud-computing/
[8] http://nginx.org/en/
[9] http://code.macournoyer.com/thin/
[10] http://www.mysql.com/
[11] http://java.net/projects/faban

que possui uma configuração de hardware similar ao referido perfil da Amazon. Dessa forma, as variações concentraram-se no perfil e na quantidade das máquinas virtuais utilizadas para executar os servidores Nginx e Thin, bem como nos valores dos parâmetros de configuração específicos de cada um desses dois servidores (no caso, o número de *workers* do Nginx e o número de instâncias do Thin). Essa estratégia de implantação permitiu avaliar tanto a escalabilidade horizontal da aplicação, alterando-se o número de máquinas virtuais alocadas aos servidores Nginx e Thin, quanto a sua escalabilidade vertical, alterando-se o perfil das máquinas alocadas a esses dois componentes e ajustando-se os seus respectivos parâmetros de configuração de acordo com o perfil escolhido.

Em todos os cenários investigados, a aplicação foi submetida a duas cargas de trabalho, definidas em termos do número de usuários concorrentes que seriam simulados pelo gerador de carga. Na primeira carga, denominada *demanda baixa*, o número de usuários concorrentes variava entre 25 e 150, enquanto na segunda, denominada *demanda moderada*, esse número variava entre 200 e 800. Durante os experimentos, cada cenário foi executado três vez para cada número de usuários de cada uma das cargas de trabalho. Essa medida visava capturar eventuais flutuações no desempenho dos recursos virtuais da nuvem, decorrentes da possível concorrência de outros usuários compartilhando a mesma infraestrutura física.

Como métrica de desempenho, foi utilizado o tempo de resposta da aplicação. Cada execução de um cenário para cada nível de demanda era considerada bem sucedida se pelo menos 90% das respostas recebidas pelo gerador de carga estivessem dentro do limite máximo estabelecido para cada operação da aplicação. Nos experimentos, esses limites foram definidos com os mesmos valores utilizados no projeto Cloudstone [Sobel et al. 2008].

A Figura 6 mostra a especificação em *Crawl* dos cenários de avaliação da aplicação Olio na nuvem Amazon EC2.[12] A especificação para a nuvem Rackspace é feita de forma análoga, mudando-se apenas as definições relativas aos perfis de máquinas virtuais de cada provedor, e foi omitida por limitações de espaço.

A primeira parte da especificação (linhas 1-41) define os atributos globais do *Benchmak*, incluindo: um conjunto de perfis de máquinas virtuais (linhas 8-20), utilizados como perfis das máquinas virtuais onde serão executados os servidores Nginx e Thin; duas máquinas virtuais (linhas 22-34), onde serão executados o gerador de carga (Faban) e o servidor de banco de dados (MySQL), respectivamente; e duas cargas de trabalho (linhas 36-41). Note que a entidade especial include é utilizada, logo no início da especificação (linhas 4-5), para importar as definições relativas aos provedores de nuvem de um arquivo externo de nome providers.crawl. Note ainda que cada perfil de máquina virtual foi definido com uma lista de propriedades contendo dois atributos, de nomes workers e thinServers. Através desses dois atributos o usuário pode definir parâmetros de configuração específicos da aplicação, os quais serão utilizados pelo *Crawler* durante a execução dos cenários para configurar o número de *threads* do sevidor Nginx e o número de instâncias do servidor Thin, respectivamente, de acordo com as características de hardware das máquinas virtuais onde cada um será executado.

A segunda e última parte da especificação (linhas 43-101) define os cenários de

[12] A especificação foi ligeiramente modificada no artigo para facilitar a sua visualização.

```
1  !benchmark
2  name: OlioBenchmark
3  rounds: 3
4  - !include
5    file: providers.crawl
6
7  properties:
8   - &low_prof !virtualMachineType
9     providerProfile: t1.micro
10    provider: *ec2
11    properties:
12     - workers: 2
13     - thinServers: 2
14  ...
15  - &high_prof !virtualMachineType
16    providerProfile: c1.xlarge
17    provider: *ec2
18    properties:
19     - workers: 6
20     - thinServers: 18
21
22 virtualMachines:
23  - &faban !virtualMachine
24     id: 20408853
25     type: *high_prof
26     name: DRIVER
27     components:
28      - *fabanApp
29  - &mysql !virtualMachine
30     id: 20406221
31     type: *high_prof
32     name: MYSQL
33     components:
34      - *mysqlApp
35
36 workloads:
37  - &low !workload
38    values: [25,50,75,100,125,150]
39  - &moderate !workload
40    values: [200,300,400,500,600,
41                700,800]
42
43 scenarios:
44  # implantação vertical
45  - !foreach
46    list: [*low_prof, ..,
47           *high_prof]
48    var: profile
49    statement:
50    - !scenario
51      workloads:
52       - *low
53       - *moderate
54      metrics:
55       - *responseTime
56      application: !OlioApp
57        database: *mysql
58        driver: *faban
59        web:
60         - !virtualMachine
61           id: 20400657
62           name: THIN SERVER
63           type: $[profile]
64           components:
65            - *railsApp
66            - *tomcatApp
67            - *nginxApp
68            - !component
69              config:
70               - &db_script !scriptlet
71                 scripts:
72                   - sudo
73                     /home/db.sh ${IP}
74
75 # implantação horizontal
76 - !scenario
77   workloads:
78    - *low
79    - *moderate
80   metrics:
81    - *responseTime
82   application: !OlioApp
83     database: *mysql
84     driver: *faban
85     web:
86      - !foreach
87        list: [239999, ..,
88               9989989]
89        var: machine_id
90        statement:
91         - !virtualMachine
92           id: $[machine_id]
93           name: THIN SERVER
94           type: *high_profile
95           components:
96            - *railsApp
97            - *tomcatApp
98            - *nginxApp
99            - !component
100             config:
101              - *db_script
```

Figura 6. Especificação dos cenários de avaliação da Olio na Amazon EC2.

avaliação da Olio considerando duas arquiteturas de implantação: uma arquitetura vertical, na qual os servidores Nginx e Thin são implantados em uma única máquina virtual com diferentes perfis (linhas 45-73); e uma arquitetura horizontal, na qual os servidores Nginx e Thin são implantados em múltiplas máquinas virtuais de mesmo perfil (linhas 76-101). Note que a entidade especial foreach é utilizada em ambas arquiteturas de implantação para definir as máquinas virtuais de cada cenário variando apenas seus perfis (linhas 45-49) ou seus IDs (linhas 86-90). Note também que as máquinas virtuais da camada web em ambas arquiteturas são definidas reutilizando um mesmo *script* de configuração (o qual é definido nas linhas 70-73 e referenciado na linha 101), cujo papel é configurar os componentes da camada web com o endereço IP da máquina virtual onde será executado o servidor de banco de dados.

Deve-se enfatizar aqui o modo através do qual foram definidas as máquinas virtuais utilizadas na avaliação da Olio. As máquinas virtuais referentes ao balanceador de carga e ao servidor de banco de dados foram definidas como sendo de escopo global, o que significa que essas máquinas serão ligadas no início do processo de avaliação e desligadas apenas ao seu final. Essa decisão justifica-se pelo fato de que essas duas máquinas serão utilizadas na execução de todos os cenários de avaliação definidos para a Olio. Já as máquinas virtuais referentes aos componentes da camada web foram definidas como sendo de escopo local, o que significa que elas serão ligadas e desligadas no contexto específico de seus respectivos cenários. Dessa forma, máquinas virtuais que não forem realmente necessárias em um cenário específico permanecerão desligadas durante a execução desse cenário pelo *Crawler*, resultando em uma significativa economia de recursos para o usuário da nuvem.

Devido a restrições de espaço, também foram omitidos deste artigo os aspectos referentes à implantação dos componentes da Olio nos dois provedores e à customização do *Crawler* através da implementação de sua interface de comunicação. Uma descrição mais detalhada desses e de outros aspectos do ambiente pode ser encontrado em [Cunha 2012].

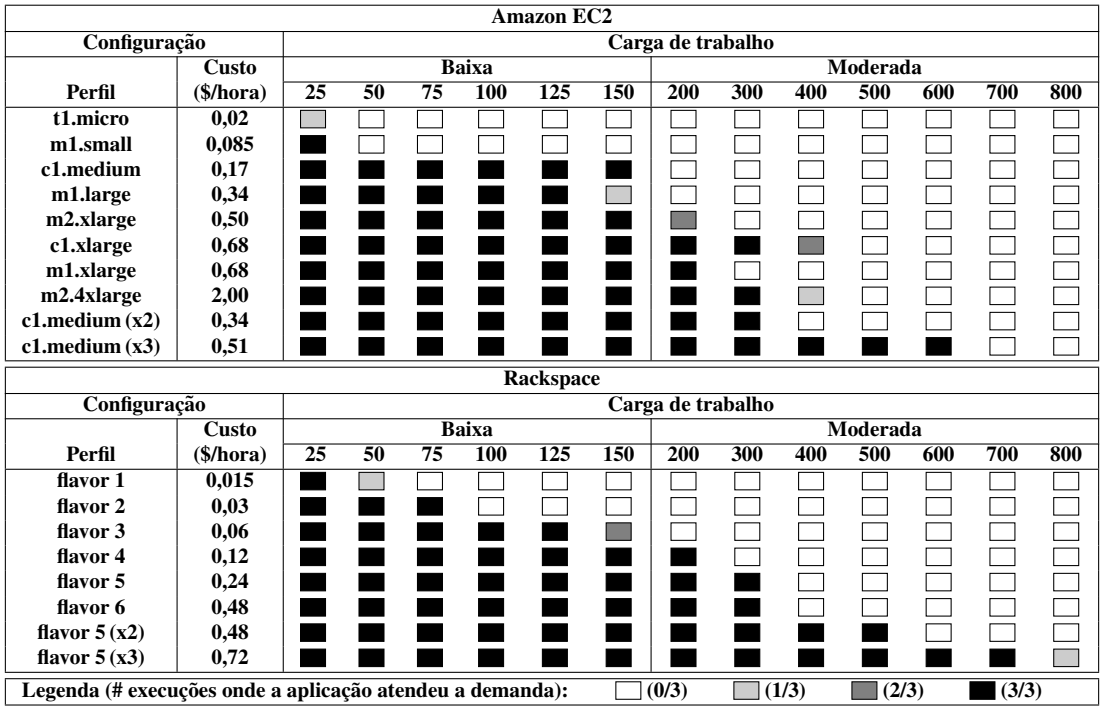

Tabela 3. Desempenho da aplicação Olio nas nuvens Amazon EC2 e Rackspace.

3.2. Resultados

A Tabela 3 mostra uma visão consolidada dos resultados coletados pelo *Crawler* após a execução dos diversos cenários de avaliação da Olio nos dois provedores.

Pelos resultados mostrados na Tabela 3, é possível observar que há diferenças significativas entre as diversas configurações de máquinas virtuais avaliadas de cada provedor, tanto em termos de desempenho quanto de custo. Por exemplo, no provedor Amazon EC2, a configuração com uma máquina virtual de perfil *c1.medium* apresenta um desempenho comparável (conseguindo ser até ligeiramente superior) ao da configuração com uma máquina virtual de perfil *m1.large* para níveis baixos de demanda, embora a segunda tenha o dobro do custo da primeira. Outra constatação com respeito a esse provedor é que configurações com preços similares podem apresentar desempenho bastante diferentes, como é o caso das configurações formadas pelas máquinas virtuais de perfil *c1.xlarge* e *m1.xlarge*, respectivamente, onde a primeira apresenta um desempenho nitidamente superior ao da segunda para níveis moderados de demanda. No caso do provedor Rackspace, as diferenças de desempenho entre as configurações foram, até certo ponto, proporcionais às suas diferenças em termos de custo, com configurações mais caras oferecendo desempenhos gradualmente superiores ao das configurações mais baratas. A exceção foi a configuração com uma máquina virtual de perfil *flavor 6*, que manteve o nível de desempenho da configuração anterior, embora custando o dobro do preço.

O uso de escalabilidade horizontal mostrou-se uma alternativa de implantação bastante atraente em ambos os provedores. No provedor Amazon EC2, por exemplo, a configuração com três máquinas virtuais de perfil *c1.medium* consegue atender uma demanda de até 600 usuários concorrentes, o que não é possível com nenhuma outra configuração, a um custo muito mais baixo do que o da maioria das configurações avali-

adas. Um resultado parecido foi observado no provedor Rackspace, onde a configuração com três máquinas virtuais de perfil *flavor 5* consegue atender uma demanda de até 800 usuários concorrentes, representando um ganho de quase 170% sobre a demanda máxima atendida pela configuração de maior preço formada por uma única máquina virtual.

4. Trabalhos Relacionados

Até o momento, poucos trabalhos foram publicados descrevendo soluções automatizadas para apoiar os usuários de nuvens IaaS na realização de seus próprios testes de desempenho. Em [Snellman et al. 2011], os autores propõem um gerador de carga que usa recursos de nuvens IaaS para a realização de testes de desempenho de aplicações web. Esse trabalho, porém, não oferece suporte automatizado para variar a arquitetura de implantação das aplicações sendo avaliadas. Um outro trabalho recente foi descrito em [Jayasinghe et al. 2012], onde os autores propõem um ambiente para execução automática de testes de desempenho em nuvens IaaS. Nesse ambiente, chamado Expertus, os usuários implementam testes de desempenho através da geração e customização de *scripts* Shell a partir de *templates* especificados na forma de documentos XML [Jayasinghe et al. 2012]. A solução adotada no Expertus, por envolver o uso de linguagens imperativas, ainda exige um alto grau de conhecimento sobre comandos básicos de configuração de sistemas por parte dos usuário. Além disso, a especificação dos testes em uma sintaxe baseada em *scripts* Shell pode acarretar problemas de portabilidade, dificultando a sua execução e reutilização em máquinas virtuais configuradas com diferentes linguagens de comando ou diferentes sistemas operacionais.

Para facilitar a especificação, execução e reutilização de testes de desempenho em diferentes tipos de máquinas virtuais e sistemas operacionais, este trabalho propõe uma nova linguagem declarativa para a descrição, em um alto nível de abstração, das entidades (componentes da aplicação, parâmetros de configuração, níveis de demanda, métricas de desempenho, perfis de recursos virtuais, etc.) que compõem um cenário de avaliação de desempenho na nuvem. Dessa forma, cabe ao motor de execução da linguagem interpretar, configurar e executar os cenários descritos pelos usuários na infraestrutura virtual do provedor de nuvem, contribuindo, assim, para uma drástica redução da complexidade e do esforço tradicionalmente envolvidos neste tipo de atividade.

5. Conclusão

Este trabalho apresentou o ambiente *Cloud Crawler*, que tem com objetivo facilitar a execução de testes de desempenho de aplicações em provedores de nuvens IaaS. O ambiente oferece um conjunto de tecnologias que permitem especificar, através de uma notação de alto nível de abstração, diferentes cenários de avaliação de desempenho na nuvem, os quais são automaticamente executados pelo ambiente. Conforme foi demonstrado através dos dois experimentos descritos no artigo, o ambiente constitui-se em uma importante ferramenta de planejamento e alocação de recursos em nuvens IaaS, podendo contribuir de forma significativa para um melhor aproveitamento dos recursos da nuvem por parte dos usuários, não apenas em termos de desempenho mas também de custo.

Atualmente, há uma série de linhas de pesquisa sendo consideradas visando melhorar e estender as funcionalidade do ambiente. Entre elas, destacam-se: o suporte à implantação automática dos componentes da aplicação na nuvem; a inclusão de meca-

nismos para permitir o controle e o monitoramento do fluxo de trabalho do motor de execução; e a implementação de uma ferramenta gráfica de edição e geração de cenários.

Agradecimentos

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

Referências

Armbrust, M. et al. (2010). A view of cloud computing. *CACM*, 53(4):50–58.

Ben-Kiki, O. et al. (2001). YAML Ain't Markup Language (YAML) Version 1.1. 11.

Cunha, M. (2012). Um Ambiente Programável para Avaliar o Desempenho de Aplicações em Nuvens de Infraestrutura. Master's thesis, Universidade de Fortaleza.

Cunha, M. et al. (2011). Investigating the Impact of Deployment Configuration and User Demand on a Social Network Application in the Amazon EC2 Cloud. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 746–751. IEEE Computer Society.

Fonseca, R. and Simões, A. (2007). Alternativas ao XML: YAML e JSON. In *5ª Conferência Nacional em XML, Aplicações e Tecnologias Aplicadas (XATA 2007)*.

Jayasinghe, D. et al. (2011). Variations in Performance and Scalability When Migrating n-Tier Applications to Different Clouds. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 73–80. IEEE Computer Society.

Jayasinghe, D. et al. (2012). Expertus: A Generator Approach to Automate Performance Testing in IaaS Clouds. In *Cloud Computing (CLOUD), 2012 IEEE International Conference on*, pages 73–80.

JSON (2011). JavaScript Object Notation. `http://www.json.org/`.

Li, A. et al. (2010). CloudCmp: Comparing Public Cloud Providers. In *Internet Measurement Conference*.

Malkowski, S. et al. (2010). CloudXplor: A tool for configuration planning in clouds based on empirical data. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 391–398. ACM.

Ostermann, S. et al. (2010). A performance analysis of EC2 cloud computing services for scientific computing. *Cloud Computing*, pages 115–131.

Rhoton, J. (2009). *Cloud Computing Explained: Handbook for Enterprise Implementation*. Recursive Ltd.

Snellman, N., Ashraf, A., and Porres, I. (2011). Towards Automatic Performance and Scalability Testing of Rich Internet Applications in the Cloud. In *Software Engineering and Advanced Applications, 2011 37th EUROMICRO Conference on*.

Sobel, W. et al. (2008). Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0. In *Proc. of CCA*. Citeseer.

Zhang, Q. et al. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.